

docker

Conteneurisation avec Docker

Projet de Téléinformatique
Superviseur : Roch-Neirey Martin
Filière : Année Passerelle ingénierie, HEIA-FR

Sevinc Rabia
rabia.sevinc@hes-so.ch

Contents

I.	INTRODUCTION	2
II.	Phase d'analyse	3
A.	Qu'est-ce que Docker ?	3
B.	Pourquoi utiliser Docker ?	3
C.	Que sont les conteneurs ?	4
1.	Comprendre le fonctionnement de Docker	4
2.	Qu'est-ce que cela veut dire, partager le noyau ?	5
3.	Fonctionnement de Docker par rapport aux hyperviseurs	6
4.	Comparaison entre conteneurs et machines virtuelles	6
D.	Images et Conteneurs Docker	8
E.	Qu'est-ce que docker compose ?	9
1.	Fonctionnement de Docker Compose	9
2.	Commandes	9
F.	Docker Swarm	10
1.	Comparaison avec Kubernetes	11
III.	Phase d'implémentation	12
A.	Installation de Docker sur Ubuntu	12
B.	Commandes Docker	13
C.	Docker Run pour lancer et gérer des conteneurs	18
D.	Création et publication d'images Docker sur Docker Hub	21
E.	Déploiement d'une application Flask basée sur Python	23
F.	Docker Compose	26
1.	Utilisation de Docker Compose	32
IV.	CONCLUSION	35
V.	SOURCES	38
A.	Images	39

I. INTRODUCTION

Ce rapport présente une exploration détaillée de Docker, une technologie de conteneurisation open source largement utilisée pour simplifier le développement, le déploiement et la gestion des applications. Il examine les concepts clés tels que les conteneurs, les images Docker, Docker Compose et Docker Swarm, tout en offrant des comparaisons avec des technologies similaires comme les machines virtuelles. À travers une phase d'analyse et une phase d'implémentation, le rapport met en lumière les avantages et les cas d'utilisation de Docker, en illustrant son fonctionnement par des exemples pratiques et des tests réels.

II. Phase d'analyse

A. Qu'est-ce que Docker ?

Docker est une plateforme open source qui regroupe une application et toutes ses dépendances sous forme de conteneurs Docker, garantissant que l'application fonctionnera de manière transparente dans n'importe quel environnement. L'utilisation des méthodologies de Docker pour l'expédition, les tests et le déploiement du code permet de réduire le délai entre l'écriture du code et sa mise en production. Dans Docker, chaque application est indépendante des autres, offrant ainsi l'assurance que les applications peuvent être développées sans risque d'interférer les unes avec les autres.

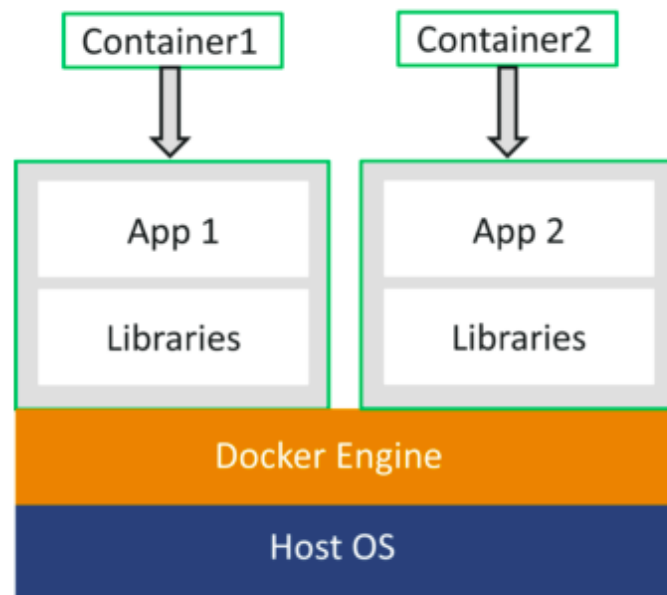


Figure 1 : Docker

B. Pourquoi utiliser Docker ?

La containerisation est une technologie de plus en plus populaire dans le monde de l'informatique, et Docker en est un acteur majeur. Cette technologie répond au problème fréquent des équipes de développement : "Ça fonctionne sur ma machine...".

Prenons un exemple : dans une équipe de quatre développeurs travaillant sur un projet, chacun utilise un environnement différent (Windows, Linux, macOS). Ces variations nécessitent l'installation de bibliothèques et de fichiers spécifiques à chaque système, ce qui peut entraîner des conflits et ralentir le cycle de développement. Docker résout ce problème en fournissant une plateforme de containerisation qui permet de créer, déployer et exécuter des applications dans des conteneurs.

Un conteneur regroupe une application avec toutes ses dépendances nécessaires, garantissant ainsi son bon fonctionnement sur n'importe quel système. Lancé en 2013 par Docker, Inc., Docker propose également d'autres composants clés comme les Docker Images, les Docker

Files et les Docker Registries, qui simplifient encore d'avantage le travail des développeurs. Grâce à Docker, il est possible de développer des applications sans se soucier des contraintes liées à l'environnement.

C. Que sont les conteneurs ?

Les conteneurs isolent les logiciels de leur environnement et garantissent qu'ils fonctionnent de manière uniforme malgré les différences, par exemple entre les environnements de développement et de préproduction.

Pour garantir cette isolation, Docker utilise des technologies comme **cgroups** (control groups). Les cgroups permettent de limiter et de surveiller les ressources système (CPU, mémoire, etc.) utilisées par chaque conteneur. Ainsi, chaque conteneur peut fonctionner indépendamment, sans affecter les performances des autres conteneurs ou du système hôte.

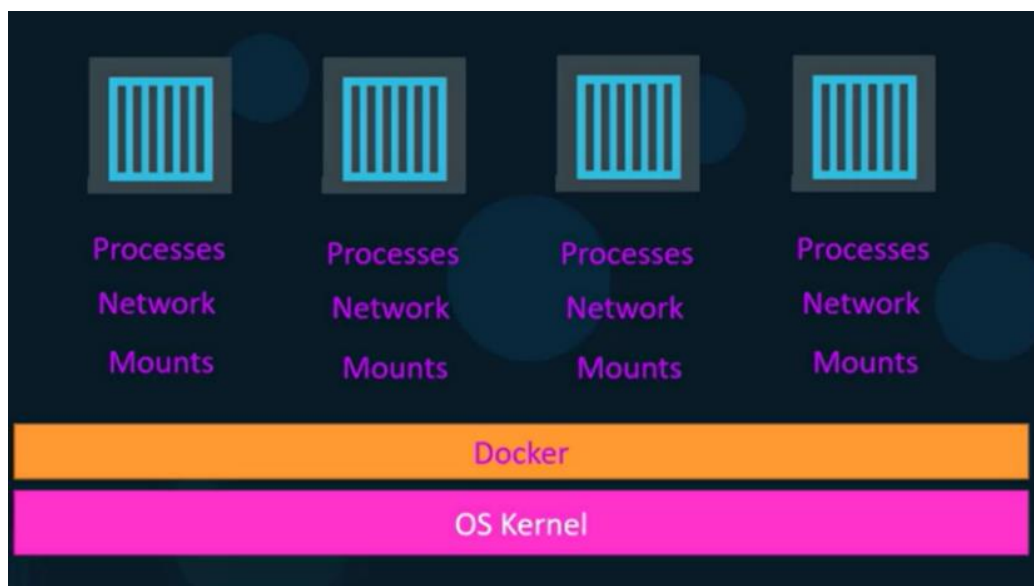


Figure 2 : Conteneurs

1. Comprendre le fonctionnement de Docker

Pour comprendre le fonctionnement de Docker, il est utile d'examiner les systèmes d'exploitation tels qu'Ubuntu, Fedora, SUSE ou CentOS. Tous ces systèmes d'exploitation sont composés de deux éléments : un noyau de système d'exploitation (OS kernel) et un ensemble de logiciels.

Le noyau de l'OS est responsable de l'interaction avec le matériel sous-jacent. Bien que le noyau de l'OS reste identique (Linux, dans ce cas), ce sont les logiciels qui se trouvent au-dessus de ce noyau qui différencient ces systèmes d'exploitation. Ces logiciels peuvent inclure une interface utilisateur, des pilotes, des compilateurs, des gestionnaires de fichiers, des outils pour développeurs, etc.

Ainsi, un noyau Linux commun est partagé entre tous ces systèmes, tandis que les logiciels spécifiques apportent des distinctions entre eux.



Figure 3 : Fonctionnement de Docker

2. Qu'est-ce que cela veut dire, partager le noyau ?

Partager un noyau signifie qu'un système d'exploitation, comme Ubuntu, qui est le système hôte, peut exécuter plusieurs conteneurs Docker représentant différents systèmes d'exploitation, à condition qu'ils reposent sur le même noyau, ici Linux. Par exemple, si le système d'exploitation principal est Ubuntu, Docker peut exécuter des conteneurs basés sur des distributions telles que Debian, Fedora, SUSE ou CentOS. Cela est possible car Docker ne virtualise pas un système entier mais utilise le noyau du système hôte pour interagir directement avec le matériel.

Cependant, il est important de noter que Docker ne peut pas exécuter directement des systèmes d'exploitation qui ne reposent pas sur le noyau Linux (comme Windows) en mode natif, sauf si des solutions comme WSL2 ou une machine virtuelle sont utilisées.



Figure 4 : Partager le noyau

3. Fonctionnement de Docker par rapport aux hyperviseurs

Contrairement aux hyperviseurs, qui virtualisent des systèmes d'exploitation entiers avec leur propre noyau et ressources matérielles, Docker se concentre sur la containerisation des applications. Cela signifie qu'il n'exécute pas différents noyaux ou systèmes d'exploitation sur le même matériel.

Son principal objectif est de regrouper une application et ses dépendances dans un conteneur léger, partageant le noyau du système hôte. Cela rend Docker plus rapide et plus efficace que les hyperviseurs pour le déploiement d'applications, tout en consommant moins de ressources. Ainsi, les conteneurs peuvent être créés, déployés et exécutés partout, à tout moment, et autant de fois que nécessaire.

4. Comparaison entre conteneurs et machines virtuelles

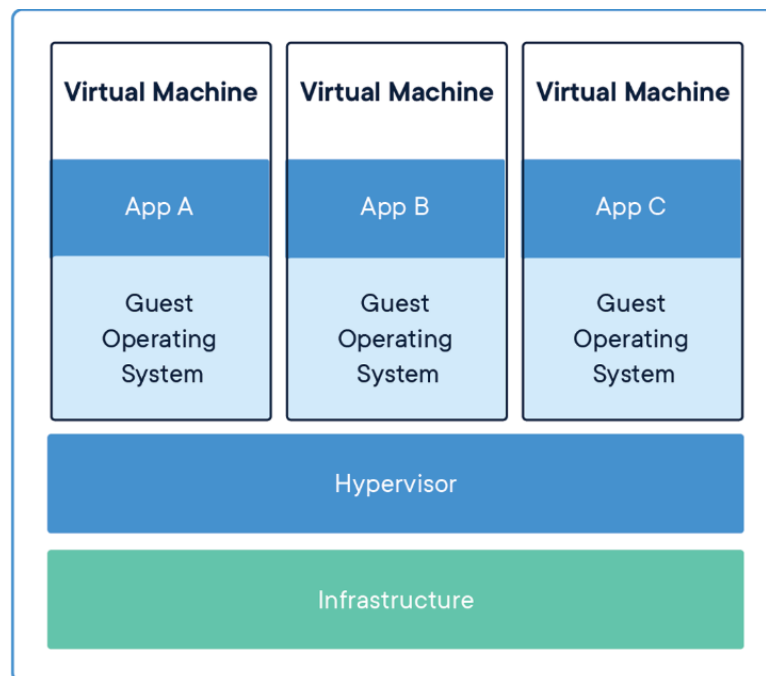


Figure 5 : Machines virtuelles

Machines virtuelles (VM)

1. Les machines virtuelles (VM) représentent une abstraction du matériel physique, permettant de transformer un serveur en plusieurs serveurs virtuels.
2. L'hyperviseur permet à plusieurs machines virtuelles de fonctionner sur une seule machine physique.
3. Chaque machine virtuelle contient une copie complète d'un système d'exploitation, de l'application, des binaires nécessaires et des bibliothèques, ce qui occupe généralement plusieurs dizaines de gigaoctets.
4. Le démarrage des machines virtuelles peut également être lent.

