Chapitre 6

Le déploiement continu (Continuous Deployment CD)

Enseignante: Dr-Ing. Amina JARRAYA
Email : amina.jarraya@ensi-uma.tn
Niveau: II3- GL

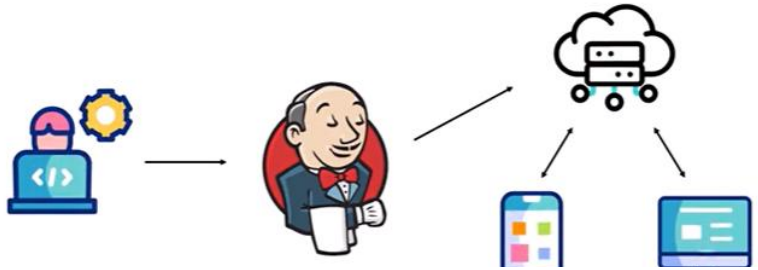
Plan

1. Introduction au déploiement continu (CD)
2. La virtualisation et la conteneurisation
3. Présentation du Docker
4. Installation et utilisation du Docker sous WSL
5. Création d’un pipeline CI/CD avec Docker

1. Introduction au déploiement Continue (CD)

Introduction au déploiement continu

Le déploiement continu qui automatise le processus de déploiement des modifications de code dans un environnement de production. En d'autres termes, chaque modification qui passe les tests automatisés est déployée automatiquement sans intervention humaine.



Necessary Plugins for Continuous Deployment:

Pipeline

SSH

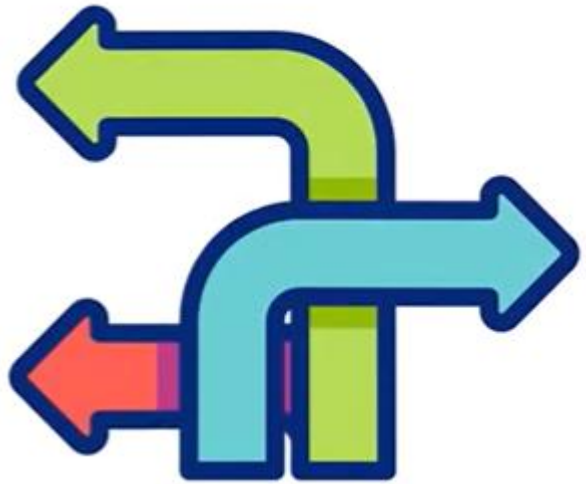
Docker

Cloud

Stratégies de déploiement

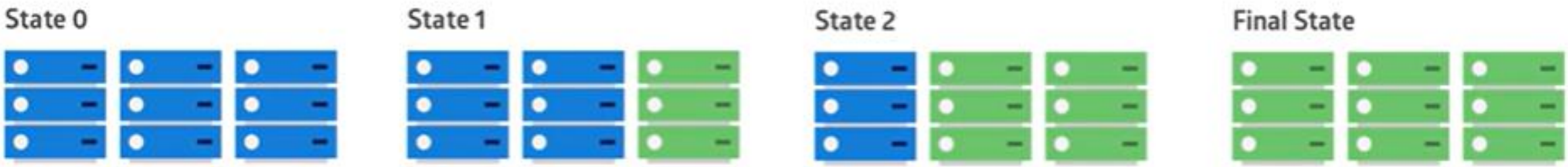
- Il existe trois types de stratégies de déploiement :

1. **Le déploiement progressif (rolling deployment)**
2. **La stratégie de déploiement blue-green, ou bleu-vert**
3. **Le déploiement canari**



Stratégies de déploiement – rolling deployement

- **Le déploiement progressif (rolling deployment),** aussi appelé déploiement par vagues, est une stratégie où une nouvelle version d'une application est déployée progressivement, remplaçant les anciennes instances **une par une, sans interruption du service**. C'est une méthode courante dans les environnements Kubernetes et permet de minimiser les risques liés au déploiement.
  - Déployer une nouvelle version d'une application petit à petit, **serveur par serveur ou groupe d'utilisateurs par groupe d'utilisateurs,** sans interruption de service.



Stratégies de déploiement – rolling deployement

- **Le déploiement progressif (rolling deployment)**

- L'objectif est :
  - de **réduire les risques** liés à un nouveau déploiement,
  - de **tester la nouvelle version en conditions réelles**,
  - et de **garantir la disponibilité** du service.

- **Exemple** : Imaginez que votre application tourne sur **5 serveurs Tomcat** derrière un **load balancer**.

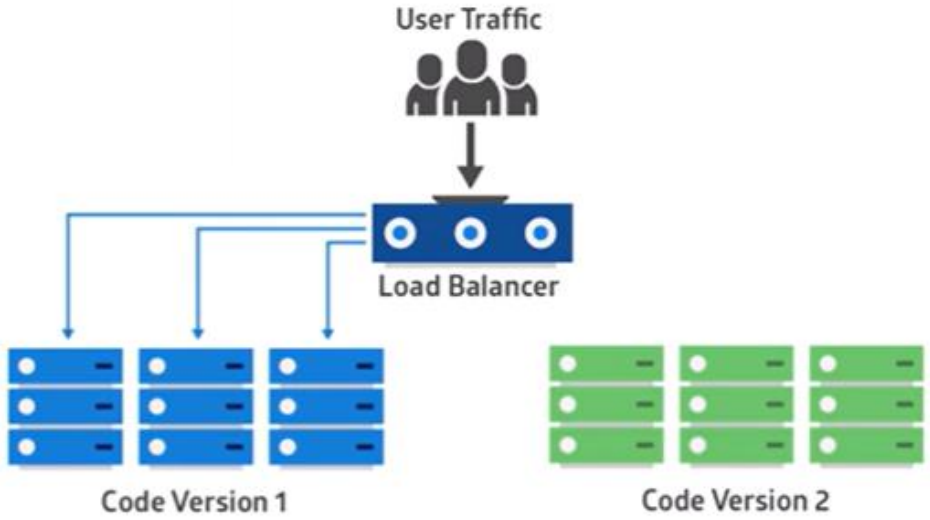
Étapes du rolling deployment :

- Tu as la version actuelle **v1.0** déployée sur tous les serveurs.
- Tu déploies la nouvelle version **v2.0** sur **1 seul serveur à la fois** :
  - Le serveur 1 est mis hors du load balancer.
  - On déploie v2.0 dessus.
  - On le remet en ligne, puis on passe au suivant.
- Quand tous les serveurs sont mis à jour → la version **v2.0** est totalement déployée.
- Si un problème est détecté, tu peux **arrêter le déploiement** et revenir à v1.0.

➢ **Ainsi, les utilisateurs ne voient jamais d'interruption complète.**

Stratégies de déploiement - blue-green deployement

- **La stratégie de déploiement blue-green, ou bleu-vert,** est une méthode qui permet de minimiser les interruptions de service lors de la mise à jour d'une application.
- Il consiste à maintenir **deux environnements de production identiques** :
- 🟦 **Blue** = environnement actuellement en production
- 🟩 **Green** = environnement de la nouvelle version (pré-production)
- Quand la **nouvelle version est prête et validée**, on **bascule le trafic** du **Blue vers le Green**. Ainsi, les utilisateurs passent instantanément sur la nouvelle version
- En cas de problème, on rebascule facilement vers l'ancienne version (Blue with rollback)



Stratégies de déploiement - blue-green deployement

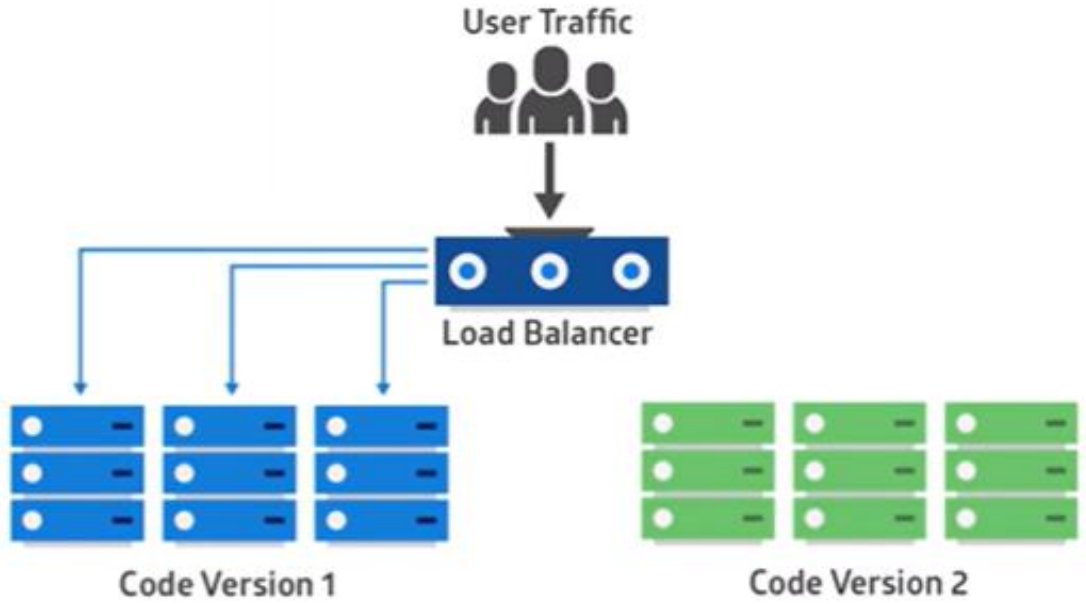
- Étape 1 — Avant déploiement :
- [Blue Environment] ← Production traffic
- [Green Environment] ← Vide

- Étape 2 — Déploiement :
- Déployer la nouvelle version sur Green

- Étape 3 — Validation :
- Tester Green → tout fonctionne ✅

- Étape 4 — Bascule :
- Load Balancer redirige le trafic vers Green

- Étape 5 — Après bascule :
- [Blue] devient inactif (rollback possible)
- [Green] devient la production active

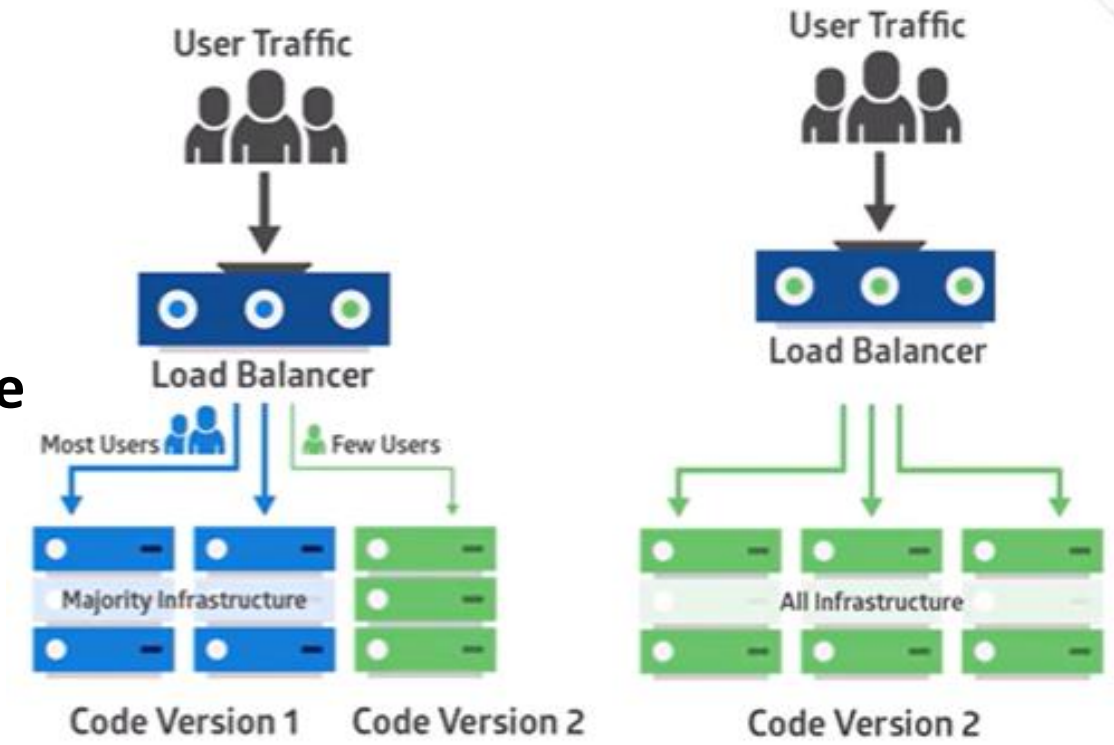


Stratégies de déploiement – canari deployement

- **Le déploiement canari** consiste à **déployer une nouvelle version de ton application (v2.0)** à **un petit pourcentage d'utilisateurs**, pendant que la majorité reste sur **l'ancienne version (v1.0)**.

- Si tout fonctionne bien (aucune erreur, pas de régression), on **augmente progressivement le pourcentage d'utilisateurs** jusqu'à ce que **toute la population utilise la nouvelle version**.

➢ Le nom "**canari**" vient du **"canari dans la mine de charbon"** : les mineurs utilisaient un petit oiseau pour détecter les gaz dangereux avant que les humains n'entrent.
Ici, les **utilisateurs canaris** "testent" la nouvelle version avant tout le monde.



