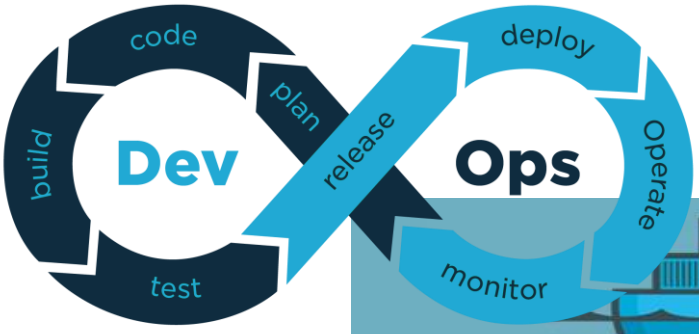
DF FORMATION

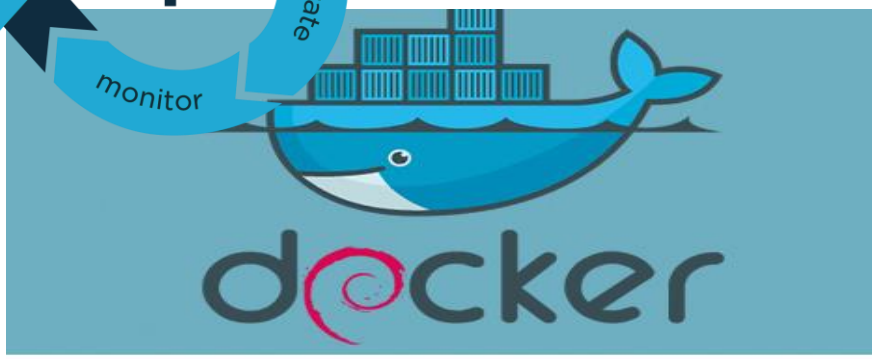
MINISTÈRE DU TRAVAIL, DE L'EMPLOI ET DE L'INSERTION

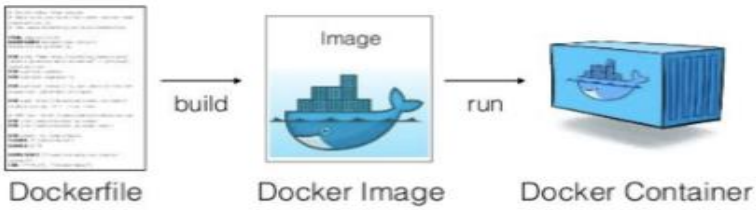
YYYOURS

EVALUATION EN COURS DE FORMATION

PARCOURS ADMINISTRATEUR SYSTEME DEVOPS







Gérer des containers

Evaluation : **ECF7**

Version : du 10/11/2025

Auteur : FT

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF7	Titre	**Gérer des containers**	Du	10/11/2025

Sommaire

Table des matières

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF7	Titre	**Gérer des containers**	Du	10/11/2025

1. PRESENTATION

1.1. DESCRIPTION DE LA COMPETENCE – PROCESSUS DE MISE EN ŒUVRE

- En tenant compte de l'architecture applicative, créer des containers pour préparer le déploiement de l'application et les connecter au réseau et au stockage distant.
- Afin de mettre en ligne une nouvelle version d'application, utiliser un outil de type Docker pour gérer la mise à jour des containers concernés

1.2. CONTEXTE(S) PROFESSIONNEL(S) DE MISE EN ŒUVRE

Cette compétence s'exerce lors de la préparation de la mise en production d'une nouvelle application ou de sa mise à jour.

1.3. CRITERES DE PERFORMANCE

- Les containers sont opérationnels
- Les containers sont connectés au réseau
- Les containers sont connectés au stockage distant
- Les containers sont mis à jour

1.4. SAVOIR-FAIRE TECHNIQUES, ORGANISATIONNELS, RELATIONNELS

- Utiliser des images pour créer des containers
- Définir pour chaque container les ressources nécessaires (CPU, mémoire, l'espace disque)
- Connecter le container au système hôte (réseau et stockage)
- Automatiser la création des containers avec un outil de type docker
- Utiliser les containers pour gérer les mises à jour applicatives
- Echanger avec les développeurs
- Consulter de la documentation technique rédigée en anglais
- Effectuer une veille technologique
- Connaissance de la notion de containers
- Connaissance de l'architecture applicative de microservices

1.5. CAHIER DES CHARGES

Docker, produit de conteneurisation crée par dotcloud et rendu open source en 2013 rend la virtualisation plus scalable, robuste et rapide avec notamment le docker engine qui vient se greffer sur le host OS.

Dans cet ECF vous serez amené à analyser le fonctionnement d'un outil de conteneurisation qu'est docker et mettre en place une architecture de type micro service.

a. Prise en main de Docker

1 - Installer Docker et tester l'installation avec un docker -version

2 - Lister les conteneurs actifs de votre machine.

3 - Lancer un conteneur toto avec l'image nginx:latest avec le paramètre -p 8080:80.

Lancer l'url 127.0.0.1/80 dans votre navigateur. Lister les conteneurs actifs et inactifs de votre machine.

4 - Supprimez l'image toto et relancez-la avec les paramètre -tid -p 8080:80 .

Lister les conteneurs actifs de votre machine.