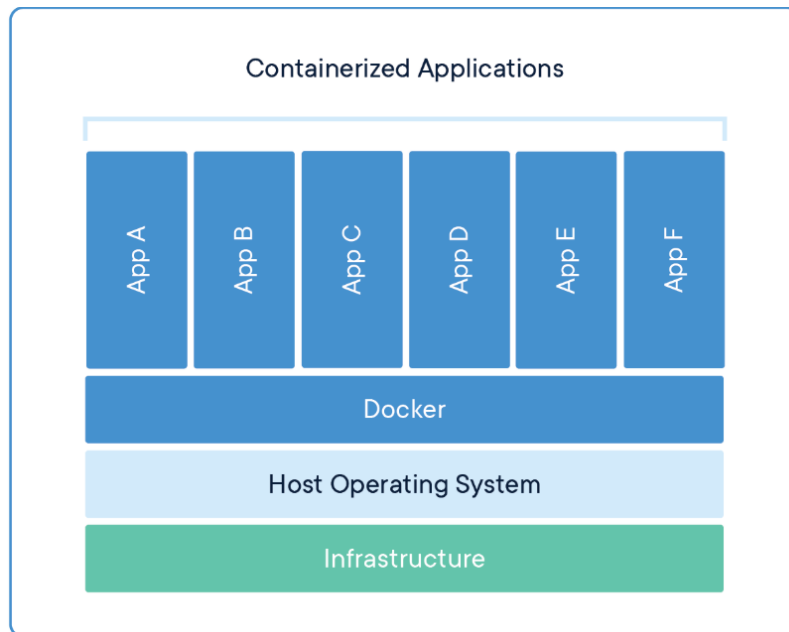


Figure 6 : Conteneurs

Caractéristiques des conteneurs

1. Les conteneurs représentent une abstraction au niveau de la couche applicative, regroupant le code et ses dépendances.
2. Plusieurs conteneurs peuvent s'exécuter sur une même machine et partagent le noyau du système d'exploitation avec d'autres conteneurs, tout en fonctionnant comme des processus isolés dans l'espace utilisateur.
3. Les conteneurs occupent moins d'espace que les machines virtuelles (les images de conteneurs mesurent généralement quelques dizaines de mégaoctets).
4. Ils permettent de gérer un plus grand nombre d'applications tout en nécessitant moins de machines virtuelles et de systèmes d'exploitation.

Les conteneurs et les machines virtuelles utilisés conjointement offrent une grande flexibilité dans le déploiement et la gestion des applications.

D. Images et Conteneurs Docker

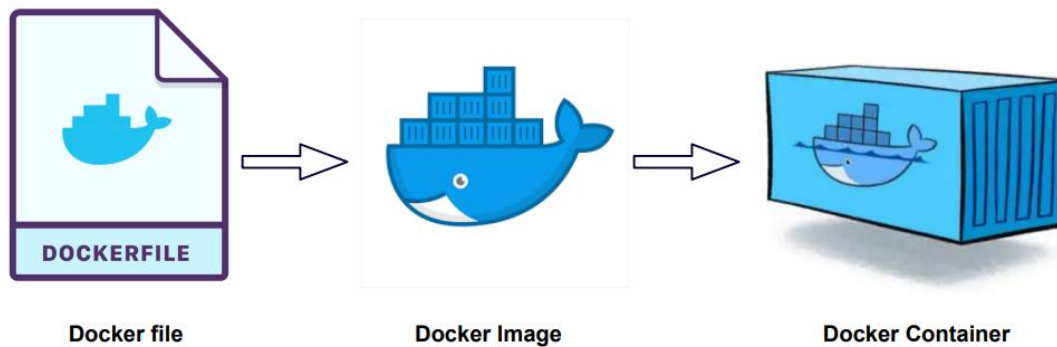


Figure 7 : Image Docker

Une image Docker est un modèle ou un package, similaire à un modèle de machine virtuelle utilisé dans le domaine de la virtualisation. Elle sert à créer un ou plusieurs conteneurs. L'image représente un instantané d'un environnement, incluant toutes les configurations nécessaires. Elle est en lecture seule et agit comme une sorte de schéma ou de plan pour les conteneurs. Si une image adaptée n'est pas disponible, il est possible d'en créer une et de la publier sur le Docker Hub pour la rendre accessible au public.

Un conteneur, quant à lui, est une instance en cours d'exécution d'une image. Il est isolé, dispose de son propre environnement et exécute un ensemble de processus spécifiques. Les conteneurs peuvent être comparés à des conteneurs de transport pour les logiciels : ils contiennent les fichiers et programmes essentiels, permettant de livrer efficacement une application du producteur au consommateur.

Pour créer une image Docker, il est nécessaire de rédiger un fichier nommé **Dockerfile** contenant les instructions nécessaires pour configurer l'application, comme l'installation des dépendances, la copie du code source et le point d'entrée de l'application. Une fois le Dockerfile prêt, l'image peut être générée localement à l'aide de la commande `docker build`, en spécifiant le fichier comme entrée et en ajoutant un tag pour identifier l'image. Enfin, pour rendre cette image accessible sur le registre public Docker Hub, la commande `docker push` permet de la publier en spécifiant son nom.

Docker utilise les images pour générer des conteneurs, lesquels sont ensuite utilisés pour exécuter les applications.

E. Qu'est-ce que docker compose ?

Docker Compose est un outil utilisé pour définir, gérer et exécuter facilement plusieurs conteneurs Docker. Il est particulièrement adapté aux applications nécessitant plusieurs conteneurs. Par exemple, si une application nécessite un serveur web, une base de données et un service de mise en cache, Docker Compose permet de regrouper ces services.

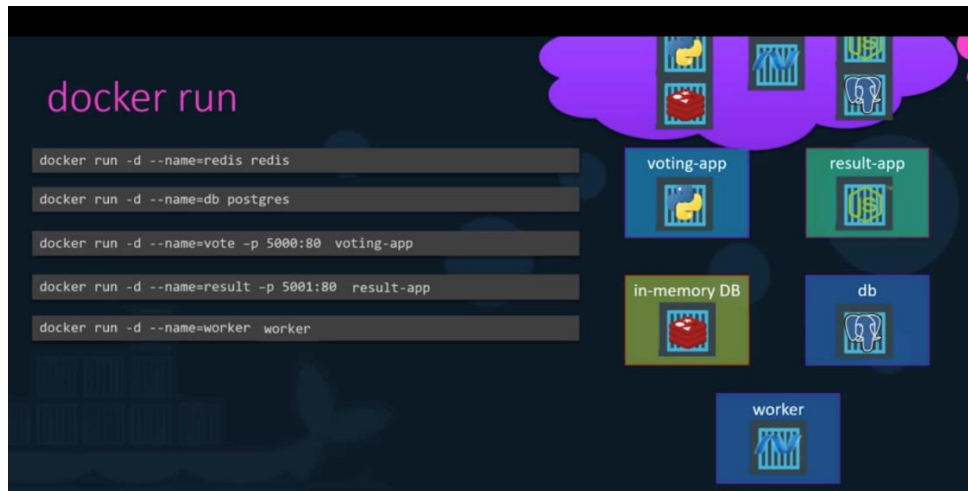


Figure 8: Plusieurs conteneurs

1. Fonctionnement de Docker Compose

Docker Compose utilise un fichier de configuration au format YAML, appelé *compose.yml*, pour définir les services d'une application. Ce fichier suit les spécifications de Compose, qui définissent les applications multi-conteneurs. Une fois configurés, tous les services peuvent être créés et démarrés via l'interface en ligne de commande de Docker Compose.

```
docker-compose.yml
services:
  web:
    image: "mmumshad/simple-webapp"
  database:
    image: "mongodb"
  messaging:
    image: "redis:alpine"
  orchestration:
    image: "ansible"
```

Figure 9: Fichier *compose.yml*

2. Commandes

Le fichier *compose.yml* est généralement placé dans le répertoire de travail. Il peut être complété par d'autres fichiers Compose pour faciliter la modularité et la maintenance. Ces fichiers peuvent être fusionnés en respectant un ordre de priorité.

Docker Compose via la commande `docker compose` et ses sous-commandes. Les principales commandes incluent :

docker compose up : Démarrer tous les services définis.

docker compose down : Arrêter et supprimer les services en cours d'exécution.

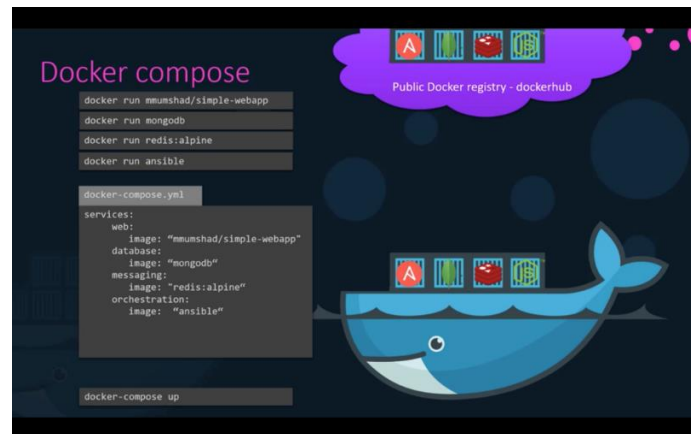


Figure 10: Docker compose

F. Docker Swarm

Dans un environnement où les applications doivent être disponibles en permanence, gérer plusieurs conteneurs peut devenir complexe. Pour lancer plusieurs instances d'une application, il faut utiliser la commande `docker run` plusieurs fois manuellement. Cela demande de surveiller la charge et les performances de l'application, ainsi que la santé des conteneurs. Si un conteneur ou son hôte tombe en panne, les conteneurs ne seront plus accessibles, et il faudra les redéployer manuellement.

Une solution d'orchestration de conteneurs permet de résoudre ces problèmes. Elle utilise des outils et des scripts pour gérer les conteneurs en production. Avec plusieurs hôtes Docker, les applications restent disponibles même si un hôte tombe en panne. De plus, elle permet de déployer rapidement des centaines ou des milliers d'instances avec une seule commande, comme avec Docker Swarm.

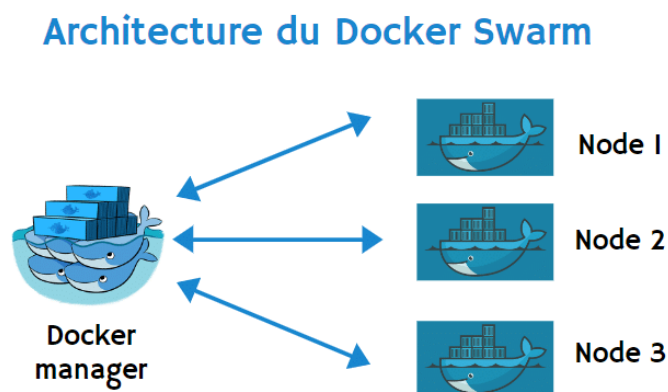


Figure 11: Architecture du Docker Swarm

Docker Swarm permet de regrouper plusieurs machines Docker en un seul cluster. Pour configurer un cluster Swarm, une machine est désignée comme gestionnaire (manager) et les autres comme travailleurs (workers). La commande **docker swarm init** est utilisée sur le gestionnaire pour initialiser le cluster, et les commandes générées permettent aux machines workers de rejoindre le cluster. Par exemple, pour exécuter trois instances d'un serveur web, on utilise la commande `docker service create` avec l'option **--replicas=3**. Les instances seront réparties automatiquement sur les différents nœuds.