Stratégies de déploiement – canari deployement

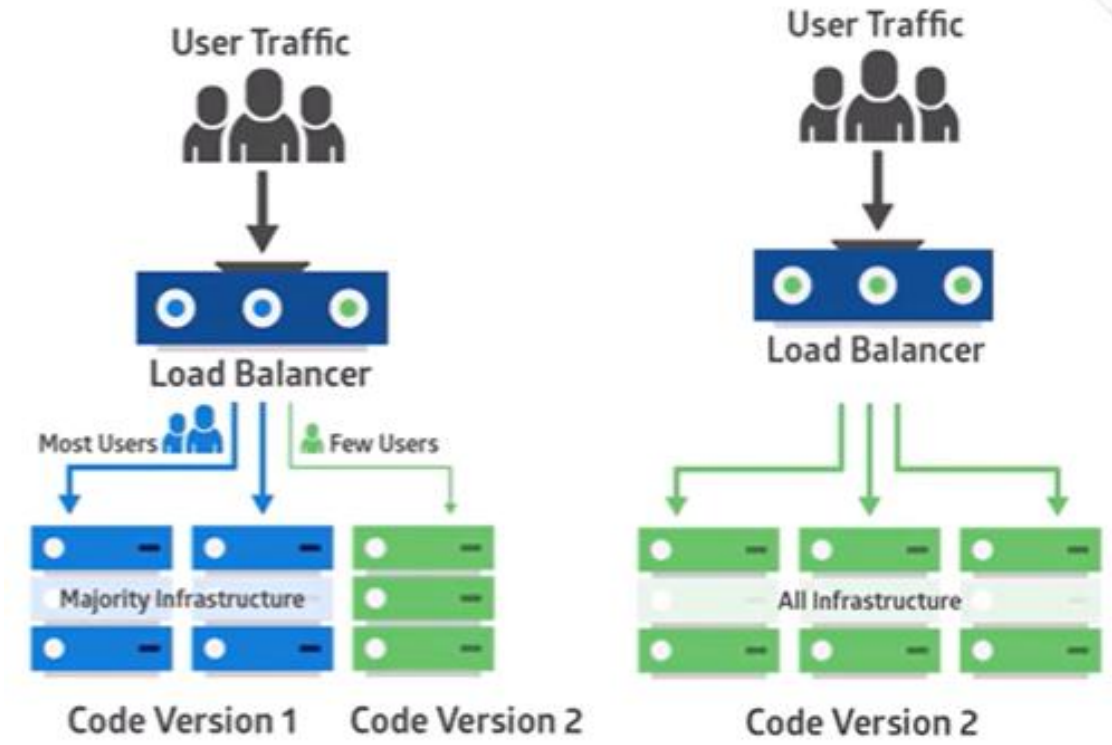
- Étape 1 — Déploiement initial :
- 100% des utilisateurs → v1.0

- Étape 2 — Déploiement canari :
- 90% des utilisateurs → v1.0
- 10% des utilisateurs → v2.0

- Étape 3 — Observation :
- Monitoring et logs : erreurs, latence, feedback

- Étape 4 — Expansion :
- 50% → v2.0, 50% → v1.0

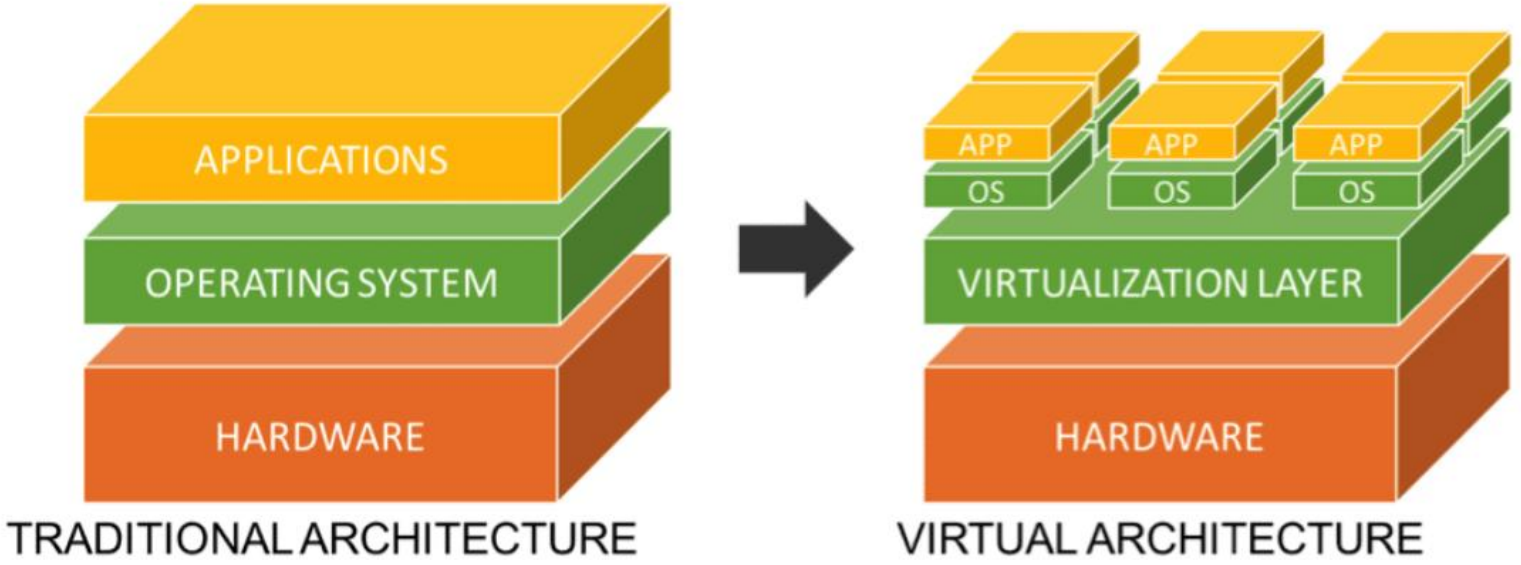
- Étape 5 — Finalisation :
- 100% → v2.0
- Rollback rapide vers v1.0 si anomalie



2. La virtualisation et la conteneurisation

La virtualisation

- **La virtualisation** est une technologie qui permet de créer des versions virtuelles de ressources informatiques (serveurs, stockage, réseaux, etc.) à partir d'une seule machine physique.
- Cela se fait grâce à un logiciel appelé **hyperviseur**, qui divise les ressources du matériel physique en plusieurs machines virtuelles. Ces machines virtuelles fonctionnent de manière indépendante et peuvent exécuter différents systèmes d'exploitation et applications, améliorant ainsi l'efficacité et la flexibilité de l'infrastructure informatique.

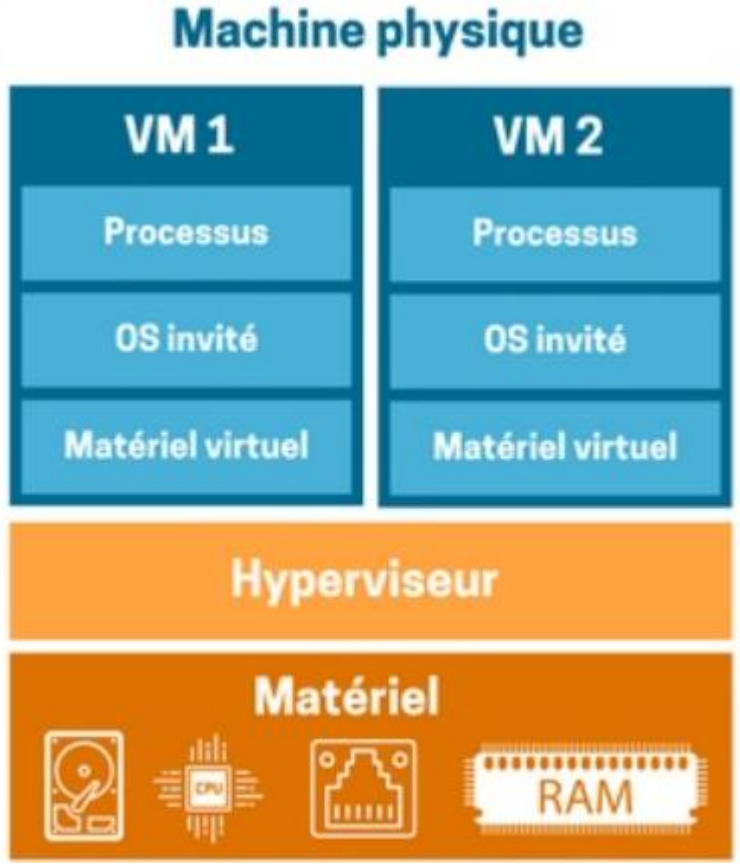


Définition d’un hyperviseur

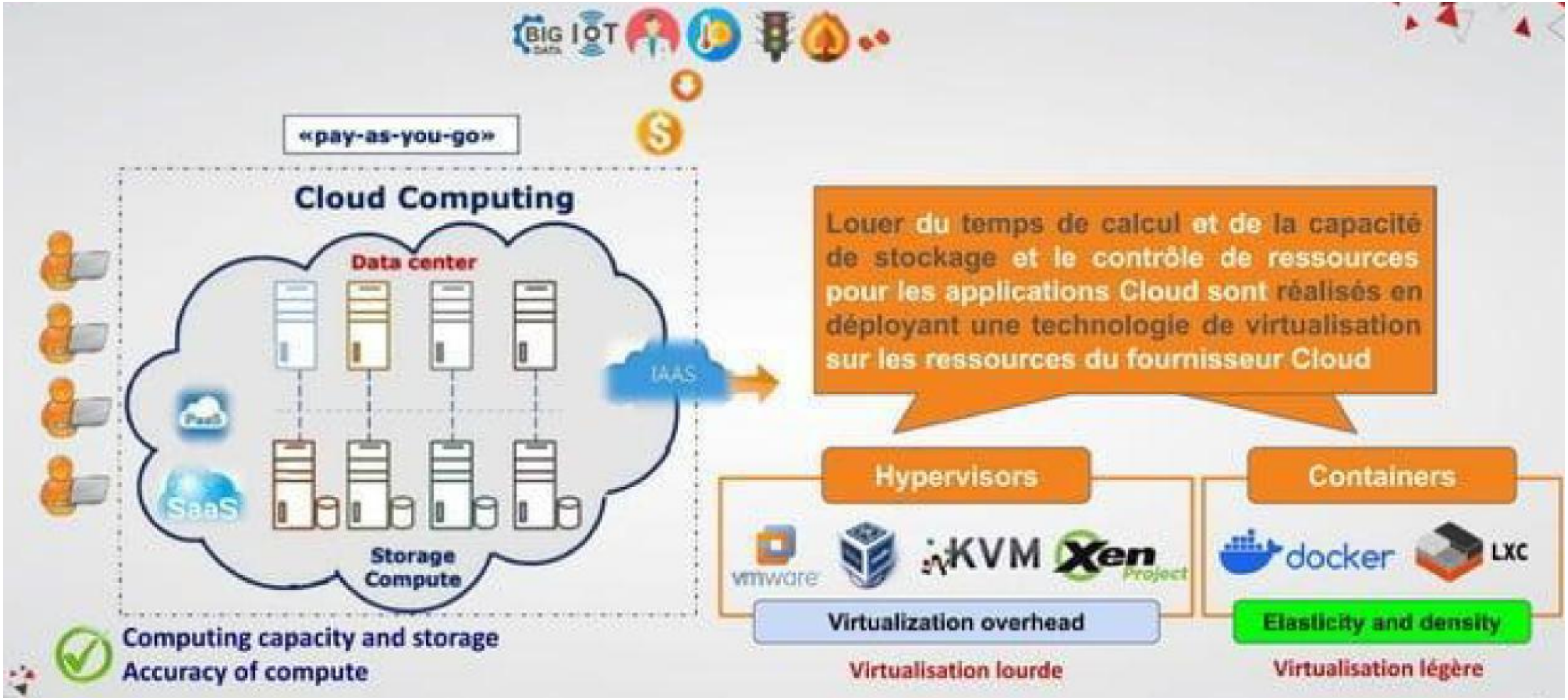
- **Un hyperviseur** est un logiciel qui permet de créer et de gérer des machines virtuelles (VM) sur un seul serveur physique. Il agit comme une couche d'abstraction entre le matériel physique et les systèmes d'exploitation et applications invités, permettant à plusieurs VM de partager les ressources du serveur hôte.

- **Exemples d'hyperviseurs:**

VMware ESXi, Microsoft Hyper-V, KVM, Xen.



La virtualisation



La virtualisation lourde

- **La virtualisation lourde**, aussi appelée **virtualisation à base d'hyperviseur**, est une technologie qui permet de faire fonctionner plusieurs systèmes d'exploitation sur un seul ordinateur physique en utilisant un hyperviseur directement sur le matériel.