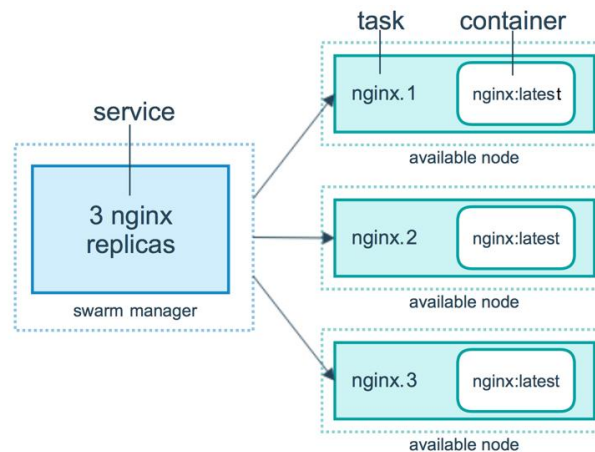


Figure 12: Commande replicas

Docker Swarm apporte ainsi une solution efficace pour simplifier la gestion des conteneurs à grande échelle, tout en assurant une haute disponibilité et une meilleure utilisation des ressources.

1. Comparaison avec Kubernetes

- Bien que Docker Swarm simplifie la gestion des conteneurs, Kubernetes offre des fonctionnalités plus avancées pour les projets à grande échelle.
- Docker Swarm est plus facile à configurer et à utiliser, ce qui le rend idéal pour les petits projets ou les équipes débutantes. En revanche, Kubernetes est plus complexe mais convient aux grandes infrastructures.

Docker Swarm est une solution simple et efficace pour gérer les conteneurs, mais Kubernetes s'impose comme un outil robuste pour les environnements complexes et les grandes organisations.

III. Phase d'implémentation

A. Installation de Docker sur Ubuntu

Vérification des prérequis

Avant de commencer, assurez-vous que les conditions sur <https://docs.docker.com/desktop/setup/install/linux/ubuntu/> sont remplies.

La commande `cat /etc/*release*` est utilisée pour afficher les informations sur la distribution Linux installée sur un système. Elle permet de consulter des détails tels que:

- Le nom de la distribution (ex. Ubuntu, Debian, etc.).
- La version de la distribution (ex. 24.04.1 LTS).
- Le nom de code associé à cette version (ex. "noble").

```
student@docker01:~$ cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=24.04
DISTRIB_CODENAME=noble
DISTRIB_DESCRIPTION="Ubuntu 24.04.1 LTS"
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

Figure 13: Commande `cat /etc/*release*`

Suppression d'ancien version de Docker

Si Docker était déjà installé, il fallait le supprimer avec la commande suivante :

```
student@docker01:~$ sudo apt-get remove docker docker-engine docker.io containerd runc
[sudo] password for student:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package 'docker' is not installed, so not removed
E: Unable to locate package docker-engine
```

Figure 14: Suppression d'ancien version de Docker

Dans ce cas, Docker n'était pas installé, donc aucun paquet n'a été supprimé.

Téléchargement et exécution du script d'installation

Docker peut être téléchargé et installé en suivant les instructions disponibles dans [la documentation officielle](#). Les commandes suivantes permettent d'automatiser ce processus :

```
student@docker01:~$ curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
# Executing docker install script, commit: 4c94a56999e10efcf48c5b8e3f6afea464f9108e
```

Figure 15: Commande d'installation

Cette procédure installe automatiquement tous les composants nécessaires à Docker.

Vérification de l'installation

Pour vérifier que l'installation a réussi, la commande suivante est utilisée :

```
student@docker01:~$ sudo docker version
Client: Docker Engine - Community
Version: 27.4.1
API version: 1.47
Go version: go1.22.10
Git commit: b9d17ea
Built: Tue Dec 17 15:45:46 2024
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
Version: 27.4.1
API version: 1.47 (minimum version 1.24)
Go version: go1.22.10
Git commit: c710b88
Built: Tue Dec 17 15:45:46 2024
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.7.24
GitCommit: 88bf19b2105c8b17560993bee28a01ddc2f97182
runc:
Version: 1.2.2
GitCommit: v1.2.2-0-g7cb3632
docker-init:
Version: 0.19.0
GitCommit: de40ad0
student@docker01:~$
```

Figure 16: Commande version

Cette commande affiche les détails sur la version installée de Docker Engine ainsi que sur ses composants associés.

B. Commandes Docker

Docker run

Pour savoir quelles images peuvent être utilisées, il est conseillé de consulter [le site web Docker Hub](https://docs.docker.com/search/). De nombreuses images officielles sont disponibles par défaut. Dans ce cas précis, l'image CentOS est utilisée. En cliquant dessus, des instructions pour démarrer CentOS sont affichées. Par exemple, il est indiqué `docker pull centos`. Ainsi, une simple commande `docker run centos` peut être exécutée. Cette commande vérifie d'abord si une image CentOS est disponible localement. Si ce n'est pas le cas, la dernière version de l'image CentOS sera automatiquement téléchargée.



Figure 17: Image centOS

```
student@docker01:~$ sudo docker run centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
ald0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
student@docker01:~$
```

Figure 18: Commande 'docker run centos'

Une image Docker CentOS a été téléchargée. Sans commande spécifique, le conteneur démarre puis s'arrête, car aucune action n'a été demandée. L'image CentOS est une base du système d'exploitation CentOS.

Pour maintenir un conteneur actif, une commande doit être exécutée, comme bash. Les options -it permettent de se connecter directement au terminal du conteneur. L'invite de commandes affiche alors root@ suivi d'un identifiant unique, représentant le conteneur en cours d'exécution.

```
student@docker01:~$ sudo docker run -it centos bash
[root@4660b10072be /]#
[root@4660b10072be /]#
[root@4660b10072be /]#
```

Figure 19: Commande 'docker run -it centos bash'

```
[root@4660b10072be /]# cat /etc/*release*
CentOS Linux release 8.4.2105
Derived from Red Hat Enterprise Linux 8.4
NAME="CentOS Linux"
VERSION="8"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="8"
PLATFORM_ID="platform:el8"
PRETTY_NAME="CentOS Linux 8"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:8"
HOME_URL="https://centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-8"
CENTOS_MANTISBT_PROJECT_VERSION="8"
CentOS Linux release 8.4.2105
CentOS Linux release 8.4.2105
cpe:/o:centos:centos:8
[root@4660b10072be /]#
```

Pour vérifier le système d'exploitation utilisé, la commande `cat /etc/*release*` peut être exécutée. Cette commande affiche les informations sur le système, confirmant ici qu'il s'agit de CentOS Linux. Cela indique que l'on se trouve à l'intérieur du conteneur CentOS.

Figure 20: Commande 'cat /etc/*release*' for conteneur

Docker ps

La commande docker ps affiche uniquement les conteneurs en cours d'exécution. Si aucun conteneur n'est actif, rien n'est listé.

```
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
student@docker01:~$
```

Figure 21: Commande 'docker ps'

Lorsqu'un conteneur CentOS est exécuté avec une commande comme sleep 20, il reste actif pendant 20 secondes avant de s'arrêter. Pour exécuter le conteneur en arrière-plan, l'option -d peut être utilisée. Cela permet de récupérer l'ID du conteneur et de retourner au terminal. Avec docker ps, il est alors possible de voir l'ID, l'image utilisée et la commande en cours d'exécution.

```
student@docker01:~$ sudo docker run -d centos sleep 20
a10f6f0c39a433dde710908605e5f6c786a505cd5e5f0b2de923853161cb3d67
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
a10f6f0c39a4   centos    "sleep 20"  6 seconds ago  Up 5 seconds           epic_khayyam
student@docker01:~$
```

Figure 22: Commande sleep

Après 20 secondes, il est également vérifié.

```
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
a10f6f0c39a4   centos    "sleep 20"  6 seconds ago  Up 5 seconds           epic_khayyam
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
student@docker01:~$
```

Figure 23: Vérification de la commande sleep

Docker ps -a

Pour afficher tous les conteneurs précédemment exécutés, y compris ceux qui sont arrêtés, la commande docker ps -a peut être utilisée. Cette option liste tous les conteneurs, y compris ceux qui ont terminé leur exécution.

```
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
a10f6f0c39a4   centos    "sleep 20"  2 minutes ago  Exited (0) 2 minutes ago           epic_khayyam
4660b10072be   centos    "bash"      8 minutes ago  Exited (0) 6 minutes ago           pedantic_shirley
252e67c499c1   centos    "/bin/bash" 13 minutes ago  Exited (0) 13 minutes ago          eloquent_blackburn
bae58ac1960f   ubuntu    "/bin/bash" 19 minutes ago  Exited (0) 19 minutes ago          unruffled_poitras
student@docker01:~$
```

Figure 24: Commande 'docker ps -a'

Docker stop

Un conteneur CentOS peut être exécuté avec la commande docker run et mis en veille avec une commande comme sleep 2000. Cela maintient le conteneur actif pendant 2000 secondes avant qu'il ne s'arrête automatiquement.

Pour arrêter un conteneur avant la fin, la commande `docker stop` peut être utilisée, en spécifiant soit l'ID du conteneur, soit son nom (par exemple, `jovial_banzai`). Une fois arrêté, le conteneur n'apparaît plus dans `docker ps`, mais reste visible avec `docker ps -a`. Le code de sortie du conteneur indique son état : 0 pour un arrêt normal ou 137 lorsqu'il est interrompu de force avec `docker stop`.

```
student@docker01:~$ sudo docker run -d centos sleep 2000
5e0665fe6af3e43a146c0eccc4ff14dd490746aebf72b3bf6b9a74e8147296ee
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
5e0665fe6af3   centos    "sleep 2000"             5 seconds ago       Up 5 seconds                jovial_banzai
student@docker01:~$ sudo docker stop jovial_banzai
jovial_banzai
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
5e0665fe6af3   centos    "sleep 2000"             About a minute ago   Exited (137) 57 seconds ago       jovial_banzai
a10f6f0c39a4   centos    "sleep 20"               7 minutes ago        Exited (0) 7 minutes ago          epic_khayyam
4660b10072be   centos    "bash"                  13 minutes ago       Exited (0) 11 minutes ago         pedantic_shirley
252e67c499c1   centos    "/bin/bash"             18 minutes ago       Exited (0) 18 minutes ago         eloquent_blackburn
bae58ac1960f   ubuntu    "/bin/bash"             23 minutes ago       Exited (0) 23 minutes ago         unruffled_poitras
student@docker01:~$
```

Figure 25: Commande 'docker stop'

Docker rm

Pour supprimer des conteneurs et libérer de l'espace disque, la commande `docker rm` peut être utilisée. Il est possible de spécifier un conteneur à l'aide de son ID ou de son nom. Plusieurs conteneurs peuvent être supprimés en une seule commande en listant leurs IDs ou noms.

Une fois les conteneurs supprimés, la commande `docker ps -a` permet de vérifier qu'ils ne sont plus présents.

```
student@docker01:~$ sudo docker rm unruffled_poitras
unruffled_poitras
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
5e0665fe6af3   centos    "sleep 2000"             4 minutes ago       Exited (137) 4 minutes ago       jovial_banzai
a10f6f0c39a4   centos    "sleep 20"               10 minutes ago      Exited (0) 10 minutes ago        epic_khayyam
4660b10072be   centos    "bash"                  16 minutes ago      Exited (0) 14 minutes ago         pedantic_shirley
252e67c499c1   centos    "/bin/bash"             21 minutes ago      Exited (0) 21 minutes ago         eloquent_blackburn
student@docker01:~$
```

Figure 26: Commande 'docker rm'

```
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
5e0665fe6af3   centos    "sleep 2000"             6 minutes ago       Exited (137) 6 minutes ago       jovial_banzai
a10f6f0c39a4   centos    "sleep 20"               12 minutes ago      Exited (0) 12 minutes ago        epic_khayyam
4660b10072be   centos    "bash"                  18 minutes ago      Exited (0) 16 minutes ago         pedantic_shirley
252e67c499c1   centos    "/bin/bash"             23 minutes ago      Exited (0) 23 minutes ago         eloquent_blackburn
student@docker01:~$ sudo docker rm a10f6f0c39a4
a10f6f0c39a4
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
5e0665fe6af3   centos    "sleep 2000"             7 minutes ago       Exited (137) 6 minutes ago       jovial_banzai
4660b10072be   centos    "bash"                  19 minutes ago      Exited (0) 17 minutes ago         pedantic_shirley
252e67c499c1   centos    "/bin/bash"             24 minutes ago      Exited (0) 24 minutes ago         eloquent_blackburn
student@docker01:~$ sudo docker rm 5e 46 25
5e
46
25
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS              NAMES
student@docker01:~$
```

Figure 27: Utilisations de 'docker rm'

Docker images

Pour afficher les images Docker actuellement disponibles, la commande `docker images` peut être utilisée. Cette commande liste toutes les images présentes sur le système, avec leurs noms, tags, et tailles.

```
student@docker01:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest   f876bfc1cc63  4 weeks ago   192MB
ubuntu        latest   b1d9df8ab815  5 weeks ago   78.1MB
centos        latest   5d0da3dc9764  3 years ago   231MB
```

Figure 28: Commande 'docker image'

Pour supprimer des images Docker, la commande `docker rmi` est utilisée. Par exemple, `docker rmi nginx` supprime l'image `nginx`, et `docker rmi ubuntu` supprime l'image `ubuntu`.

```
student@docker01:~$ sudo docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:42e917aaab1b5bb40dd0f6f7f4f857490ac7747d7ef73b391c774a41a8b994f15
Deleted: sha256:f876bfc1cc63d905bb9c8ebc5adc98375bb8e22920959719d1a96e8f594868fa
Deleted: sha256:e0f1c40b04bce92241b6a81812e29990b5ff711bfd3fb6817e4ec03f3f09cb72
Deleted: sha256:f3764bf5781131fa5df6d78bcde2d5905c7d2451b7084ff73d6cbb679fa2a573
Deleted: sha256:d0516283d34466f5e50f48966ce9c304cb665bd9a6fe984ca21166ff6d519264
Deleted: sha256:bf29edbdcd853fb80ac4623db054796c3050017aa68dd79958ab61762fa85dc
Deleted: sha256:344c966b8cc1774f55cf5f6fb3c438c497a2a84d4e9e09befc7e1623f97029bf
Deleted: sha256:59db063f63f68b942f3c60769828c15efe9abd12362d5c6d925a0484bbf031d0
Deleted: sha256:8b296f48696071aafb5a6286ca60d441a7e559b192fc7f94bb63ee93dae98f17
student@docker01:~$ sudo docker rmi ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Deleted: sha256:b1d9df8ab81559494794e522b380878cf9ba82d4c1fb67293bcf931c3aa69ae4
Deleted: sha256:687d50f2f6a697da02e05f2b2b9cb05c1d551f37c404e55fdec44b0ae8aa5c
student@docker01:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
centos        latest   5d0da3dc9764  3 years ago   231MB
student@docker01:~$
```

Figure 29: Commande 'docker rmi'

Si une image est utilisée par un conteneur existant, sa suppression échoue. Dans ce cas, le conteneur dépendant doit d'abord être supprimé avec la commande `docker rm`. Une fois le conteneur supprimé, l'image associée peut être retirée avec `docker rmi`. Cette méthode garantit un nettoyage complet des images inutilisées.

```
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
student@docker01:~$ sudo docker run centos
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
bc13c8323064   centos    "/bin/bash"   3 seconds ago   Exited (0) 2 seconds ago   agitated_wiles
student@docker01:~$
```

Figure 30: Vérification de l'utilisation des images Docker

Elle est utilisée par un conteneur, sa suppression sera impossible.

```
student@docker01:~$ sudo docker rmi centos
Error response from daemon: conflict: unable to remove repository reference "centos" (must force) - container bc13c8323064 is using its referenced image 5d0da3dc9764
student@docker01:~$
```

Figure 31: Erreur suppression image

Il est nécessaire de supprimer d'abord le conteneur associé avec la commande docker rm.

```
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
bc13c8323064   centos    "/bin/bash"             3 minutes ago   Exited (0) 3 minutes ago           agitated_wiles
student@docker01:~$ sudo docker rm bc
bc
student@docker01:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
student@docker01:~$ sudo docker rmi centos
Untagged: centos:latest
Untagged: centos@sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Deleted: sha256:5d0da3dc976460b72c77d94c8a1ad043720b0416bfc16c52c45d4847e53fadb6
Deleted: sha256:74ddd0ec08fa43d09f32636ba91a0a3053b02cb4627c35051aff89f853606b59
student@docker01:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
student@docker01:~$
```

Figure 32: Supprimer le conteneur avant l'image

C. Docker Run pour lancer et gérer des conteneurs

Pour utiliser une image spécifique sur Docker, il est d'abord nécessaire de visiter le site [Docker Hub](#) et d'identifier le dépôt concerné, comme celui d'Ubuntu. Chaque dépôt liste des tags supportés, correspondant aux différentes versions disponibles. Par défaut, si aucun tag n'est spécifié, Docker utilise le tag latest. Par exemple, la commande docker run ubuntu télécharge automatiquement la version 24.04, correspondant au tag latest

Supported tags and respective `Dockerfile` links

- [20.04](#) [focal-20241011](#) [focal](#) [↗](#)
- [22.04](#) [jammy-20240911.1](#) [jammy](#) [↗](#)
- [24.04](#) [noble-20241118.1](#) [noble](#) [latest](#) [↗](#)
- [24.10](#) [oracular-20241120](#) [oracular](#) [rolling](#) [↗](#)
- [25.04](#) [plucky-20241213](#) [plucky](#) [devel](#) [↗](#)

Figure 33: Liste des tags supportés

```
student@docker01:~$ sudo docker run ubuntu cat /etc/*release*
[sudo] password for student:
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=24.04
DISTRIB_CODENAME=noble
DISTRIB_DESCRIPTION="Ubuntu 24.04.1 LTS"
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

Figure 34: Affichage des informations de version Ubuntu

Pour exécuter une version spécifique, comme 20.04, le tag doit être ajouté après le nom de l'image, séparé par un deux-points, par exemple : `docker run ubuntu:20.04`. Si l'image avec ce tag n'est pas disponible localement, Docker la télécharge automatiquement depuis Docker Hub.

```
student@docker01:~$ sudo docker run ubuntu:20.04 cat /etc/*release*
Unable to find image 'ubuntu:20.04' locally
20.04: Pulling from library/ubuntu
d9802f032d67: Pull complete
Digest: sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44affc1cd27555141b
Status: Downloaded newer image for ubuntu:20.04
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.6 LTS"
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
student@docker01:~$
```

Figure 35: Commande avec la version

L'image correspondant à la version spécifiée a été correctement téléchargée, car un tag différent a été fourni.

Commande -d

Avec la commande `sleep 15`, le conteneur s'arrête automatiquement après 15 secondes, mais aucune action n'est possible depuis la console pendant ce temps. Pour éviter ce blocage, un conteneur peut être lancé en mode détaché (en arrière-plan) en ajoutant l'option `-d`. Cela permet au conteneur de s'exécuter en arrière-plan, tout en laissant la console accessible immédiatement.

Exécuter NGINX

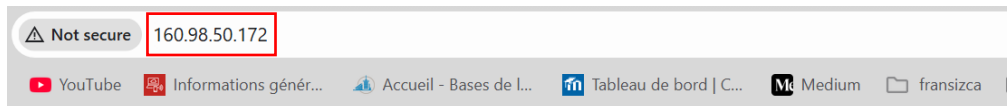
```
student@docker01:~$ sudo docker run ubuntu sleep 15
^C^C^C
got 3 SIGTERM/SIGINTs, forcefully exiting
student@docker01:~$ sudo docker run -d ubuntu sleep 1500
0ce9fcd024325b29a879221d9819f9c0659e1f15499b1fd0e31d5034afa54b21
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
0ce9fcd02432   ubuntu   "sleep 1500"            11 seconds ago   Up 10 seconds          funny_panini
student@docker01:~$
```

Figure 36: Commande -d

Pour exécuter NGINX, la commande `docker run nginx` peut être utilisée. Cette commande télécharge l'image NGINX et ses couches, puis lance une instance du serveur web NGINX. Une fois l'image téléchargée et extraite, NGINX démarre automatiquement. Il est ensuite possible d'accéder au serveur web via un navigateur en entrant l'URL appropriée.

```
student@docker01:~$ sudo docker run -p 80:80 -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fd674058ff8f: Pull complete
566e42bcee1c: Pull complete
2b99b9c5d9e5: Pull complete
bd98674871f5: Pull complete
1e109dd2a0d7: Pull complete
da8cc133ff82: Pull complete
c44f27309ea1: Pull complete
Digest: sha256:42e917aaa1b5bb40dd0f6f7f4f857490ac7747d7ef73b391c774a41a8b994f15
Status: Downloaded newer image for nginx:latest
f905e17d17b8ac18cb5aa4424cda951de68f0fdaa15e4a40102e56259ef8782
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
f905e17d17b8   nginx    "/docker-entrypoint..." 17 seconds ago Up 14 seconds 0.0.0.0:80->80/tcp, :::80->80/tcp   jovial_bassi
student@docker01:~$
```

Figure 37: Commande -p



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 38: Contrôle de l'installation NGINX

Exécuter Jenkins/Jenkins

```
student@docker01:~$ sudo docker run -p 8080:8080 jenkins/jenkins
```

Figure 39: Commande 'docker run' pour jenkins/jenkins

```
student@docker01:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
0ae824f37f9a   jenkins/jenkins  "/usr/bin/tini -- /u..." 2 minutes ago Up 2 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 50000/tcp   silly_wilbur
student@docker01:~$
```

Figure 40: Contrôle des ports avec la commande 'docker ps'

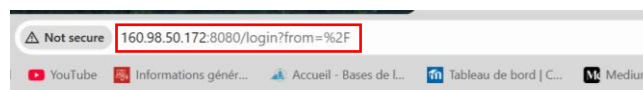


Figure 41: Connexion Jenkins via IP

En spécifiant les ports avec -p, il est possible de mapper un port du conteneur à un port spécifique de l'hôte. Cela garantit que le service est accessible via le port choisi sur la machine

hôte. Si les ports ne sont pas mappés, Docker peut attribuer un port aléatoire, ce qui complique l'accès au service, car l'utilisateur doit trouver le port assigné automatiquement pour se connecter.

D. Création et publication d'images Docker sur Docker Hub

Création d'un fichier Dockerfile

Le fichier Dockerfile est utilisé pour automatiser la création d'images Docker. Il définit les instructions nécessaires (comme la base du système, les logiciels à installer, et les commandes à exécuter) pour construire une image personnalisée de manière reproductible et standardisée.

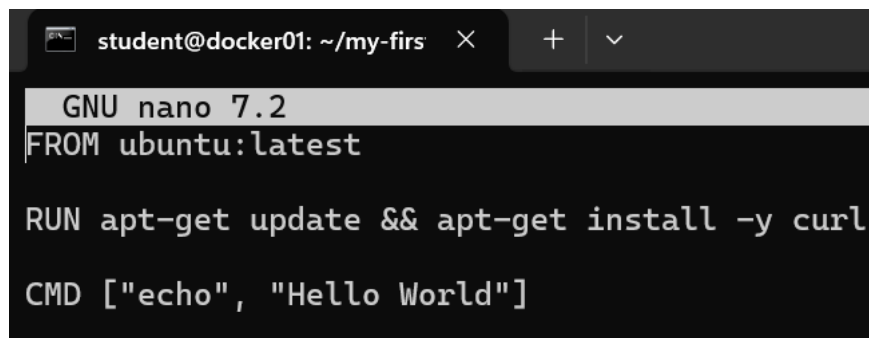
Un fichier nommé Dockerfile est créé dans un dossier spécifique (my-first-container).

```
student@docker01:~$ cd my-first-container
student@docker01:~/my-first-container$ ls
Dockerfile
student@docker01:~/my-first-container$ |
```

Figure 42: Dockerfile

Contenu du Dockerfile:

Ce fichier utilise l'image de base ubuntu:latest, exécute une mise à jour du système, installe curl et définit une commande par défaut pour afficher "Hello World".



```
student@docker01: ~/my-firs × + v
GNU nano 7.2
FROM ubuntu:latest

RUN apt-get update && apt-get install -y curl

CMD ["echo", "Hello World"]
```

Figure 43: Contenu du Dockerfile

Construction de l'image Docker

Une image est créée à partir du Dockerfile avec la commande :

Cette commande crée une image appelée hello en utilisant le contenu du Dockerfile.

docker buildx build : Utilise l'outil Buildx de Docker pour construire une image.