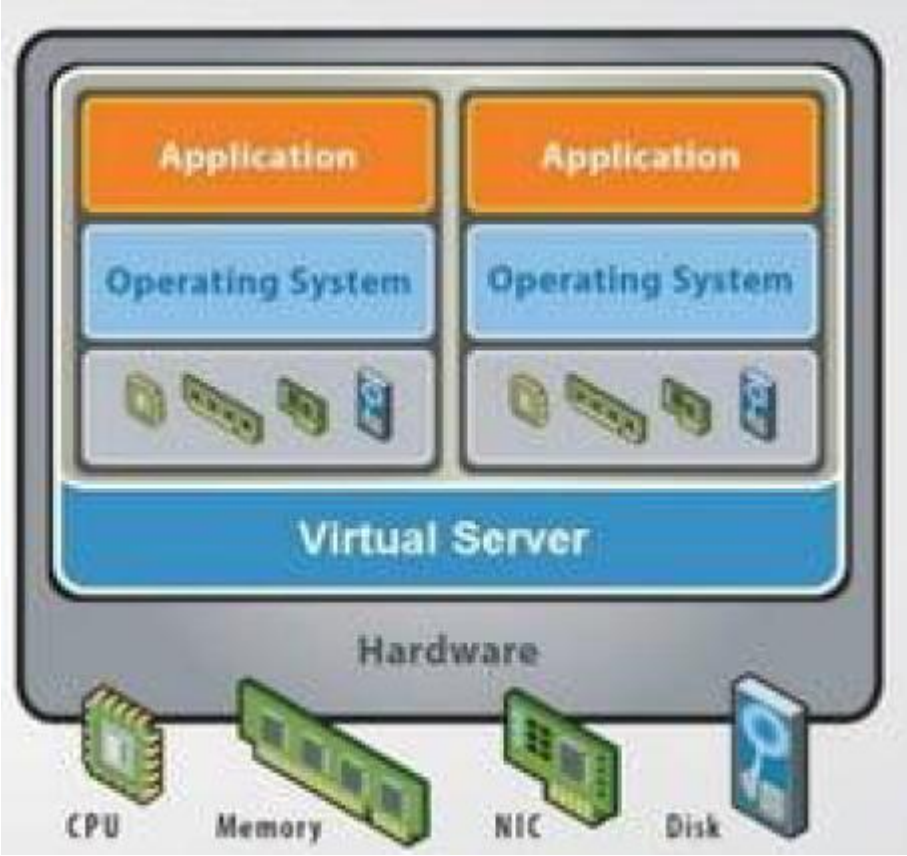
- **Simulation des machines physiques** : elle permet de simuler une ou plusieurs machines physiques
- **Exécution sous forme de VM** : ces VMs s'exécutent sur un serveur.
- **OS intégré** : chaque VM intègre son système d'exploitation.
- **Exécution d'application** : les applications sont exécutées à l'intérieur de ces machines virtuelles

Avantages de a virtualisation lourde

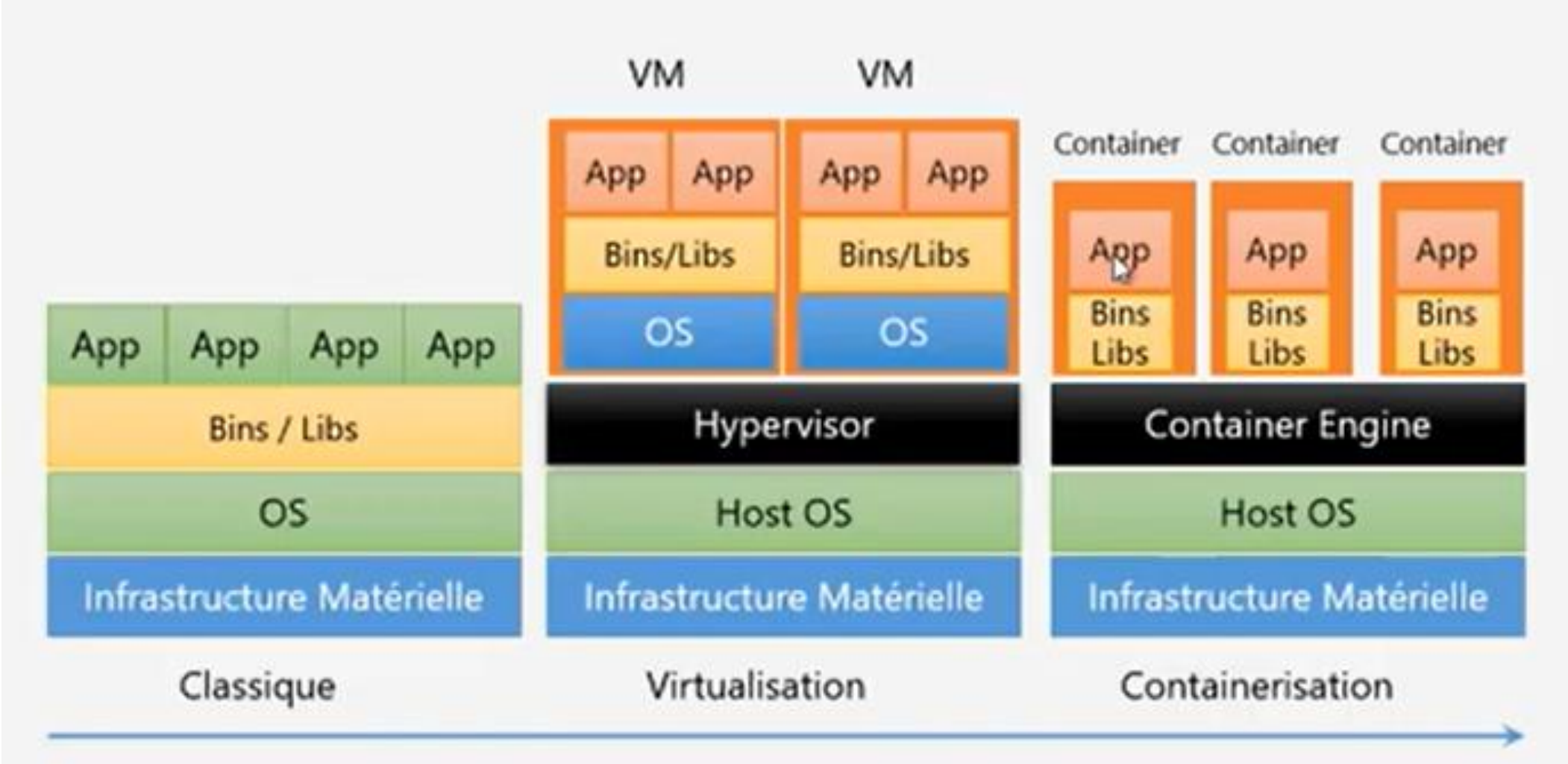
- **Réduction des coûts:** La virtualisation permet de consolider plusieurs serveurs physiques en un seul, réduisant ainsi les coûts d'achat, de maintenance et d'énergie.
- **Amélioration de l'efficacité des ressources:** En utilisant plus efficacement les ressources matérielles disponibles, la virtualisation optimise l'utilisation des serveurs et des équipements.
- **Flexibilité et agilité:** La virtualisation permet de créer et de déployer rapidement de nouvelles machines virtuelles, s'adaptant ainsi aux besoins changeants de l'entreprise.
- **Gestion simplifiée:** La virtualisation offre une gestion centralisée des machines virtuelles, facilitant la maintenance, la sauvegarde et la restauration des systèmes.
- **Haute disponibilité et reprise après sinistre:** La possibilité de migrer rapidement les machines virtuelles d'un serveur à un autre (migration à chaud) assure une continuité des services en cas de panne.
- **Meilleure sécurité:** L'isolement des machines virtuelles permet de mieux contrôler les accès et de réduire les risques de propagation des menaces.

Limites de la virtualisation lourde

- Chaque VM a besoin de ressources (CPU, stockage (disque), RAM, OS invité)
- En augmentant les VMs, on demande plus de ressources :
  - Chaque OS invité alloue ses propres ressources donc c'est du gaspillage
  - Portabilité des applications n'est pas garantie



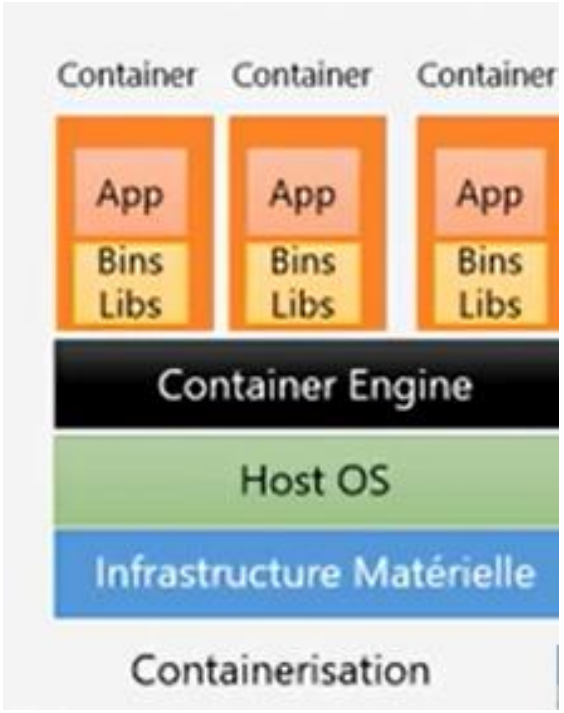
Evolution de l'infrastructure



La virtualisation légère : la conteneurisation

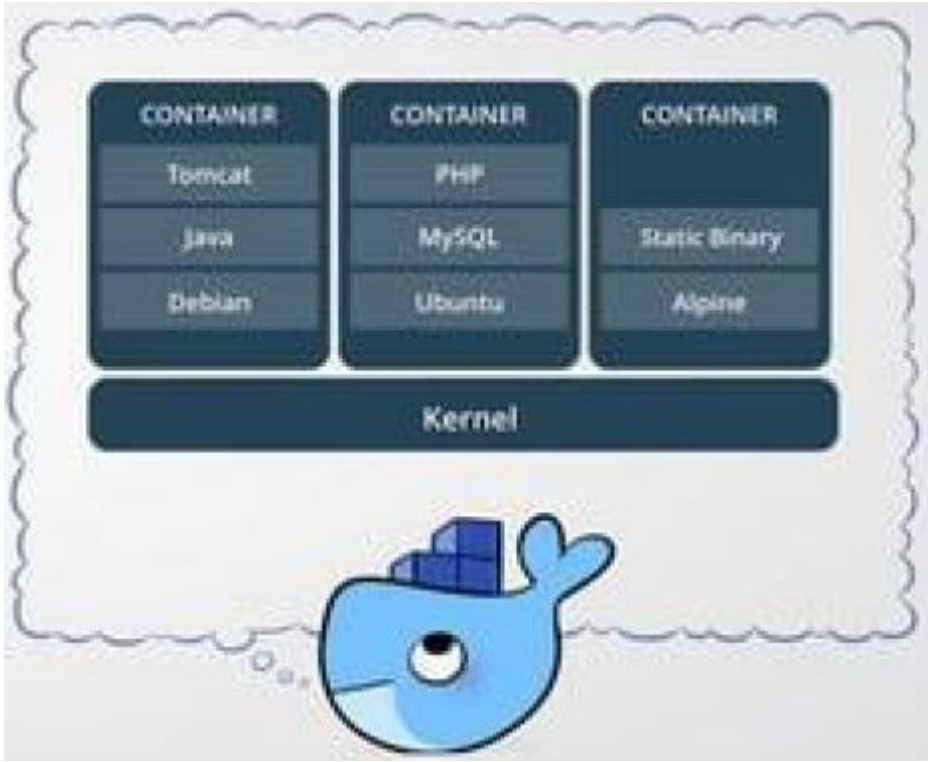
- La conteneurisation est une technique qui permet d'empaqueter une application et ses dépendances dans un environnement isolé, appelé **conteneur**, pour assurer une exécution fiable et uniforme sur différentes plateformes.

- Docker, en particulier, est une plateforme qui facilite la création, le déploiement et la gestion de ces conteneurs.



Avantages de la conteneurisation

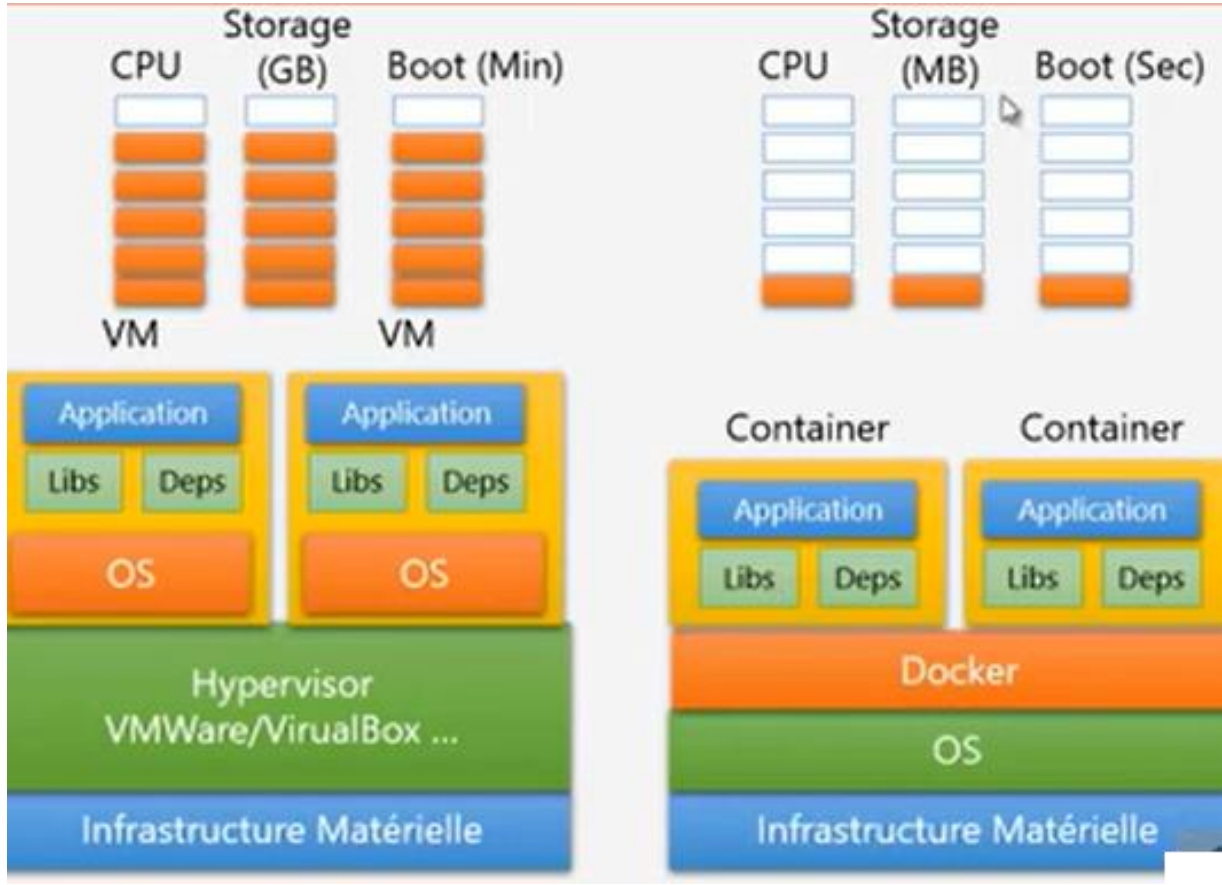
- **Isoler les applications**: Chaque conteneur fonctionne indépendamment des autres, avec son propre système de fichiers, ses librairies et ses dépendances.
- **Augmenter la portabilité:** Les conteneurs, grâce à leur format standardisé, peuvent être déployés et exécutés sur différentes machines ou environnements (développement, test, production) sans modification, ce qui facilite le développement et le déploiement d'applications.
- **Améliorer l'efficacité:** Les conteneurs sont plus légers que les machines virtuelles traditionnelles car ils partagent le noyau du système d'exploitation hôte, ce qui permet de gagner en rapidité et en ressources.
- **Faciliter la gestion et l'orchestration:** Des outils comme Docker permettent de gérer facilement les conteneurs, de les démarrer, les arrêter, de les mettre à l'échelle et de les interconnecter, ce qui simplifie la gestion des applications complexes.