-t hello : Définit le tag (nom) de l'image créée. Ici, l'image est appelée hello.

. (**point**) : Cela indique que tous les fichiers nécessaires à la construction de l'image (y compris le Dockerfile) se trouvent dans le répertoire actuel.

```
student@docker01:~/my-first-container$ docker buildx build --network=host --build-arg BUILDKIT_INLINE_CACHE=1 -t hello .
[+] Building 0.2s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 164B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/2] FROM docker.io/library/ubuntu:latest
=> CACHED [2/2] RUN apt-get update && apt-get install -y curl
=> exporting to image
=> => exporting layers
=> => preparing layers for inline cache
=> => writing image sha256:36bef0f6ea84f166bfe531f7f071412991693cd48e05beccc32cfd054c8366ae
=> => naming to docker.io/library/hello
student@docker01:~/my-first-container$
```

Figure 44: Commande 'docker build'

Test de l'image localement :

Le résultat attendu est l'affichage du message "Hello World".

```
student@docker01:~/my-first-container$ docker run hello
Hello World
```

Figure 45: Test de l'image localement

Publication de l'image sur Docker Hub :

Un tag est nécessaire pour pousser une image Docker sur Docker Hub, car il identifie l'image et le dépôt où elle doit être envoyée.

L'image est taguée pour correspondre au nom du dépôt Docker Hub :

```
student@docker01:~/my-first-container$ docker tag hello rabiasevinc/hello:latest
```

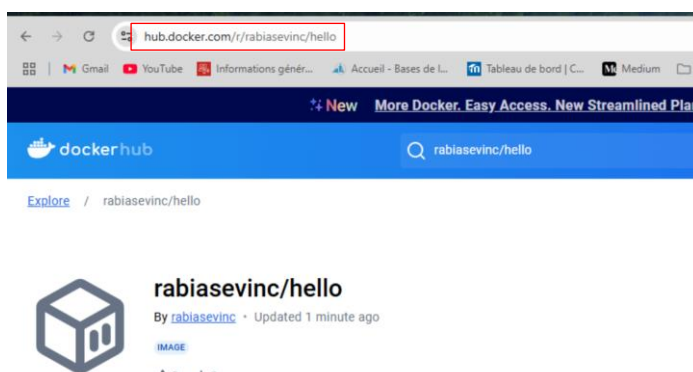
Figure 46: Création d'un tag pour l'image Docker

L'image est ensuite poussée sur Docker Hub avec la commande docker push

```
student@docker01:~/my-first-container$ docker push rabiasevinc/hello
Using default tag: latest
The push refers to repository [docker.io/rabiasevinc/hello]
cb3340fc533a: Pushed
687d50f2f6a6: Pushed
latest: digest: sha256:6d065033a3b6fe97829b56ebe2d16e53d7f6dd5a0d88482389ef89b3ec52f701 size: 741
```

Figure 47: Commande 'docker push'

Une fois publiée, l'image est disponible sur la page Docker Hub correspondante (rabiasevinc/hello).



<https://hub.docker.com/r/rabiasevinc/hello>

Figure 48: Image sur Docker Hub

E. Déploiement d'une application Flask basée sur Python

Création d'un environnement virtuel Python

Un environnement virtuel nommé myenv a été créé avec la commande `sudo python3 -m venv myenv`

L'environnement a été activé avec la commande `source myenv/bin/activate`

Flask a ensuite été installé dans cet environnement virtuel

```
student@docker01:~$ sudo python3 -m venv myenv
student@docker01:~$ source myenv/bin/activate
(myenv) student@docker01:~$ pip install flask
```

Figure 49: Création et activation d'un environnement virtuel Python

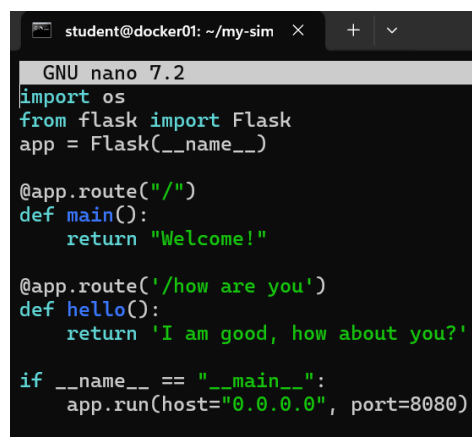
Création et test de l'application Flask

Un fichier Python nommé `app.py` a été créé avec le contenu suivant

Le fichier a été placé dans un répertoire spécifique.

```
(myenv) student@docker01:~$ cd opt
(myenv) student@docker01:~/opt$ ls
__pycache__  app.py
(myenv) student@docker01:~/opt$ |
```

Figure 50: Création `app.py`



```
student@docker01: ~/my-sim × + v
GNU nano 7.2
import os
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return "Welcome!"

@app.route('/how are you')
def hello():
    return 'I am good, how about you?'

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Figure 51: Contenu de `app.py`

Création d'un Dockerfile

Un fichier `Dockerfile` a été créé pour contenir les instructions de construction de l'image Docker.

Le fichier `app.py` est également créé dans le même répertoire que le fichier Docker.


```
student@docker01:~$ source myenv/bin/activate
(myenv) student@docker01:~$ cd my-simple-container
(myenv) student@docker01:~/my-simple-container$ ls
Dockerfile  app.py
(myenv) student@docker01:~/my-simple-container$ |
```

Figure 52: Création Dockerfile and app.py

Voici le contenu de Dockerfile :

```
student@docker01: ~/my-sim  X  +  v
GNU nano 7.2
FROM ubuntu

RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip \
    python3-venv \
    && apt-get clean

RUN python3 -m venv /opt/myenv
RUN /opt/myenv/bin/pip install --upgrade pip
RUN /opt/myenv/bin/pip install flask

COPY app.py /opt/app.py

ENTRYPOINT ["/opt/myenv/bin/python3", "/opt/app.py"]
```

Figure 53: Contenu de Dockerfile

L'image Docker a été construite avec la commande suivante :

```
(myenv) student@docker01:~/my-simple-container$ (myenv) student@docker01:~/my-simple-container$ docker buildx build --ne
twork=host --build-arg BUILDKIT_INLINE_CACHE=1 -t simple_webapp .
[+] Building 58.2s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 531B                                0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest   0.0s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> CACHED [1/6] FROM docker.io/library/ubuntu:latest              0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 28B                                       0.0s
=> [2/6] RUN apt-get update && apt-get install -y python3 python3-pip python3-venv && apt-get c 46.2s
=> [3/6] RUN python3 -m venv /opt/myenv                             3.7s
=> [4/6] RUN /opt/myenv/bin/pip install --upgrade pip              2.5s
=> [5/6] RUN /opt/myenv/bin/pip install flask                      2.4s
=> [6/6] COPY app.py /opt/app.py                                    0.1s
=> exporting to image                                              3.0s
=> => exporting layers                                              2.9s
=> => preparing layers for inline cache                             0.0s
=> => writing image sha256:a15911b828ca421c6de4cac94eab1374256ba4ed20e8ac37de6ebc9c6e2d981a 0.0s
=> => naming to docker.io/library/simple_webapp                    0.0s
(myenv) student@docker01:~/my-simple-container$ |
```

Figure 54: Création d'une image (docker build)

L'image créée est visible avec la commande docker images

```
(myenv) student@docker01:~/my-simple-container$ docker images
REPOSITORY          TAG         IMAGE ID        CREATED         SIZE
simple_webapp        latest     a15911b828ca   57 seconds ago  578MB
```

Figure 55: Vérification de l'image créée avec docker build

Lancement du conteneur Docker

Le conteneur a été lancé en exposant le port 8080 avec la commande docker run -p 8080:8080 simple_webapp

```
(myenv) student@docker01:~/my-simple-container$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
(myenv) student@docker01:~/my-simple-container$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
simple_webapp   latest    a15911b828ca   44 minutes ago  578MB
rabiasevinc/simple_webapp   latest    a15911b828ca   44 minutes ago  578MB
hello         latest    36bef0f6ea84   14 hours ago   132MB
rabiasevinc/hello         latest    36bef0f6ea84   14 hours ago   132MB
(myenv) student@docker01:~/my-simple-container$ docker run hello
Hello World
(myenv) student@docker01:~/my-simple-container$ docker run -p 8080:8080 simple_webapp
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://172.17.0.2:8080
Press Ctrl+C to quit
169.98.166.173 - - [31/Dec/2024 09:59:26] "GET / HTTP/1.1" 200 -
169.98.166.173 - - [31/Dec/2024 09:59:40] "GET /how%20are%20you HTTP/1.1" 200 -
(myenv) student@docker01:~/my-simple-container$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
25ab76f5417b   simple_webapp   "/opt/myenv/bin/python" 54 seconds ago  Exited (0) 4 seconds ago        peaceful_poitr
as
3e93d4adbbd9   hello         "echo 'Hello World'"     About a minute ago  Exited (0) About a minute ago        affectionate_p
erlman
(myenv) student@docker01:~/my-simple-container$
```

Figure 56: Lancement du conteneur simple_webapp

Accès à l'application via le navigateur

L'application est accessible à l'adresse suivante :

<http://160.98.50.172:8080>

L'application affiche "Welcome!" lorsqu'on accède à la racine (/) et "I am good, how about you?" à l'URL /how are you ?

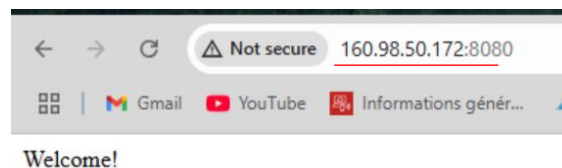


Figure 57: Accès à l'application via le navigateur

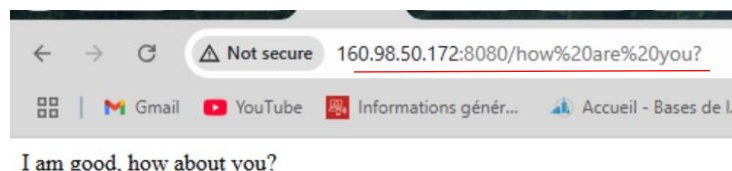


Figure 58: Accès à l'application via le navigateur 2

Elle est maintenant déployée dans un conteneur Docker et prête à être utilisée.

L'image est ensuite poussée sur Docker Hub.