Conteneurisation vs virtualisation

- Machine virtuelle :
  - Permet de virtualiser une machine physique.
  - Chaque VM a son propre OS
  - Une VM consomme bcp de ressources (CPU, Stockage) et prend assez de temps pour booster (qq minutes).

- Conteneur :
  - Permet de créer un environnement d'exécution des applications
  - Les conteneurs utilisent le même OS
  - Tous les conteneurs utilisent le même kernel OS (Linux), consomme peu de ressources, boot rapide (qq secondes)

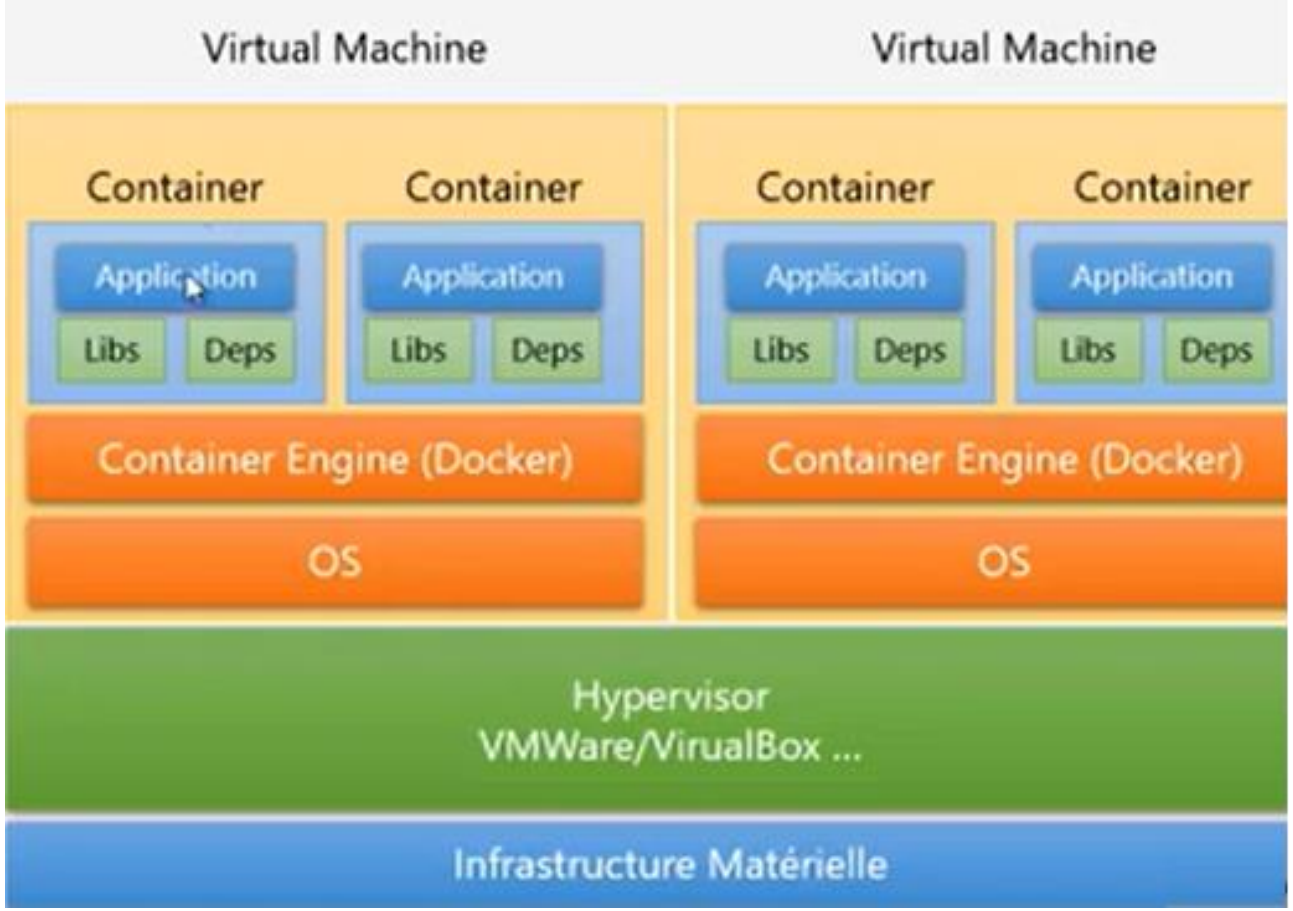


Conteneurisation vs virtualisation

Virtualisation	Conteneurisation
Une virtualisation lourde	Une virtualisation légère
À base d'hyperviseur	À base des conteneurs
Virtualisation des ressources **hardware** (CPU, RAM, disque, …)	Virtualisation au niveau du système d'exploitation
Machine invité (machine virtuelle)	Conteneur
Image ISO	Librairies nécessaires
Machine hôte	Moteur de conteneur
Démarrage plus lent	Démarrage en quelques secondes
Entièrement isolé et donc plus sécurisé	Isolation au niveau du processus et donc moins sécurisée

Utiliser des conteneurs dans des machines virtuelles

- Docker ne vient pas pour remplacer les machines virtuelles.
- Dans la pratique on utilise les deux :
  - Les VMs pour virtualiser les machines
  - Utiliser Docker pour isoler les environnements d'exécution des applications dans des machines virtuelles
- Ceci pour tirer le bénéfice des technologies.



3. Présentation du docker

C’est quoi Docker ?

- **Docker** est une plateforme logicielle open source , créé par ***Solomon Hykes et développé par docker, Inc.*** qui permet de créer, déployer et gérer des applications conteneurisées.

- Les **conteneurs**, créés avec Docker, regroupent tout ce dont une application a besoin pour fonctionner : code, bibliothèques, outils système et runtime. Cela permet une exécution cohérente et fiable des applications, quel que soit l'environnement.

- En d'autres termes, Docker facilite la création d'environnements d'exécution standardisés et isolés pour les applications, appelés **conteneurs**. Ces conteneurs garantissent que l'application se comportera de la même manière, quel que soit l'endroit où elle est exécutée (sur une machine de développement, un serveur de test, ou en production).

Docker - éditions

- Docker propose deux éditions distinctes, qui offrent des fonctionnalités adaptées à différents besoins :
- **Version communauté (Community Edition : CE)** : pour les utilisateurs individuels et les petites équipes. Elle offre les bases de Docker.
- **Version Entreprise (Enterprise Edition : EE) :** pour les grandes organisations. Elle inclut des fonctionnalités avancées et un support technique.

Docker - lexique

- **Image** : template de conteneur en read-only contenant un système de base et une application. Il contient tous les fichies et dépendances nécessaires à l'exécution d'une application

- **Conteneur** : Image exécutable d'un environnement complet incluant code, librairies, outils et configuration. **Un conteneur est une instance d'une image exécutée.**

- **Docker Hub** : dépôt public d'images mises à disposition par Docker
  - DockerHub : https://store.docker.com

- **Dockerfile** : fichier texte de description d'une image

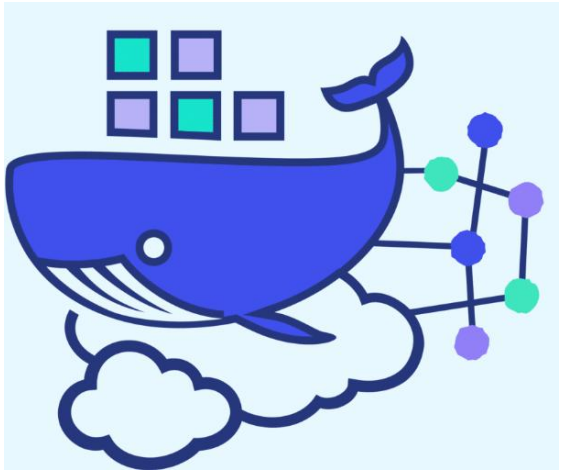
- **Docker desktop** : est une application qui simplifie la création, l'exécution et le partage d'applications conteneurisées sur les systèmes d'exploitation comme Windows, macOS et Linux. : https://docs.docker.com/desktop/

- **Registry** : Dépôt privé d'images Docker, https://hub.docker.com/_/registry

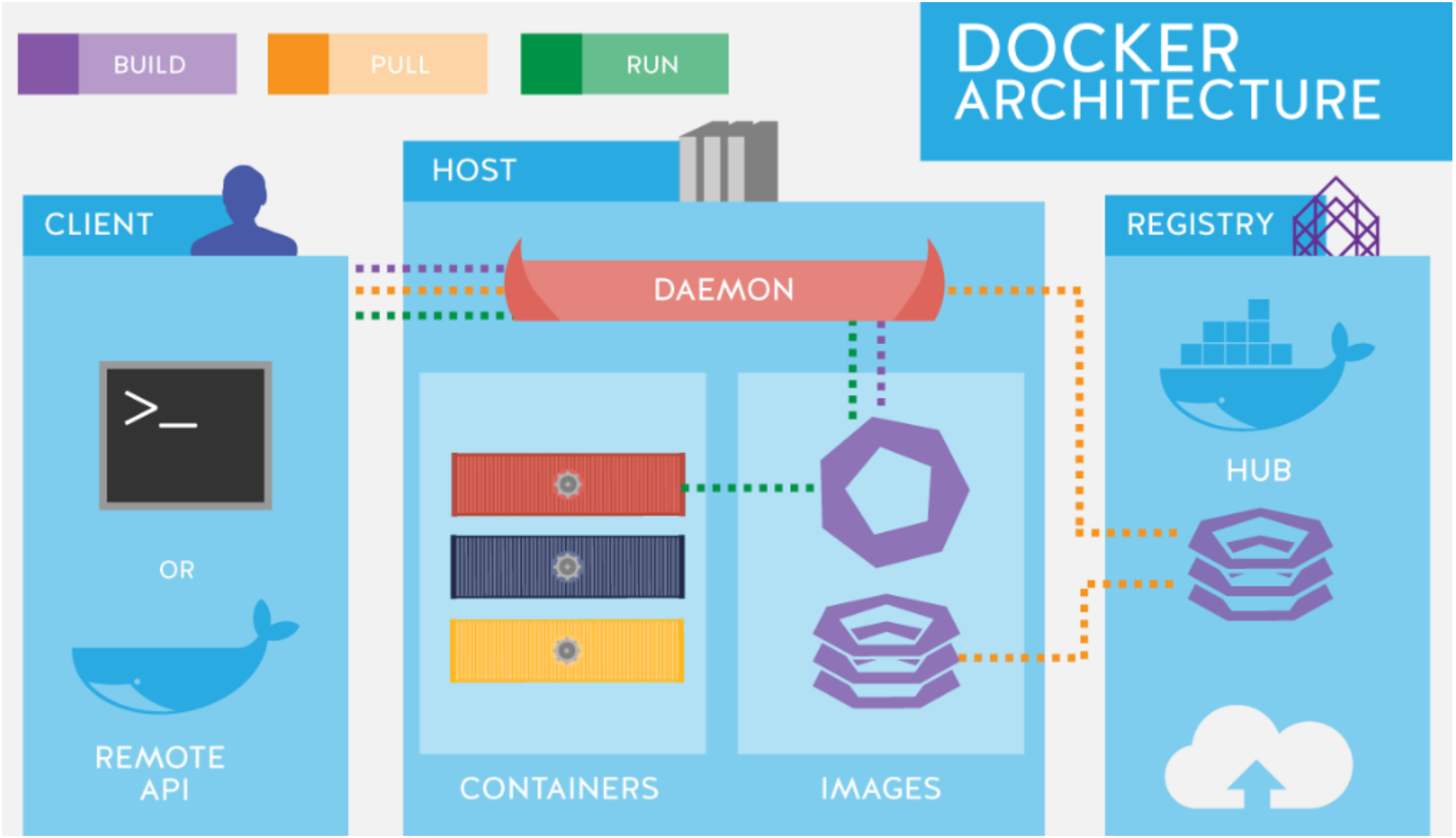
Docker engine

- Le **Docker Engine** est l'application à installer sur la machine hôte pour créer, exécuter et gérer des conteneurs Docker. Comme son nom l'indique, il s'agit du moteur du système Docker.