```
(myenv) student@docker01:~/my-simple-container$ docker push rabiasevinc/simple_webapp
Using default tag: latest
The push refers to repository [docker.io/rabiasevinc/simple_webapp]
891d53963e66: Pushed
62960a85cff2: Pushed
c468bfe41614: Pushed
f9d0f93e3118: Pushed
3ab6e8a24481: Pushed
687d50f2f6a6: Pushed
latest: digest: sha256:e86d1ced7a65d20de362f8d9aee42ec54b58d5920c216a87fb52dc072957bbde size: 1582
(myenv) student@docker01:~/my-simple-container$
```

Figure 59: Publication de l'image sur Docker Hub

Une fois publiée, l'image est disponible sur la page Docker Hub correspondante (rabiasevinc/simple_webapp).

https://hub.docker.com/r/rabiasevinc/simple_webapp

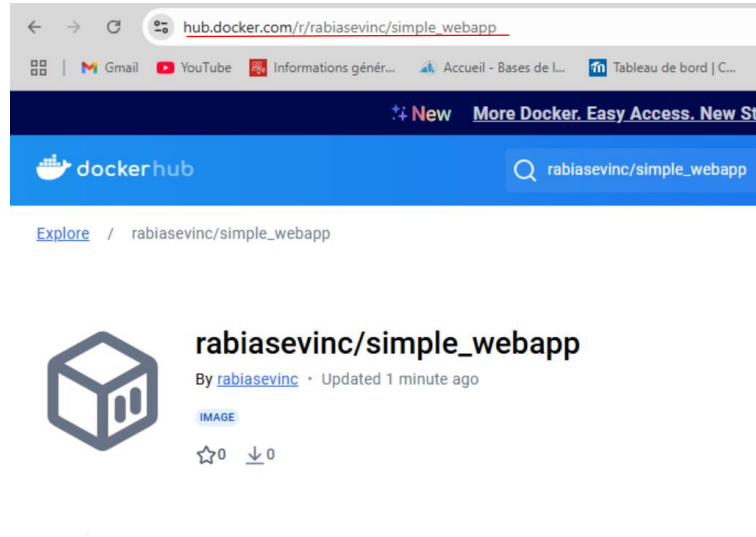


Figure 60: Image simple_webapp sur Docker Hub

F. Docker Compose

Docker Compose est un outil qui permet de gérer plusieurs conteneurs en même temps avec un seul fichier YAML. Dans ce fichier, chaque service (application) est défini avec ses propriétés comme l'image Docker, les ports, et les connexions entre les services.

L'Exemple Voting App est disponible sur GitHub dans le dépôt [Docker samples](#), sous le nom example-voting-app. Cette application de vote est un exemple illustrant une architecture d'application composée de plusieurs composants. Elle offre une interface permettant à un utilisateur de voter et une autre pour afficher les résultats.

L'application comprend :

voting-app : une application web en Python, permettant de choisir entre deux options (chat ou chien). Le vote est ensuite stocké dans Redis, qui sert de base de données en mémoire.

worker : une application en .NET qui traite les votes en mettant à jour une base de données persistante PostgreSQL. Cette dernière contient un tableau indiquant le nombre de votes pour chaque catégorie (chats et chiens).

result-app : une application web en Node.js qui lit les données de PostgreSQL et affiche les résultats à l'utilisateur.

Architecture

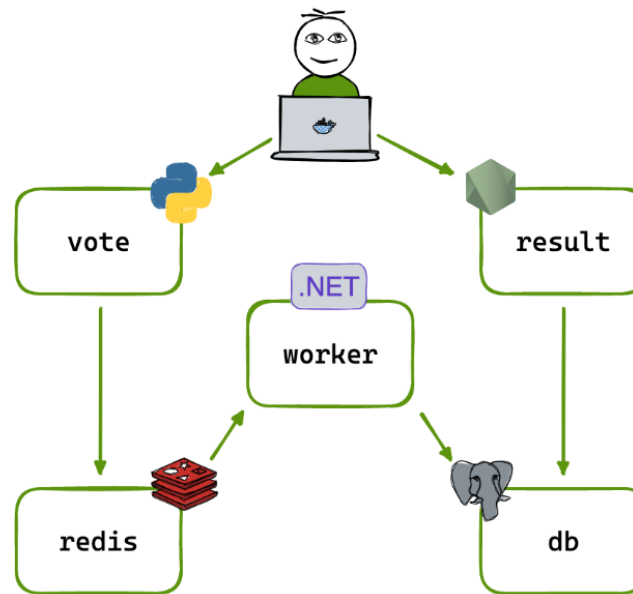


Figure 61: Architecture de l'application de vote

Le dépôt GitHub de l'application de vote est cloné localement en utilisant la commande suivante :

```
student@docker01:~/sample-application$ git clone https://github.com/docker-samples/example-voting-app.git
Cloning into 'example-voting-app'...
remote: Enumerating objects: 1179, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 1179 (delta 8), reused 4 (delta 4), pack-reused 1158 (from 2)
Receiving objects: 100% (1179/1179), 1.21 MiB | 19.05 MiB/s, done.
Resolving deltas: 100% (440/440), done.
student@docker01:~/sample-application$
```

Figure 62: Clonage du projet depuis GitHub

Le code source est placé dans le répertoire **example-voting-app**. Après le clonage, la navigation dans le répertoire **example-voting-app** permet d'accéder à tous les fichiers nécessaires, notamment le fichier **docker-compose.yml** et les répertoires des composants.

Dans le répertoire **vote**, le fichier **Dockerfile** est vérifié pour s'assurer qu'il est intact et prêt pour la création d'une image Docker.

```
student@docker01:~/sample-application/example-voting-app$ ls -al
total 232
drwxrwxr-x 11 student student 4096 Jan  3 14:13 .
drwxrwxr-x  3 student student 4096 Jan  3 14:13 ..
drwxrwxr-x  8 student student 4096 Jan  3 14:13 .git
-rw-rw-r--  1 student student 494 Jan  3 14:13 .gitattributes
drwxrwxr-x  3 student student 4096 Jan  3 14:13 .github
-rw-rw-r--  1 student student  53 Jan  3 14:13 .gitignore
drwxrwxr-x  2 student student 4096 Jan  3 14:13 .vscode
-rw-rw-r--  1 student student 10758 Jan  3 14:13 LICENSE
-rw-rw-r--  1 student student  288 Jan  3 14:13 MAINTAINERS
-rw-rw-r--  1 student student 2472 Jan  3 14:13 README.md
-rw-rw-r--  1 student student 151461 Jan  3 14:13 architecture.excalidraw.png
-rw-rw-r--  1 student student 1346 Jan  3 14:13 docker-compose.images.yml
-rw-rw-r--  1 student student 1946 Jan  3 14:13 docker-compose.yml
-rw-rw-r--  1 student student  982 Jan  3 14:13 docker-stack.yml
drwxrwxr-x  2 student student 4096 Jan  3 14:13 healthchecks
drwxrwxr-x  2 student student 4096 Jan  3 14:13 k8s-specifications
drwxrwxr-x  4 student student 4096 Jan  3 14:13 result
drwxrwxr-x  2 student student 4096 Jan  3 14:13 seed-data
drwxrwxr-x  4 student student 4096 Jan  3 14:13 vote
drwxrwxr-x  2 student student 4096 Jan  3 14:13 worker
student@docker01:~/sample-application/example-voting-app$ cd vote/
student@docker01:~/sample-application/example-voting-app/vote$ ls
Dockerfile  app.py  requirements.txt  static  templates
student@docker01:~/sample-application/example-voting-app/vote$ |
```

Figure 63: Vérification du Dockerfile

La commande suivante est utilisée pour construire l'image Docker pour le composant de vote :

```
student@docker01:~/sample-application/example-voting-app/vote$ docker buildx build --network=host --build-arg BUILDKIT_INLINE_CACHE=1 -t voting-app .
[+] Building 11.0s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 1.08kB                             0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 0.8s
=> [auth] library/python:pull token for registry-1.docker.io      0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> CACHED [base 1/5] FROM docker.io/library/python:3.11-slim@sha256:873952659a04188d2a62d5f7e30fd673d2559432a847 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 274B                                      0.0s
=> [base 2/5] RUN apt-get update && apt-get install -y --no-install-recommends curl && rm -rf /var/lib/a 4.9s
=> [base 3/5] WORKDIR /usr/local/app                             0.1s
=> [base 4/5] COPY requirements.txt ./requirements.txt            0.1s
=> [base 5/5] RUN pip install --no-cache-dir -r requirements.txt 4.3s
=> [final 1/1] COPY . .                                           0.1s
=> exporting to image                                              0.4s
=> => exporting layers                                              0.3s
=> => preparing layers for inline cache                            0.0s
=> => writing image sha256:46b9169f73ea2420f96b5f11df25fe1c5e6ed78adfc2a4aa4736565b401f94c 0.0s
=> => naming to docker.io/library/voting-app                      0.0s
student@docker01:~/sample-application/example-voting-app/vote$ |
```

Figure 64: Construction de l'image dans le répertoire contenant le Dockerfile

Une fois la construction terminée, l'image apparaît dans la liste des images Docker.

```
student@docker01:~/sample-application/example-voting-app/vote$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
voting-app          latest     46b9169f73ea  46 seconds ago 153MB
simple_webapp        latest     a15911b828ca  3 days ago    578MB
rabiasevinc/simple_webapp latest     a15911b828ca  3 days ago    578MB
hello               latest     36bef0f6ea84  3 days ago    132MB
rabiasevinc/hello   latest     36bef0f6ea84  3 days ago    132MB
student@docker01:~/sample-application/example-voting-app/vote$ |
```

Figure 65: Vérification de l'image voting-app

Le conteneur est lancé avec la commande suivante, en mappant le port 5000 de l'hôte au port 80 du conteneur :

```
student@docker01:~/sample-apps/application/example-voting-app/votes$ docker run -p 5000:80 voting-app
[2025-01-03 14:24:54 +0000] [1] [INFO] Starting unicorn 23.0.0
[2025-01-03 14:24:54 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
[2025-01-03 14:24:54 +0000] [1] [INFO] Using worker: sync
[2025-01-03 14:24:54 +0000] [7] [INFO] Booting worker with pid: 7
[2025-01-03 14:24:54 +0000] [8] [INFO] Booting worker with pid: 8
[2025-01-03 14:24:54 +0000] [9] [INFO] Booting worker with pid: 9
[2025-01-03 14:24:54 +0000] [10] [INFO] Booting worker with pid: 10
172.17.0.1 -- [03/Jan/2025:14:27:47 +0000] "GET / HTTP/1.0" 200 1285 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36"
172.17.0.1 -- [03/Jan/2025:14:27:47 +0000] "GET /static/stylesheets/style.css HTTP/1.0" 200 0 "http://160.98.50.172/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36"
172.17.0.1 -- [03/Jan/2025:14:27:48 +0000] "GET /favicon.ico HTTP/1.0" 404 207 "http://160.98.50.172/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36"
```

Figure 66: Exécution de l'image de vote

L'application est accessible via un navigateur à l'adresse <http://160.98.50.172:80>

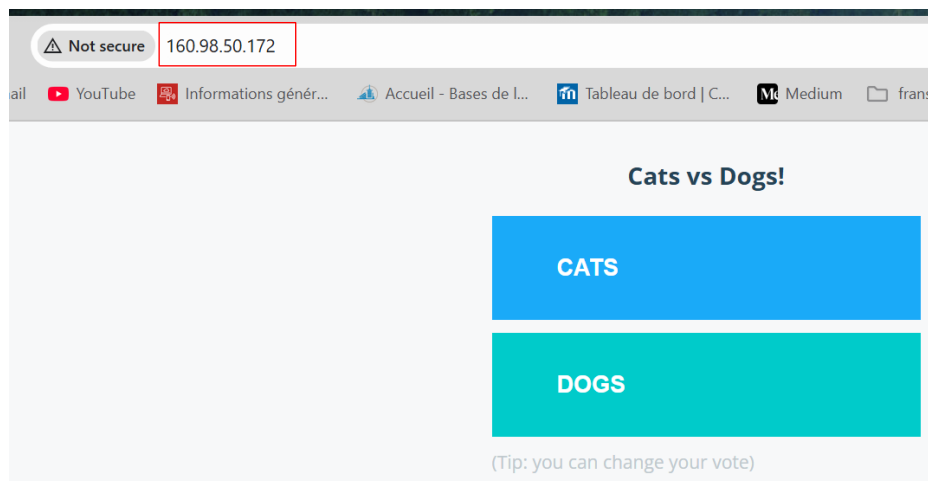
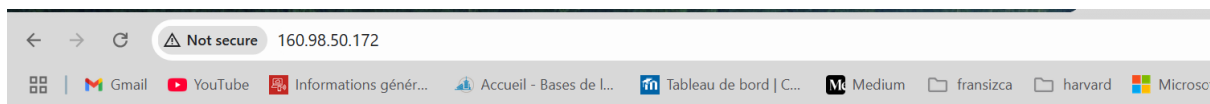


Figure 67: Accès à l'application voting-app via le navigateur

L'interface de l'application est affichée, permettant de voter pour Cats ou Dogs. Cependant, après avoir cliqué sur un bouton, une erreur Internal Server Error apparaît.



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Figure 68: Erreur après interaction avec l'application

Les journaux révèlent une erreur à la ligne `redis.rpush`, indiquant que l'application n'a pas pu se connecter à Redis. Cela est dû au fait qu'aucun conteneur Redis n'est actuellement en cours d'exécution.