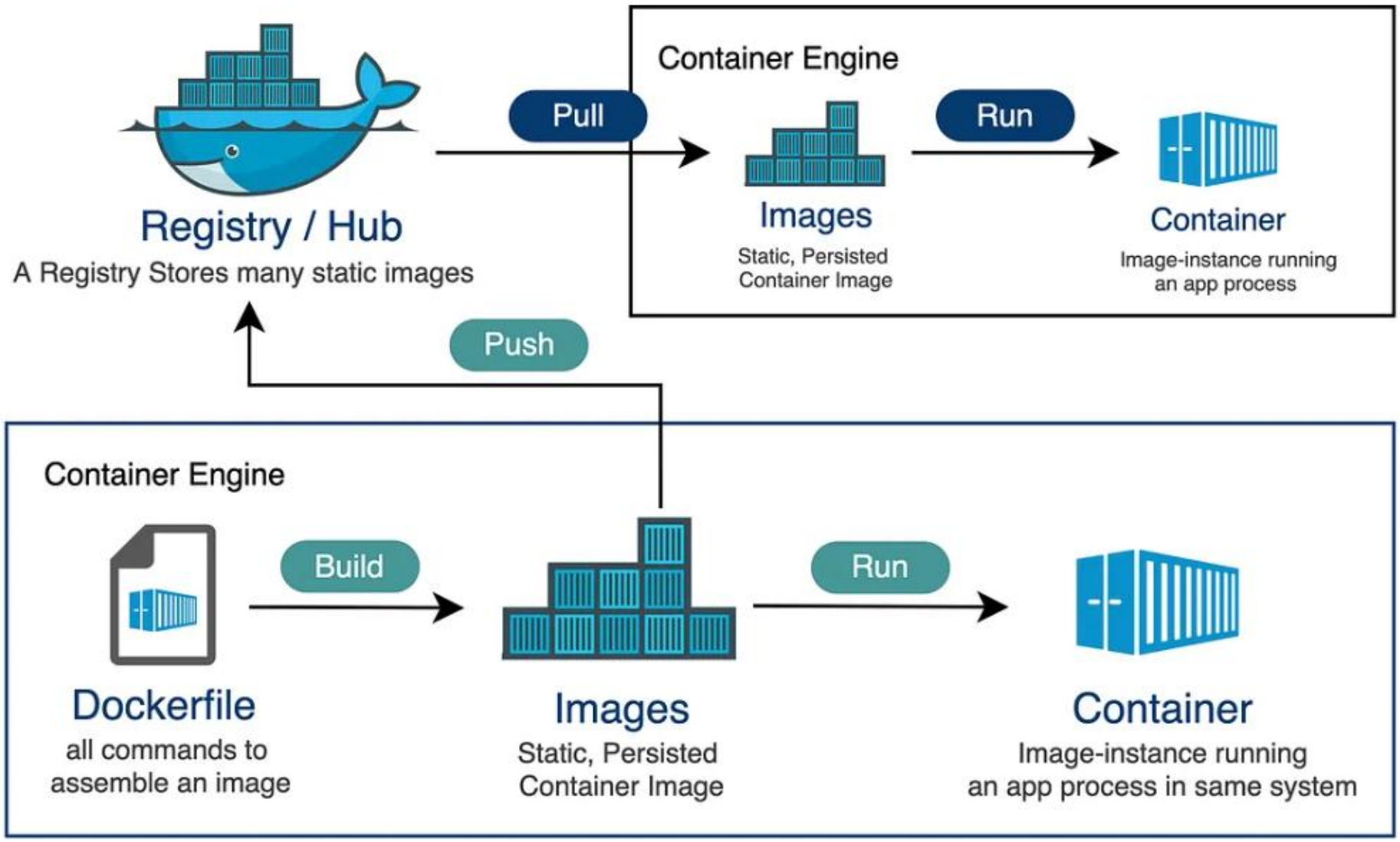
- C'est ce moteur qui regroupe et relie les différents composants entre eux. C'est la technologie client-serveur permettant de créer et d'exécuter les conteneurs, et le terme Docker est souvent employé pour désigner Docker Engine.



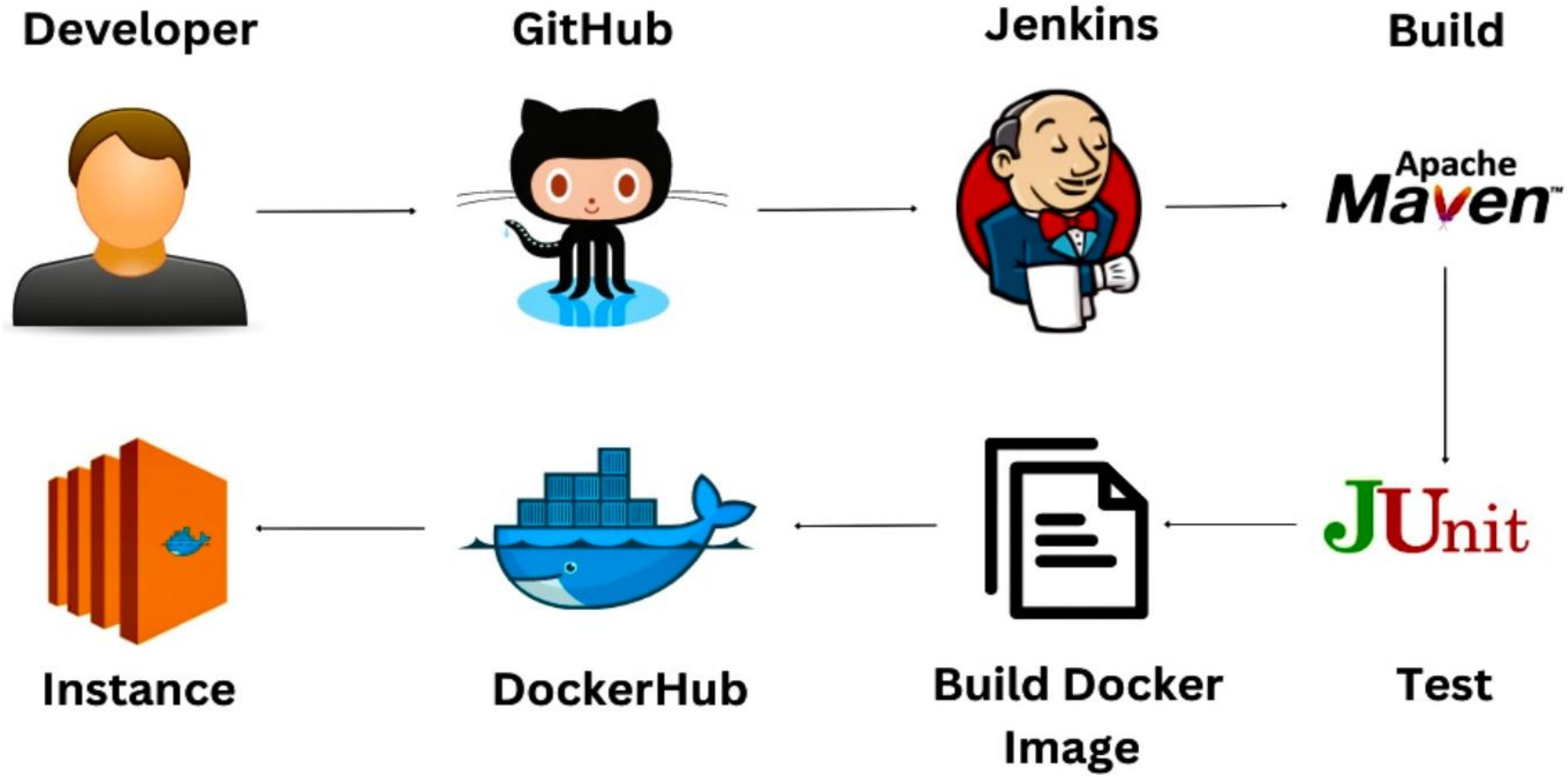
Architecture de Docker



Docker workflow et commandes



Docker dans le pipeline CI/CD



4. Installation et utilisation de docker sous WSL

Installer docker sous WSL

1. **Installer Docker Desktop**: Téléchargez et installez **Docker Desktop pour Windows** (**https://docs.docker.com/desktop/setup/install/windows-install/**).
2. **Activer l'intégration WSL**: Ouvrez Docker Desktop et accédez aux paramètres (icone de roue).

Dans l'onglet "General", assurez-vous que l'option "Use the WSL 2 based engine" est cochée.
Dans l'onglet "Resources" -> "WSL INTEGRATION", sélectionnez votre distribution Linux et activez l'intégration.

3. **Vérifier l'installation**: Ouvrez une invite de commande ou PowerShell et tapez **wsl -l -v** pour vérifier la version de votre distribution WSL. Ouvrez une invite de commande (PowerShell ou invite de commandes) en mode administrateur et tapez **wsl --install** pour installer une distribution Linux si vous n'en avez pas. Redémarrez votre ordinateur après l'installation de WSL.
4. **Utiliser Docker dans WSL**: Ouvrez votre distribution WSL (par exemple, Ubuntu).

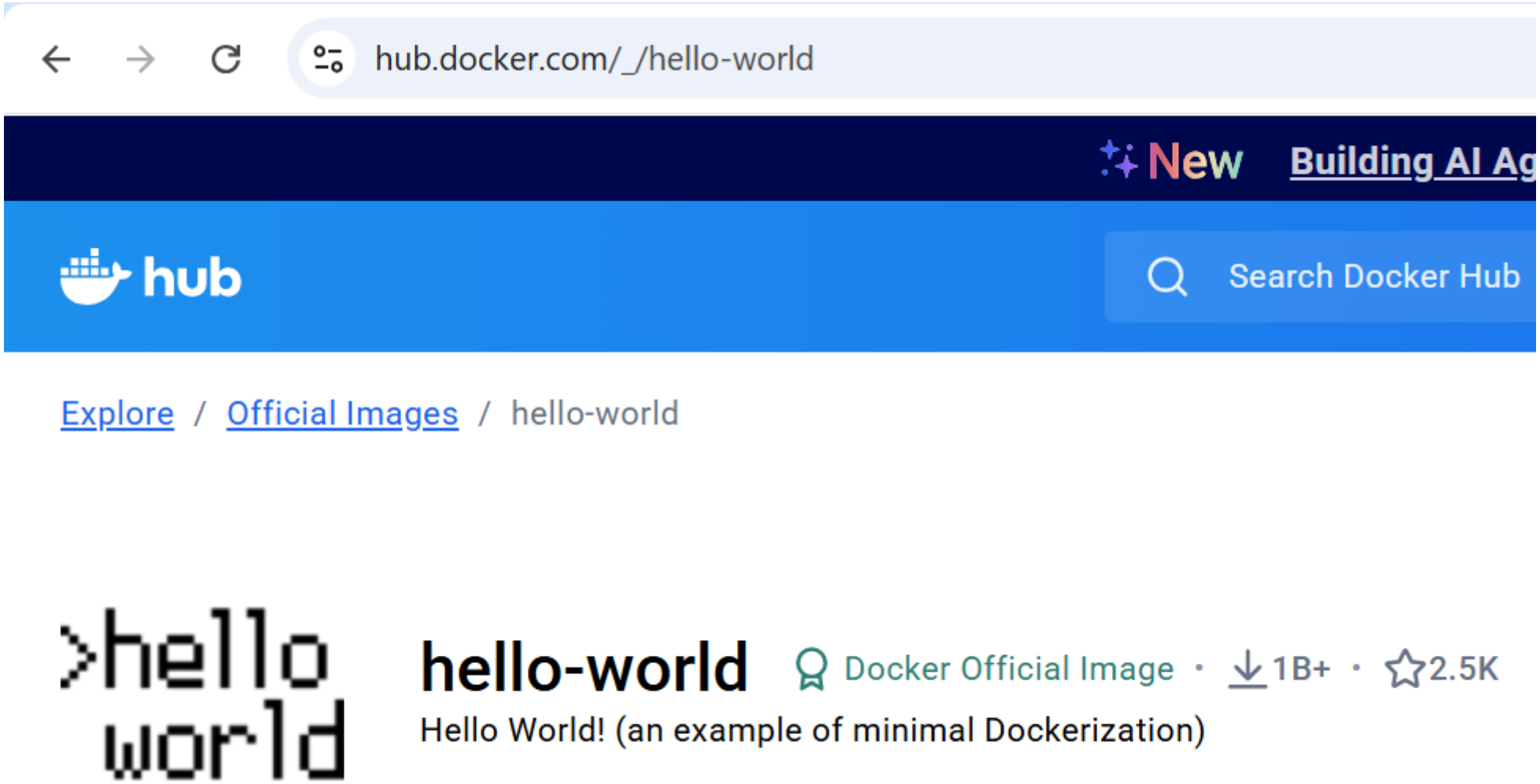
Docker Desktop devrait être accessible depuis votre distribution WSL.
Vous pouvez maintenant utiliser les commandes Docker comme docker **run hello-world**.

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker --version
Docker version 28.0.4, build b8034c0
```

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker run hello-world

Hello from Docker!
```

L’image Hello world sur DockerHub



Le fichier Dockerfile

- Un **Dockerfile** est un fichier texte contenant les instructions pour construire une image Docker. Il sert de script pour automatiser la création d'images, permettant de définir l'environnement et les dépendances nécessaires pour exécuter une application dans un conteneur.

- Exemples d'instructions :
  - **FROM** : Définit l'image de base à partir de laquelle construire l'image.
  - **Volume** : indique l'endroit ou les données vont être stockées
  - **COPY** ou **ADD** : Copie des fichiers ou des répertoires du système hôte vers le conteneur.
  - **EXPOSE** : Indique les ports du conteneur qui doivent être accessibles de l'extérieur.
  - **CMD** ou **ENTRYPOINT** : Définit la commande à exécuter lorsque le conteneur démarre.
-

```
FROM openjdk:17-oracle
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Création et affichage d'une image

- Pour créer une image Docker, vous devez utiliser la commande **docker build**. Cette commande analyse le fichier **Dockerfile** qui contient les instructions pour construire l'image, puis crée l'image en suivant ces instructions.

Code

```
docker build -t mon-image:latest .
```

- **docker build** lance la construction de l'image.
- **-t mon-image:latest** tag l'image avec le nom "mon-image" et le tag "latest".
- « . » indique que le contexte de construction est le répertoire courant.

```
C:\Users\jarra\eclipse-workspace\compte-service>docker build . -t my-compte-service:v1

C:\Users\jarra\eclipse-workspace\compte-service>docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
my-compte-service   v1        b0843e9c302b   12 minutes ago   524MB
```

Extraction depuis DockerHub

- Pour extraire une image ou un référentiel d'un registre Docker, tapez la commande suivante :

```
docker pull "nom_de_l'image":"version"
```

- Si vous ne spécifiez pas la version, Docker télécharge automatiquement la dernière version.

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
9824c27679d3: Pull complete
Digest: sha256:4bcff63911fcb4448bd4fdacec207030997caf25e9bea4045fa6c8c44de311d1
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Création d'un conteneur

- Un conteneur est la version exécutable d'une image.

- Pour exécuter une image, utilisez la commande suivante :

```
docker run "nom_de_l'image"
```

docker run my-compte-service:v1

- Si docker ne trouve pas l'image localement, il la télécharge automatiquement depuis Dockerhub, puis crée le conteneur.

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker run mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
6dfd9f336bec: Download complete
afaae28bbee0: Download complete
19f77cf53703: Download complete
2ba13c6c03c3: Pulling fs layer
31c5b5c14efc: Pulling fs layer
90fd4b47993e: Pulling fs layer
3ea90c3a51a3: Pulling fs layer
6c4bb10a796d: Pulling fs layer
938fdecc6487: Download complete
468bb678f509: Download complete
```

Lister les conteneurs en cours

- Pour afficher l'ensemble des conteneurs qui sont en cours d'exécution :

docker ps

```
C:\Users\jarra\eclipse-workspace\compte-service>docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS          PORTS
   NAMES
244b2c31c167   my-compte-service:v1   "java -jar /app.jar"   25 hours ago   Up 28 seconds
   keen_aryabhata
```

- Si vous tentez d'accéder à votre application via localhost, il ne va rien afficher puisque le conteneur n'est pas lié au port 8080.