```
File "/usr/local/app/app.py", line 37, in hello
    redis.rpush('votes', data)
File "/usr/local/lib/python3.11/site-packages/redis/commands/core.py", line 2818, in rpush
    return self.execute_command("RPUSH", name, *values)
*****
```

Figure 69: Erreur de connexion à Redis

Une instance de Redis est lancée avec la commande :

```
student@docker01:~/sample-application/example-voting-app/vote$ docker run --name=redis redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
fd674058ff8f: Already exists
f3615eb0792b: Pull complete
e0b2e54213f5: Pull complete
d334c6665cc1: Pull complete
47cee545c70a: Pull complete
76f0f8a4aae4: Pull complete
4f4fb700ef54: Pull complete
4c9c306fe0ac: Pull complete
Digest: sha256:bb142a9c18ac18a16713c1491d779697b4e107c22a97266616099d288237ef47
Status: Downloaded newer image for redis:latest
```

Figure 70: Exécution de l'image de redis

Une fois Redis actif, une instance de l'application de vote est démarrée avec un lien vers Redis :

```
student@docker01:~/sample-application/example-voting-app/vote$ docker run -p 5000:80 --link redis:redis voting-app
[2025-01-03 14:34:50 +0000] [1] [INFO] Starting gunicorn 23.0.0
[2025-01-03 14:34:50 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
[2025-01-03 14:34:50 +0000] [1] [INFO] Using worker: sync
[2025-01-03 14:34:50 +0000] [7] [INFO] Booting worker with pid: 7
[2025-01-03 14:34:50 +0000] [8] [INFO] Booting worker with pid: 8
[2025-01-03 14:34:50 +0000] [9] [INFO] Booting worker with pid: 9
[2025-01-03 14:34:50 +0000] [10] [INFO] Booting worker with pid: 10
172.17.0.1 - - [03/Jan/2025:14:34:58 +0000] "GET / HTTP/1.0" 200 1285 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0 Safari/537.36"
172.17.0.1 - - [03/Jan/2025:14:34:58 +0000] "GET /static/stylesheets/style.css HTTP/1.0" 304 0 "http://160.98.50.172/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0 Safari/537.36"
[2025-01-03 14:35:14,129] INFO in app: Received vote for a
[2025-01-03 14:35:14 +0000] [10] [INFO] Received vote for a
172.17.0.1 - - [03/Jan/2025:14:35:14 +0000] "POST / HTTP/1.0" 200 1688 "http://160.98.50.172/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0 Safari/537.36"
172.17.0.1 - - [03/Jan/2025:14:35:14 +0000] "GET /static/stylesheets/style.css HTTP/1.0" 304 0 "http://160.98.50.172/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0 Safari/537.36"
```

Figure 71: Exécution de voting-app avec connexion au conteneur Redis.

Le port 5000 de l'hôte est mappé au port 80 du conteneur. L'application de vote fonctionne correctement et peut communiquer avec Redis.

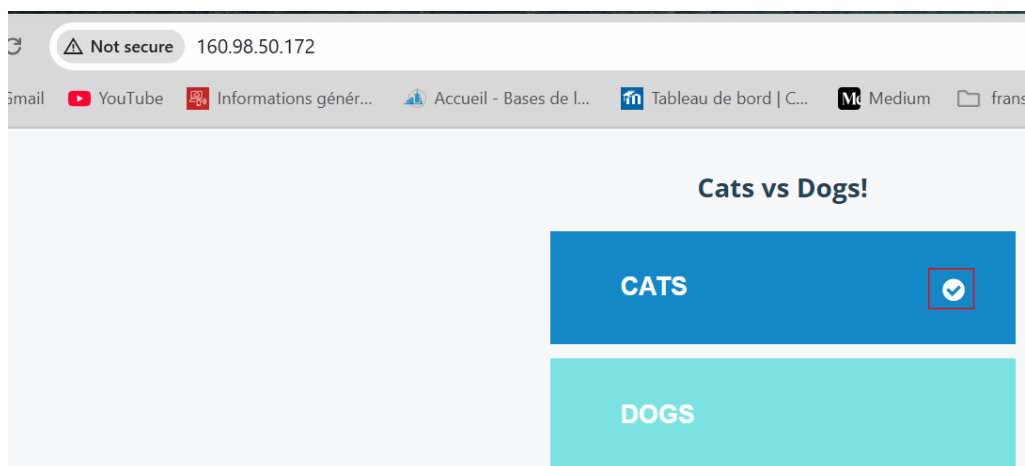


Figure 72: L'application de vote fonctionne

Une instance de PostgreSQL est lancée :


```
student@docker01:~/sample-application/example-voting-app/vote$ docker run -d --name=db -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres postgres:15-alpine
Unable to find image 'postgres:15-alpine' locally
15-alpine: Pulling from library/postgres
38a8310d387e: Pull complete
e83e247a335b: Pull complete
```

Figure 73: Exécution de l'image PostgreSQL

L'image du worker est construite dans le répertoire worker :

```
student@docker01:~/sample-application/example-voting-app/worker$ docker buildx build --network=host --build-arg BUILDKIT_INLINE_CACHE=1 -t worker-app .
[+] Building 30.3s (16/16) FINISHED
docker:default
```

Figure 74: Construction de l'image worker-app

Une fois l'image prête, une instance est lancée avec des liens vers Redis et PostgreSQL :

```
student@docker01:~/sample-application/example-voting-app/worker$ docker run -d --link redis:redis --link db:db worker-app
8a35f91170bac0cb5d1b6f39d61495d58205e61eb8d569dac36717b2492f0dc1
student@docker01:~/sample-application/example-voting-app/worker$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
8a35f91170ba   worker-app                          "dotnet Worker.dll"     4 seconds ago Up 3 seconds
291dabeeaf5e   postgres:15-alpine                 "docker-entrypoint.s..." 3 minutes ago Up 2 minutes   5432/tcp
99a460f9f93e   voting-app                          "unicorn app:app -b..." 16 minutes ago Up 16 minutes   0.0.0.0:5000->80/tcp, [::]:5000->80/tcp
f163705ad746   redis                               "docker-entrypoint.s..." 32 minutes ago Up 32 minutes   6379/tcp
```

Figure 75: Exécution de worker-app avec connexion à Redis et PostgreSQL

L'image de l'application de résultats est construite dans le répertoire result :

```
student@docker01:~/sample-application/example-voting-app/worker$ cd ..
student@docker01:~/sample-application/example-voting-app$ ls
LICENSE      README.md    docker-compose.images.yml  docker-stack.yml  k8s-specifications  seed-data  worker
MAINTAINERS  architecture.excalidraw.png  docker-compose.yml        healthchecks      result              vote
student@docker01:~/sample-application/example-voting-app$ cd result
student@docker01:~/sample-application/example-voting-app/result$ ls -l
total 92
-rw-rw-r-- 1 student student 486 Jan 3 14:13 Dockerfile
-rw-rw-r-- 1 student student 852 Jan 3 14:13 docker-compose.test.yml
-rw-rw-r-- 1 student student 68438 Jan 3 14:13 package-lock.json
-rw-rw-r-- 1 student student 414 Jan 3 14:13 package.json
-rw-rw-r-- 1 student student 1820 Jan 3 14:13 server.js
drwxrwxr-x 2 student student 4096 Jan 3 14:13 tests
drwxrwxr-x 3 student student 4096 Jan 3 14:13 views
student@docker01:~/sample-application/example-voting-app/result$ docker buildx build --network=host --build-arg BUILDKIT_INLINE_CACHE=1 -t result-app .
[+] Building 20.1s (13/13) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 525B
=> [internal] load metadata for docker.io/library/node:18-slim
```

Figure 76: Construction de l'image result-app

Une instance est lancée avec un lien vers PostgreSQL et un mappage des ports :

```
student@docker01:~/sample-application/example-voting-app$ docker run -d -p 5001:80 --link db:db result-app
130c7700878b8d8c241735c1b0eac3fcf7ed29246ac78465b35e99041e6808f
student@docker01:~/sample-application/example-voting-app$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
130c7700878b   result-app                          "/usr/bin/tini -- no..." 7 seconds ago Up 7 seconds   0.0.0.0:5001->80/tcp, [::]:5
001->80/tcp
8a35f91170ba   worker-app                          "dotnet Worker.dll"     2 hours ago   Up 2 hours
291dabeeaf5e   postgres:15-alpine                 "docker-entrypoint.s..." 2 hours ago   Up 2 hours   5432/tcp
99a460f9f93e   voting-app                          "unicorn app:app -b..." 2 hours ago   Up 2 hours   0.0.0.0:5000->80/tcp, [::]:5
000->80/tcp
f163705ad746   redis                               "docker-entrypoint.s..." 3 hours ago   Up 3 hours   6379/tcp
```

Figure 77: Exécution de worker-app avec connexion à PostgreSQL

Le port 5001 de l'hôte est mappé au port 80 du conteneur, car le port 5000 est déjà utilisé par l'application de vote.

Les conteneurs Redis, PostgreSQL, l'application de vote, le worker, et l'application de résultats fonctionnent correctement.

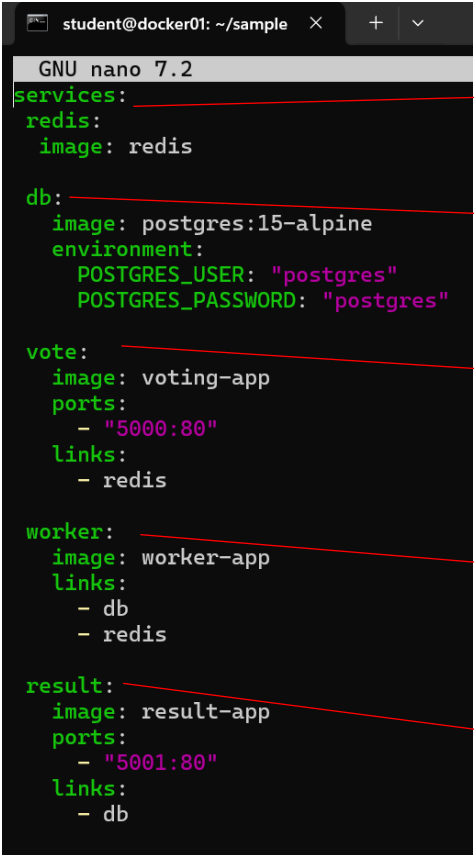
1. Utilisation de Docker Compose

Création et utilisation du fichier docker-compose.yml

Le fichier docker-compose.yml a été créé pour définir plusieurs services nécessaires à l'application, notamment :

```
student@docker01: ~/sample-application$ ls
docker-compose.yml  example-voting-app
```

Figure 78: Fichier docker-compose.yml



The screenshot shows the content of the `docker-compose.yml` file in a nano editor. Red arrows point from specific sections of the file to descriptive text on the right.

Service	Description
redis	Utilisé comme système de messagerie.
db	PostgreSQL : Base de données pour stocker les résultats.
vote	Voting App : Interface web pour voter. Le port 5000 de l'hôte est mappé au port 80 du conteneur. Lien : Connecté au service redis pour enregistrer les votes.
worker	Worker : Traitement des votes. Lien : Connecté à db pour écrire les données dans la base de données. Connecté à redis pour récupérer les données de vote.
result	Result App : Interface web pour afficher les résultats. Le port 5001 de l'hôte est mappé au port 80 du conteneur. Lien : Connecté à db pour lire les résultats et les

Figure 79: Contenu de docker-compose.yml

La configuration du fichier YAML inclut les ports exposés, les variables d'environnement (pour PostgreSQL), et les liens entre les services.

Commandes utilisées

La commande suivante lance tous les services définis dans le fichier YAML :