Arrêter l'exécution d'un conteneur

- Arrêter un conteneur docker

```
docker stop 244
```

- Il suffit de mettre les 3 premiers chiffres de l'id du conteneur

```
C:\Users\jarra\eclipse-workspace\compte-service>docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS          PORTS
   NAMES
244b2c31c167   my-compte-service:v1   "java -jar /app.jar"   25 hours ago   Up 28 seconds
   keen_aryabhata
```

Associer l'exécution d'un conteneur à un port

- Mapper un conteneur docker avec un port

```
docker run -p 8080:8080 my-compte-service:v1
```

```
C:\Users\jarra\eclipse-workspace\compte-service>docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS         PORTS
   NAMES
244b2c31c167   my-compte-service:v1   "java -jar /app.jar"   25 hours ago   Up 28 seconds   0.0.0.0:8080->8080/tcp
   keen_aryabhata
```

- Nous pouvons démarrer plusieurs conteneurs sur différents ports :

```
docker run -p 8081:8080 my-compte-service:v1
```

```
C:\Users\jarra\eclipse-workspace\compte-service>docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS         PORTS
  NAMES
b25573181133   my-compte-service:v1   "java -jar /app.jar"   25 hours ago   Up 5 seconds   0.0.0.0:8081->8080/tcp
  busy_hofstadter
244b2c31c167   my-compte-service:v1   "java -jar /app.jar"   25 hours ago   Up 7 minutes   0.0.0.0:8080->8080/tcp
  keen_aryabhata
```

Démarrer un conteneur en arrière plan

- Démarrer un conteneur docker en arrière plan

```
docker run -d -p 8082:8080 my-compte-service:v1
```

- Remarque :

```
docker run -p <port1>:<port2> <image>
```

➢ “Expose le **port du conteneur (port2)** sur le **port de ta machine (port1)**.”

Élément	Nom	Rôle
port1	Port **hôte**	C’est le port de **ta machine (ordinateur, serveur, VM)** sur lequel Docker écoute. C’est celui que tu utilises dans ton navigateur ou ton client HTTP.
port2	Port **conteneur**	C’est le port **à l’intérieur du conteneur**, celui sur lequel **l’application Dockerisée écoute**.

Supprimer des images

- Pour supprimer une image

```
docker image rm "nom_image" --force

docker rmi "nom_image" --force
```

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker images
REPOSITORY                                TAG       IMAGE ID       CREATED         SIZE
alpine                                    latest    4bcff63911fc   2 weeks ago     12.8MB
eclipse-workspace-my-compte-service2      latest    47e7339ff204   3 months ago    827MB
eclipse-workspace-my-compte-service1      latest    b6bfd1ef3c6f   3 months ago    827MB
hello-world                               latest    ec153840d1e6   6 months ago    20.4kB
jamina_dev@DESKTOP-SKOOCDK:~$ docker rmi alpine --force
Untagged: alpine:latest
Deleted: sha256:4bcff63911fcb4448bd4fdacec207030997caf25e9bea4045fa6c8c44de311d1
jamina_dev@DESKTOP-SKOOCDK:~$ docker images
REPOSITORY                                TAG       IMAGE ID       CREATED         SIZE
eclipse-workspace-my-compte-service1      latest    b6bfd1ef3c6f   3 months ago    827MB
eclipse-workspace-my-compte-service2      latest    47e7339ff204   3 months ago    827MB
hello-world                               latest    ec153840d1e6   6 months ago    20.4kB
```

Supprimer des conteneurs

- Pour supprimer un conteneur : `docker rm "ID_du_conteneur"`

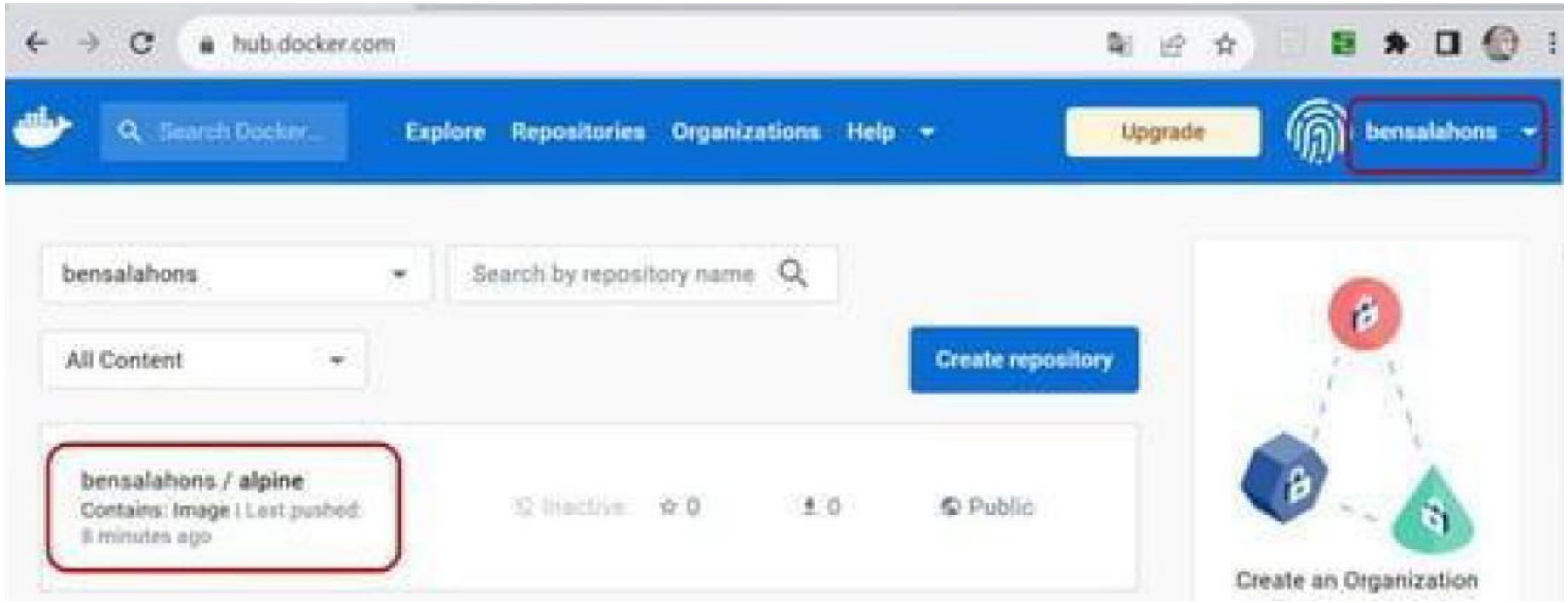
- La commande **docker rm** permet de supprimer un ou plusieurs conteneurs Docker de votre système. Elle est utilisée pour nettoyer votre environnement Docker en supprimant **les conteneurs arrêtés** et leurs ressources.

- **Options courantes :**
- **docker rm <container_id>**: Supprime le conteneur identifié par son ID.
- **docker rm <container_name>:** Supprime le conteneur nommé.
- **docker rm -f <container_id>:** Force la suppression d'un conteneur en cours d'exécution (équivalent à docker stop suivi de docker rm).
- **docker rm $(docker ps -aq):** Supprime tous les conteneurs arrêtés.

Déposer une image sur DockerHub

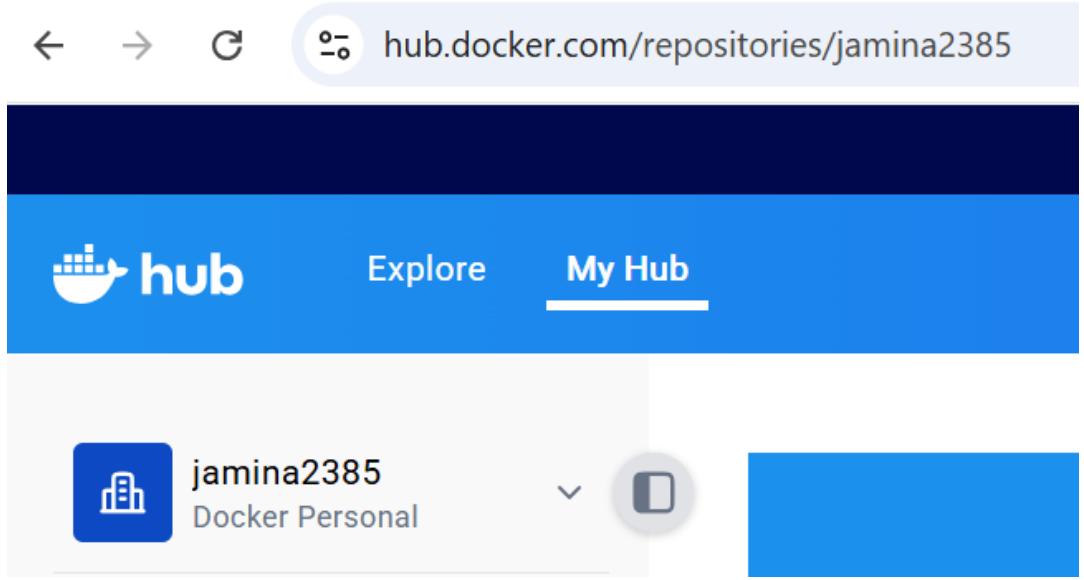
- Pour soumettre une image que vous avez créé dans DockerHub, utilisez la commande **push** :

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker push bensalahons/alpine:1.0.0
The push refers to repository [docker.io/bensalahons/alpine]
764e3bdab165: Pushed
4693057ce236: Mounted from library/alpine
1.0.0: digest: sha256:7fad49bab9b99cb0bff6012d0925b91f3d35e5a28ad597e52fb7ac1624d502f8 size: 741
```



Se connecter à Dockerhub

- Créez un compte Docker hub



- Se connecter sous WSL :

```
jamina_dev@DESKTOP-SKOOCDK:~$ docker login -u jamina2385

i Info → A Personal Access Token (PAT) can be used instead.
         To create a PAT, visit https://app.docker.com/settings


Password:
Login Succeeded
jamina_dev@DESKTOP-SKOOCDK:~$
```

5. Orchestration multi-conteneurs

Docker-compose c’est quoi ?

- **Docker Compose** est un outil qui permet de définir et d'exécuter des applications composées de plusieurs conteneurs Docker. Il utilise un fichier **YAML** pour configurer et orchestrer ces conteneurs, simplifiant ainsi la gestion et le déploiement d'applications complexes.

- **Docker Compose** automatise les étapes de création, de configuration et de mise en réseau des conteneurs, ce qui accélère considérablement le processus de développement et de déploiement.

- Toutes les définitions et configurations de l'application sont stockées dans un fichier unique, **docker-compose.yml**, ce qui rend la gestion de l'application plus simple et plus lisible.

Exemple de docker-compose.yml

- Le fichier « docker-compose.yml » se trouve dans le dossier contenant le projet en question. Dans notre cas c'est **eclipse-workspace**.

- Attention à l'indentation. Le fichier doit être indenté

- Deux conteneurs se lancent sur les ports 8080 et 8081

```
services:
 my-compte-service1:
  build: ./compte-service
  container_name: my-compte-service1
  ports:
   - '8080:8080'
  expose:
   - '8080'
 my-compte-service2:
  build: ./compte-service
  container_name: my-compte-service2
  ports:
   - '8081:8080'
  expose:
   - '8080'
```