```
student@docker01:~/sample-application$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
student@docker01:~/sample-application$ docker-compose up -d
[+] Running 6/6
✓Network sample-application_default Created                                0.1s
✓Container sample-application-db-1 Started                               0.2s
✓Container sample-application-redis-1 Started                             0.2s
✓Container sample-application-result-1 Started                             0.1s
✓Container sample-application-worker-1 Started                             0.2s
✓Container sample-application-vote-1 Started                               0.2s
student@docker01:~/sample-application$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
a7c8195f090a   worker-app     "dotnet Worker.dll"      4 seconds ago  Up 4 seconds  5000->80/tcp  sample-application-worker-1
68540d4ff129   voting-app     "gunicorn app:app -b..." 4 seconds ago  Up 4 seconds  5000->80/tcp  sample-application-vote-1
a04569cfd833   result-app     "/usr/bin/tini -- no..." 4 seconds ago  Up 4 seconds  5001->80/tcp  sample-application-result-1
766edb932fca   redis         "docker-entrypoint.s..." 5 seconds ago  Up 4 seconds  6379/tcp     sample-application-redis-1
924a55eedfd9   postgres:15-alpine "docker-entrypoint.s..." 5 seconds ago  Up 4 seconds  5432/tcp     sample-application-db-1
student@docker01:~/sample-application$
```

Figure 80: Commande 'docker-compose up'

Tous les conteneurs sont démarrés simultanément, avec les réseaux et dépendances correctement configurés.

Une fois démarrés, les conteneurs sont accessibles sur les ports définis

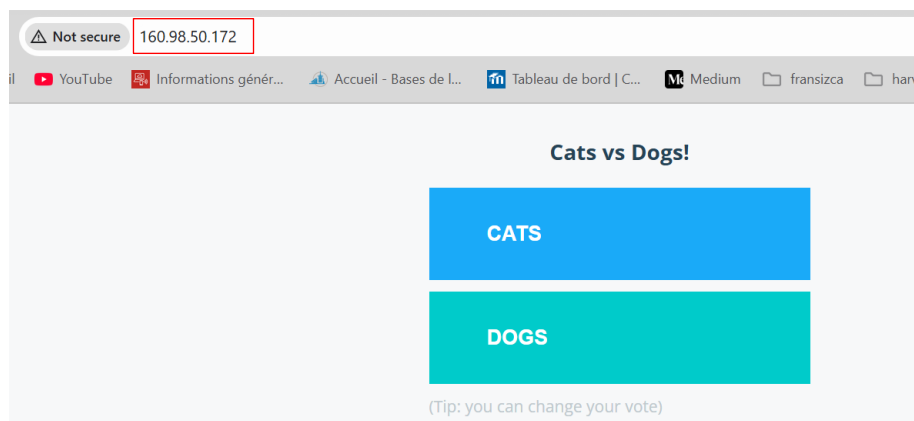


Figure 81: Accès à l'application voting-app après la commande compose

La commande suivante arrête et supprime tous les conteneurs ainsi que le réseau créé :

```
student@docker01:~/sample-application$ docker-compose down
[+] Running 6/6
✓Container sample-application-result-1 Removed                                0.3s
✓Container sample-application-vote-1 Removed                                0.5s
✓Container sample-application-worker-1 Removed                              0.3s
✓Container sample-application-db-1 Removed                                 0.3s
✓Container sample-application-redis-1 Removed                              0.2s
✓Network sample-application_default Removed                                0.2s
student@docker01:~/sample-application$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
student@docker01:~/sample-application$
```

Figure 82: Commande 'docker-compose down'

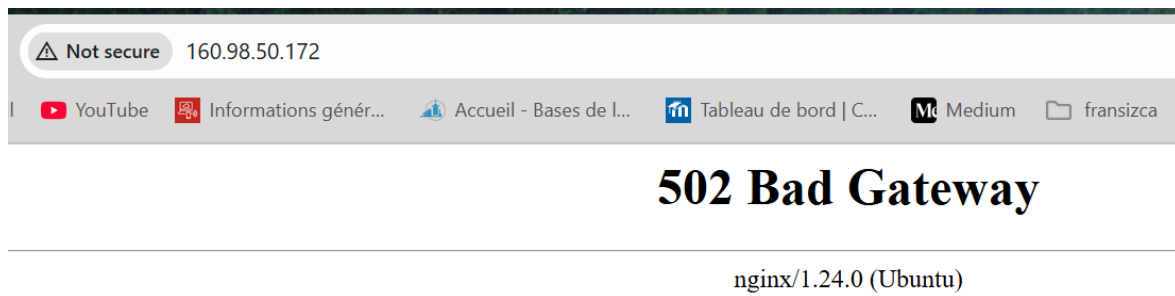


Figure 83: Après la commande 'docker-compose down'

Cela permet de libérer les ressources et de maintenir un environnement propre.

Docker Compose permet de gérer facilement des conteneurs interconnectés grâce à un seul fichier de configuration. Avec une simple commande, il est possible de démarrer ou d'arrêter tous les services liés, tout en assurant une gestion cohérente des dépendances et des réseaux.

IV. CONCLUSION

En conclusion, Docker est un outil très utile pour simplifier le développement et le déploiement des applications. Grâce à Docker, il est possible de créer, gérer et exécuter des applications rapidement et efficacement. Les exemples pratiques, comme Docker Compose, montrent comment Docker peut aider à gérer plusieurs conteneurs en même temps. Cela fait de Docker un choix important pour les développeurs et les entreprises modernes.

LISTE DES FIGURES

Figure 1 : Docker	3
Figure 2 : Conteneurs	4
Figure 3 : Fonctionnement de Docker	5
Figure 4 : Partager le noyau	5
Figure 5 : Machines virtuelles	6
Figure 6 : Conteneurs	7
Figure 7 : Image Docker	8
Figure 8: Plusieurs conteneurs	9
Figure 9: Fichier compose.yml	9
Figure 10: Docker compose	10
Figure 11: Architecture du Docker Swarm	10
Figure 12: Commande replicas	11
Figure 13: Commande cat /etc/*release*	12
Figure 14: Suppression d'ancien version de Docker	12
Figure 15: Commande d'installation	13
Figure 16: Commande version	13
Figure 17: Image CentOS	14
Figure 18: Commande 'docker run centos'	14
Figure 19: Commande 'docker run -it centos bash'	14
Figure 20: Commande 'cat /etc/*release*' for conteneur	14
Figure 21: Commande 'docker ps'	15
Figure 22: Commande sleep	15
Figure 23: Vérification de la commande sleep	15
Figure 24: Commande 'docker ps -a'	15
Figure 25: Commande 'docker stop'	16
Figure 26: Commande 'docker rm'	16
Figure 27: Utilisations de 'docker rm'	16
Figure 28: Commande 'docker image'	17
Figure 29: Commande 'docker rmi'	17
Figure 30: Vérification de l'utilisation des images Docker	17
Figure 31: Erreur suppression image	17
Figure 32: Supprimer le conteneur avant l'image	18
Figure 33: Liste des tags supportés	18
Figure 34: Affichage des informations de version Ubuntu	18
Figure 35: Commande avec la version	19
Figure 36: Commande -d	19
Figure 37: Commande -p	20
Figure 38: Contrôle de l'installation NGINX	20
Figure 39: Commande 'docker run' pour jenkins/jenkins	20
Figure 40: Contrôle les portes avec la commande 'docker ps'	20
Figure 41: Connexion Jenkins via IP	20
Figure 42: Dockerfile	21
Figure 43: Contenu du Dockerfile	21
Figure 44: Commande 'docker build'	22

Figure 45: Test de l'image localement	22
Figure 46: Création d'un tag pour l'image Docker	22
Figure 47: Commande 'docker push'	22
Figure 48: Image sur Docker Hub	22
Figure 49: Création et activation d'un environnement virtuel Python	23
Figure 50: Création app.py	23
Figure 51: Contenu de app.py	23
Figure 52: Création Dockerfile and app.py	24
Figure 53: Contenu de Dockerfile	24
Figure 54: Création d'une image (docker build)	24
Figure 55: Vérification de l'image créée avec docker build	24
Figure 56: Lancement du conteneur simple_webapp	25
Figure 57: Accès à l'application via le navigateur	25
Figure 58: Accès à l'application via le navigateur 2	25
Figure 59: Publication de l'image sur Docker Hub	25
Figure 60: Image simple_webapp sur Docker Hub	26
Figure 61: Architecture de l'application de vote	27
Figure 62: Clonage du projet depuis GitHub	27
Figure 63: Vérification du Dockerfile	28
Figure 64: Construction de l'image dans le répertoire contenant le Dockerfile	28
Figure 65: Vérification de l'image voting-app	28
Figure 66: Exécution de l'image de vote	29
Figure 67: Accès à l'application voting-app via le navigateur	29
Figure 68: Erreur après interaction avec l'application	29
Figure 69: Erreur de connexion à Redis	29
Figure 70: Exécution de l'image de redis	30
Figure 71: Exécution de voting-app avec connexion au conteneur Redis.	30
Figure 72: L'application de vote fonctionne	30
Figure 73: Exécution de l'image PostgreSQL	31
Figure 74: Construction de l'image worker-app	31
Figure 75: Exécution de worker-app avec connexion à Redis et PostgreSQL	31
Figure 76: Construction de l'image result-app	31
Figure 77 : : Exécution de worker-app avec connexion à PostgreSQL	31
Figure 78: Fichier docker-compose.yml	32
Figure 79: Contenu de docker-compose.yml	32
Figure 80: Commande 'docker-compose up'	33
Figure 81: Accès à l'application voting-app après la commande compose	33
Figure 82: Commande 'docker-compose down'	33
Figure 83: Après la commande 'docker-compose down'	34

V. SOURCES

Qu'est-ce que Docker ? :

- <https://tudip.com/blog-post/docker-and-docker-container/>

Comprendre le fonctionnement de Docker :

- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/home/module/1>

Qu'est-ce que cela veut dire, partager le noyau ? :

- <https://techalmirah.com/how-docker-shares-kernel/>
- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/home/module/1>

Comparaison entre conteneurs et machines virtuelles :

- <https://www.docker.com/resources/what-container/>

Images et Conteneurs Docker

- <https://circleci.com/blog/docker-image-vs-container/>
- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/home/module/1>

Pourquoi utiliser Docker ? :

- <https://www.geeksforgeeks.org/why-should-you-use-docker-7-major-reasons/>

Installation de Docker :

- <https://docs.docker.com/desktop/setup/install/linux/ubuntu/>
- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/lecture/4Pmla/demo-setup-and-install-docker>

Commandes Docker :

- https://docs.docker.com/get-started/docker_cheatsheet.pdf
- <https://docs.docker.com/reference/cli/docker/container/>
- <https://docs.docker.com/reference/cli/docker/image/>

Docker Run

- <https://www.youtube.com/watch?v=mgwo8fq-SkA>
- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/home/module/3>

Création et publication d'images Docker sur Docker Hub

- rabiasevinc/hello <https://www.youtube.com/watch?v=AdiuNwYEjDQ&t=160s>
- rabiasevinc/simple_webapp <https://github.com/mmumshad/simple-webapp-flask?tab=readme-ov-file>

Docker Compose

- <https://github.com/docker-samples/example-voting-app>
- <https://docs.docker.com/compose/intro/compose-application-model/>
- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/lecture/ncOJm/docker-compose>

Docker Swarm:

- <https://www.coursera.org/learn/docker-for-the-absolute-beginner/lecture/OJY6b/docker-swarm>
- <https://blog.alphorm.com/utiliser-docker-swarm-pour-lorchestration>

A. Images

<https://www.coursera.org/learn/docker-for-the-absolute-beginner/home/module/1>

<https://www.docker.com/resources/what-container/>

<https://medium.com/@mrdevsecops/docker-objects-e561f0ce3365>

<https://docs.olderimes.me/docker/engine/swarm/how-swarm-mode-works/services/index.html>