Gestion des conteneurs dans docker-compose.yml

- Démarrer l'ensemble des conteneurs

```
docker compose up
```

- Lister tous les conteneurs en cours d'exécution

```
docker ps
```

- Arrêter tous les conteneurs

```
docker compose down
```

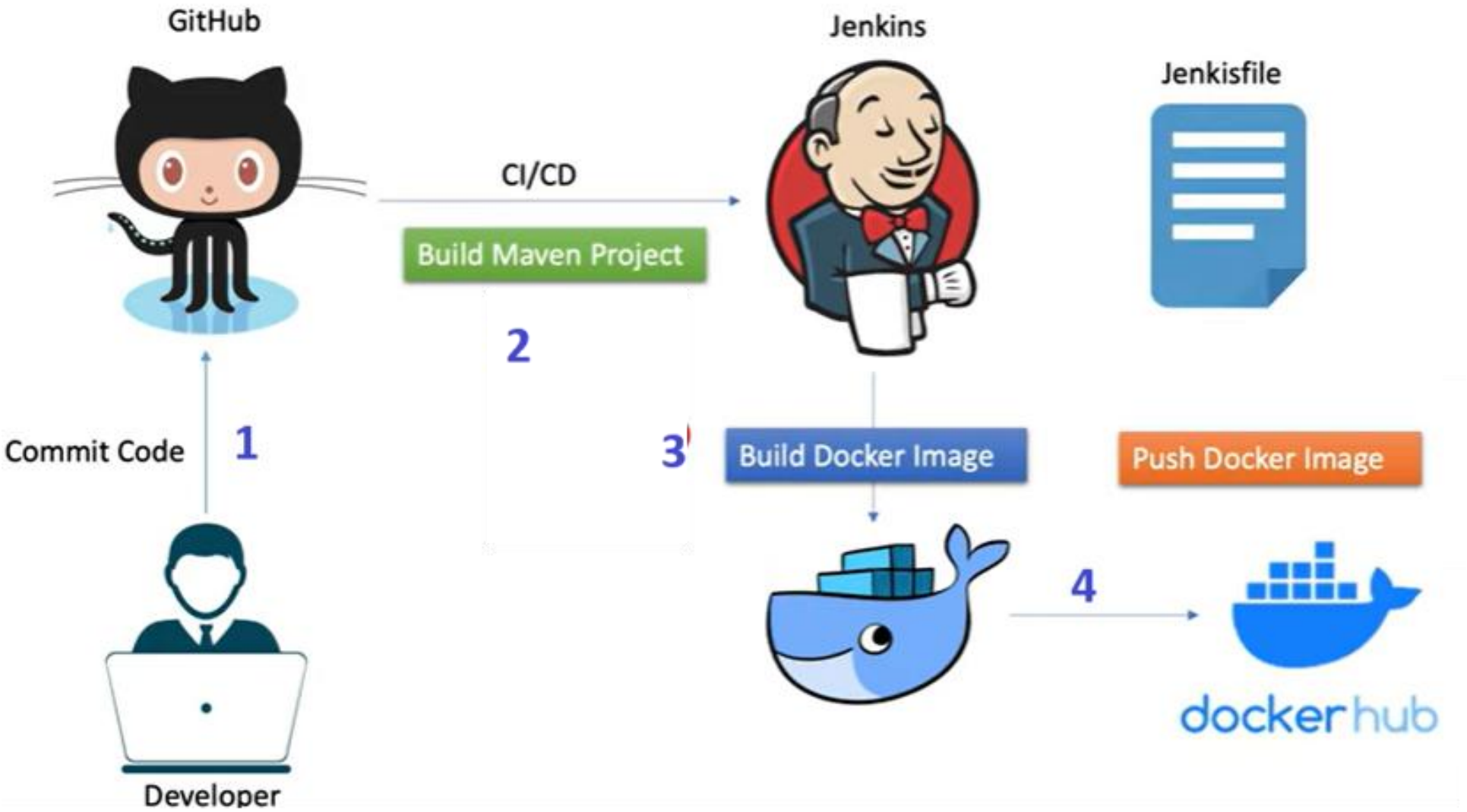
- Démarrez tous les conteneurs dans le fichier docker-compose.yml en arrière-plan

```
docker compose up -d
```

```
C:\Users\jarra\eclipse-workspace\compte-service>docker ps
CONTAINER ID   IMAGE                                    COMMAND                  CREATED         STATUS
       PORTS                                NAMES
ee5941203a4b   eclipse-workspace-my-compte-service2   "java -jar /app.jar"   4 minutes ago   Up 2 min
utes             0.0.0.0:8081->8080/tcp   my-compte-service2
7247f7e6995c   eclipse-workspace-my-compte-service1   "java -jar /app.jar"   4 minutes ago   Up 2 min
utes   0.0.0.0:8080->8080/tcp             my-compte-service1
```

6. Création d’un pipeline CI/CD avec Docker

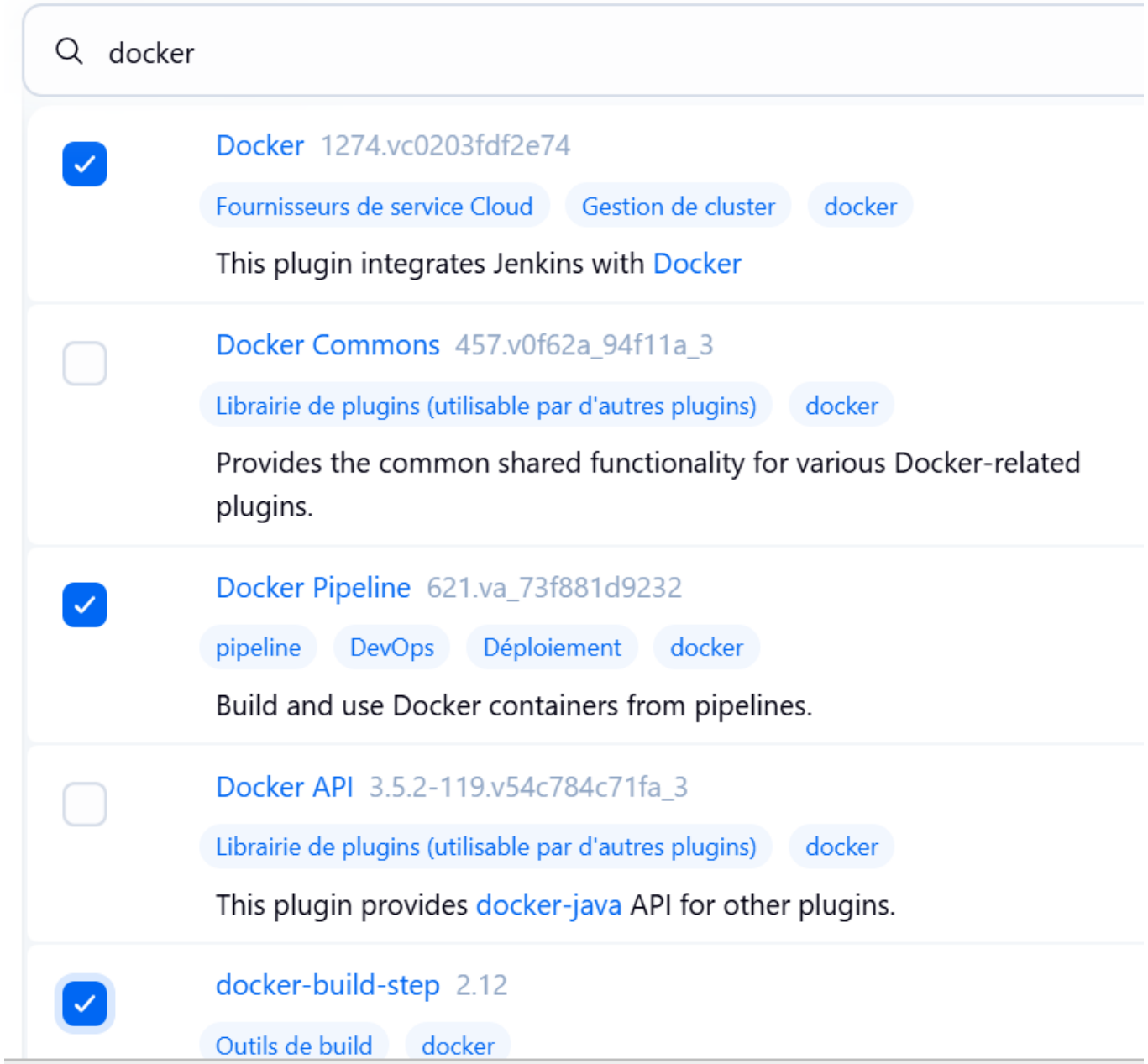
Le pipeline CI/CD à mettre en place



Plugins à installer sous Jenkins

- Pour intégrer Docker dans Jenkins, installez les plugins suivants :

- Docker
- Docker pipeline
- Docker-build-step

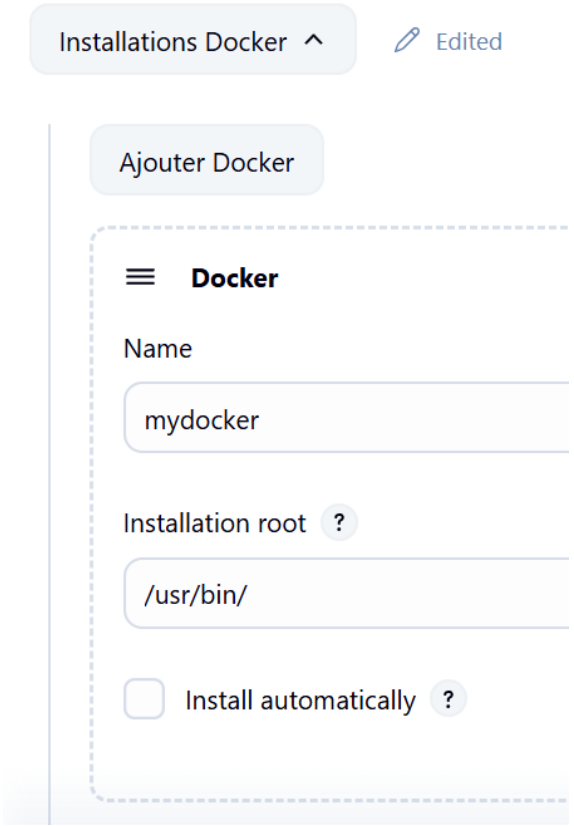


Configuration de docker dans Jenkins

- Sous WSL, tapez les commandes suivantes :

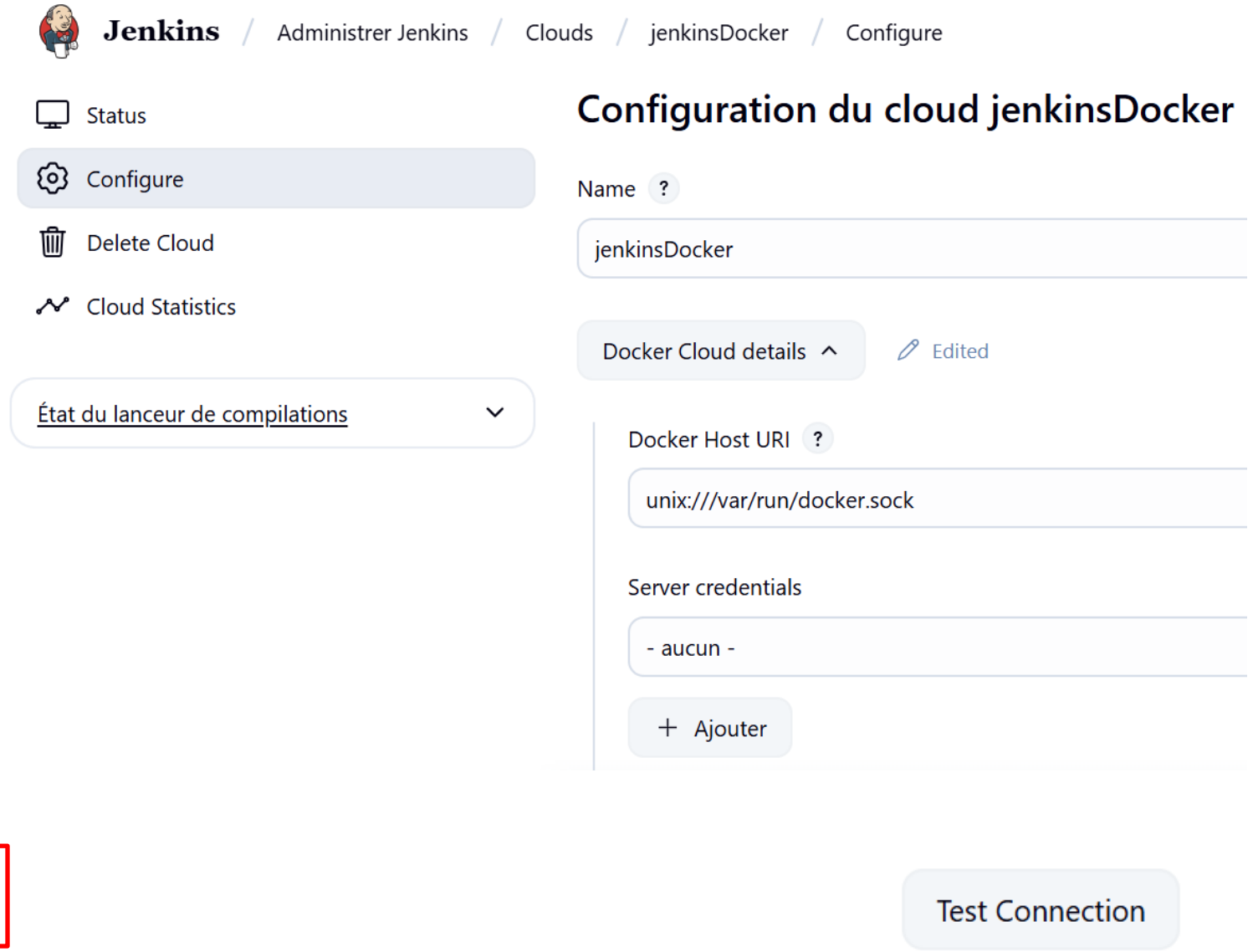
```
jamina_dev@DESKTOP-SKOOCDK:~$ sudo su -
[sudo] password for jamina_dev:
root@DESKTOP-SKOOCDK:~# chmod 777 /var/run/docker.sock
root@DESKTOP-SKOOCDK:~# usermod -a -G docker jenkins
root@DESKTOP-SKOOCDK:~# systemctl restart jenkins
```

- Sous Jenkins, allez Configurer – Tools :



Création d’un nouveau cloud docker sous Jenkins

- Allez Administrer Jenkins-> Clouds -> ajouter cloud
- Ici j’ai nommé le cloud « jenkinsDocker »
- Configurer le cloud comme dans la figure puis tester la connexion.

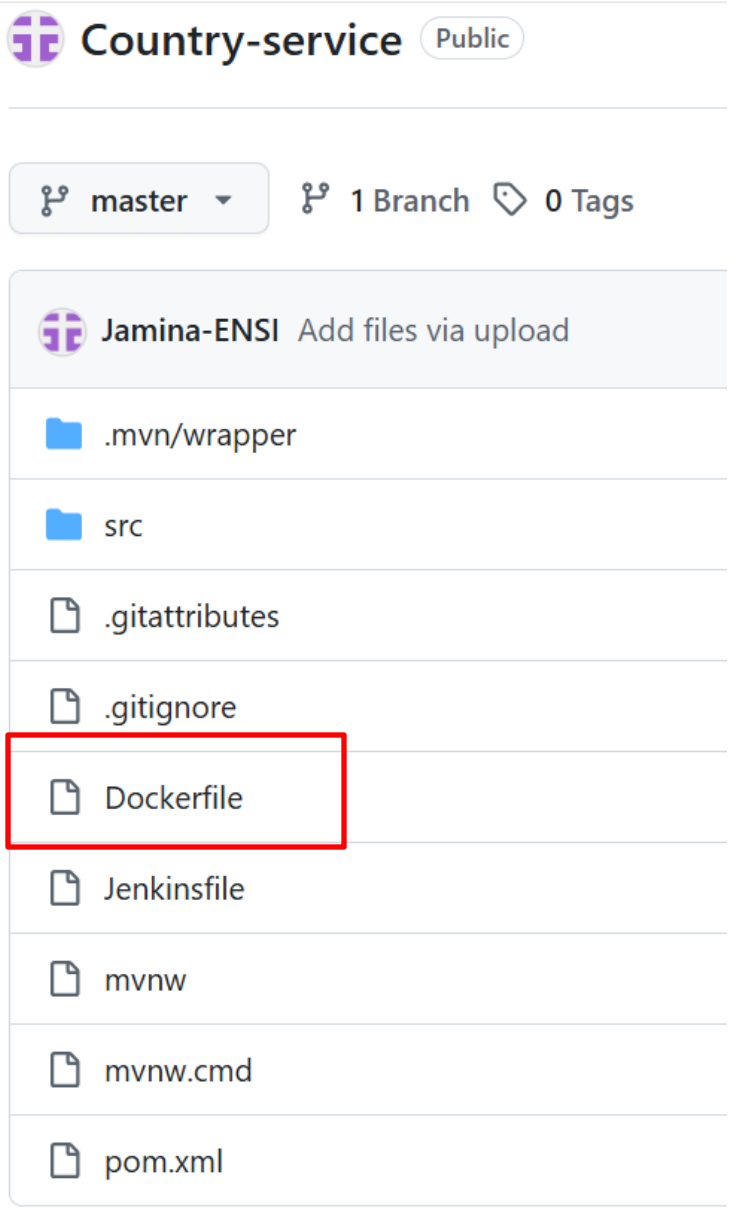


Création du pipeline CI/CD avec Dockerfile

- Faites le push du Dokerfile dans le Github « Country-service »

Dockerfile

```
1 FROM openjdk:21-oracle
2 VOLUME /tmp
3 COPY target/*.jar app.jar
4 ENTRYPOINT ["java","-jar","/app.jar"]
```



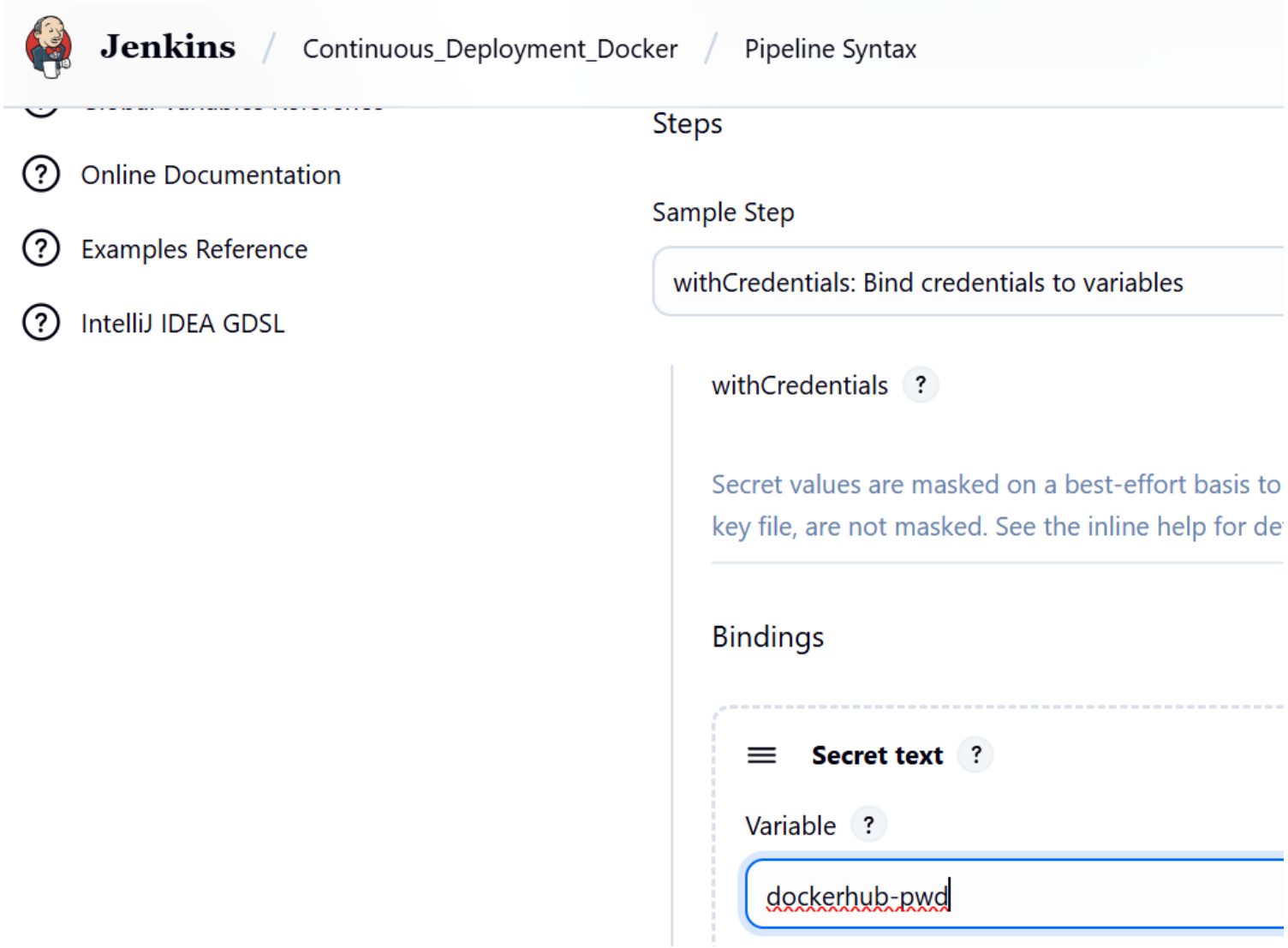
Création du pipeline CI/CD avec Dockerfile

```
pipeline {
        agent any
        tools {
            maven 'mymaven'
        }
        stages {
            stage('Checkout code')
            {
                steps {
                    git branch: 'master', url: 'https://github.com/Jamina-ENSI/Country-service
                }
            }
            stage('Build maven')
            {
                steps {
                    sh 'mvn clean install'
                }
            }
```

Création du pipeline CI/CD avec Dockerfile

```
19  stage('Build Dockerfile ')
20  {
21      steps {
22          sh 'docker build . -t my-country-service:$BUILD_NUMBER '
23          withCredentials([string(credentialsId: 'dockerhub-pwd', variable: 'dockerhubpwd')]) {
24          sh 'docker login -u jamina2385 -p ${dockerhubpwd}'
25          }
26          sh 'docker tag my-country-service:$BUILD_NUMBER jamina2385/my-country-service:$BUILD_NU
27          sh 'docker push jamina2385/my-country-service:$BUILD_NUMBER'
28      }
29  }
30              stage('Deploy micro-service')
31              {
32                  steps {
33                      sh 'docker rm -f $(docker ps -aq)'
34                      sh 'docker run -d -p 8082:8082 my-country-service:$BUILD_NUMBER'
35                  }
36              }
37          }
38      }
```

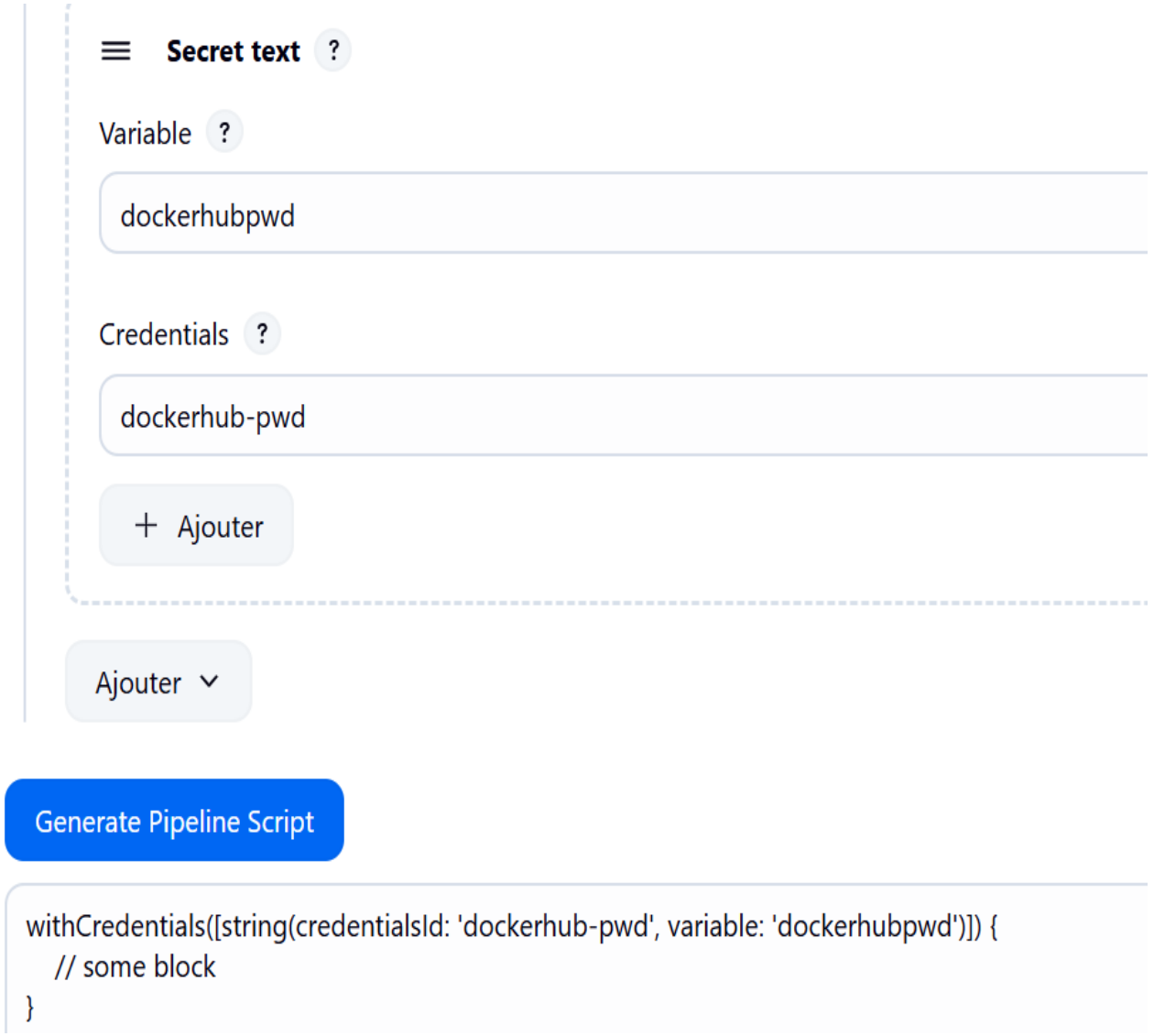
Configuration du credentials



Configuration du credentials

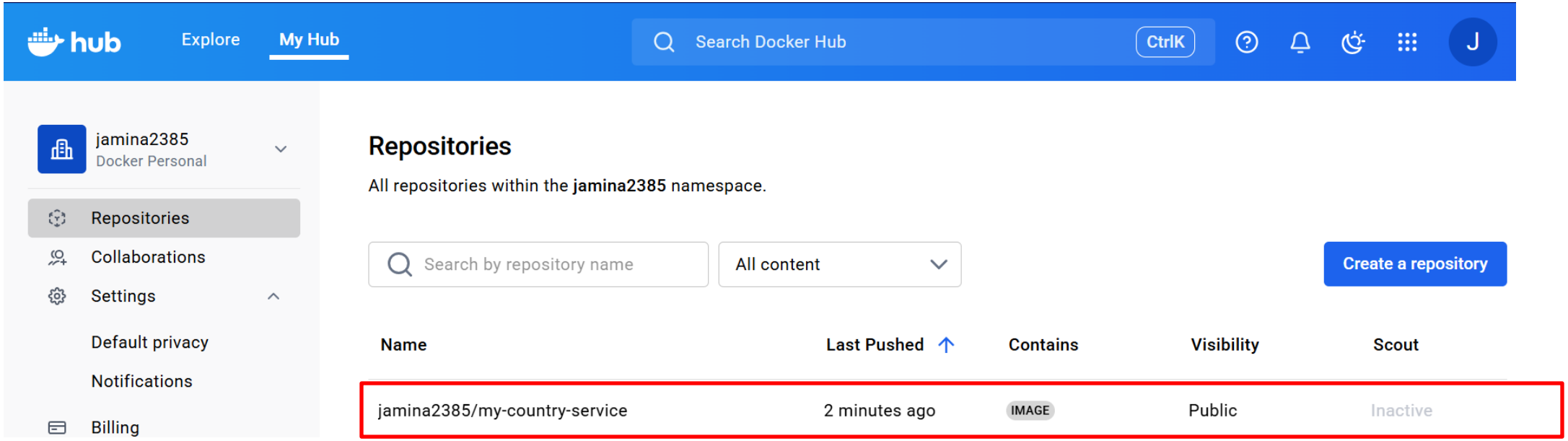
Configuration du credentials

- Cliquez sur Generate Pipeline Script et le rajouter dans votre script du pipeline



Création du pipeline CI/CD avec Dockerfile

- Le pipeline s'est bien exécuté et l'image est bien déposée dans Dockerhub.



- *"Apprendre par le projet, c'est découvrir par l'action, créer par la compréhension, et réussir par la persévérance."*

amina.jarraya@ensi-uma.tn

ENSI Manouba