5 - Modifier le fichier html lancé par le serveur web(/usr/share/nginx/html/index.html)

Vérifier si la modification sur l'url 127.0.0.1 a bien eu lieu.

6 - Entrer dans l'image avec la commande docker exec -ti toto increment.sh

7 - Supprimer l'image.

b. Construire une architecture de type micro-service

1 - Créer une structure de fichier de type micro-service :

```
myapp/
├── serveurweb/
│   └── Dockerfile
├── worker1/
│   ├── affichage.py
│   ├── Dockerfile
│   └── increment.sh
└── worker2/
    ├── affichage.py
    ├── Dockerfile
    └── increment.sh
```

2 – Définissez dans le service serveurweb, un Dockerfile créant un conteneur Nginx avec un volume de montage persistant, le dockerfile devrait ressembler à :

```
FROM    nginx base latest /

MAINTAINER you /

RUN aptget update && aptget install -y nginx /

VOLUME /var/www/html /

ENTRYPOINT [ "nginx", "-g", "daemon o_;"]
```

3 - Implémenter un fichier shell qui incrémente une variable de 1 chaque seconde et qui sauvegarde cette variable dans les fichiers /var/www/html/wrk1.txt et /var/www/html/wrk2.txt puis lance le fichier python affichage.py.

Ce mini script que vous implémenterez lira ce fichier au format text pour modifier le fichier html du serveur web.

4 - Définissez dans les services worker1 et worker2, un Dockerfile créant un conteneur Ubuntu avec une gestion des fichiers de code dans le cadre d'un binding avec le volume de montage persistant du serveur web, le dockerfile devrait ressembler à :

```
FROM ubuntu latest /

MAINTAINER you /

COPY increment.sh /

&& affichage.py /

RUN chmod 755 /increment.sh /

ENTRYPOINT ["/ increment.sh"]
```

5 - Lancer les services un par un :

docker run -tid -name toto serveurweb

docker run -tid -name worker1 -volumes-from toto worker1

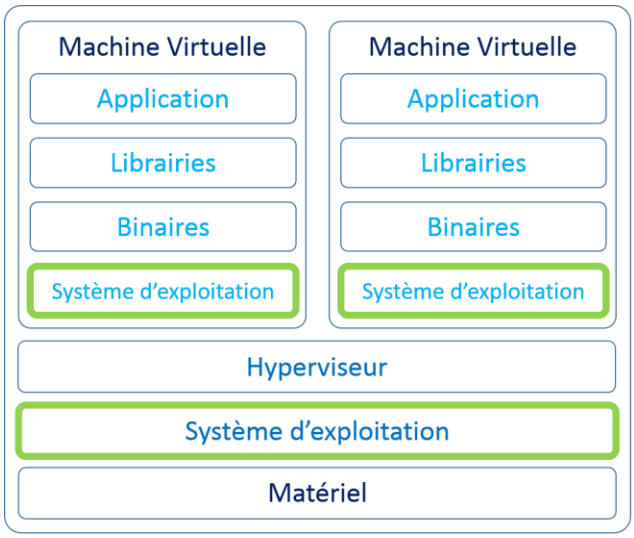
docker run -tid -name worker2 -volumes-from toto worker2

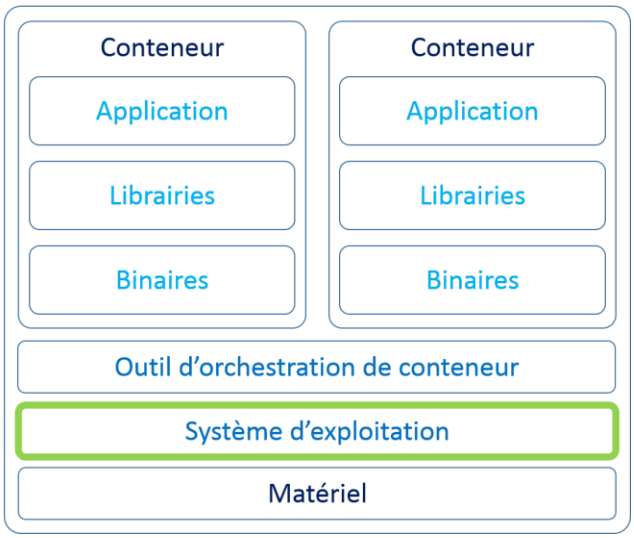
6 - Exportez le projet docker et en joignant une capture d'écran du rendu web.

2. QUESTIONS ECF7

2.1. DONNER LES DIFFERENCES PRINCIPALES ENTRE UN SYSTEME DE CONTENEURISATION ET UNE MACHINE VIRTUELLE.

- Les conteneurs et les machines virtuelles diffèrent considérablement en termes d'autonomie du système d'exploitation.
- Les **conteneurs** adhèrent à la virtualisation du système d'exploitation, ce qui signifie qu'ils **exploitent les ressources du système d'exploitation hôte.**
- En revanche, chaque instance d'une **VM constitue un système d'exploitation invité à part entière**, une copie matérielle virtuelle pour l'exécuter, une application, ses bibliothèques et dépendances. Elle peut exécuter différents systèmes d'exploitation sur le même serveur physique.
- Comme les **conteneurs** virtualisent le système d'exploitation, ils **ne contiennent que l'application avec ses bibliothèques et dépendances.**





2.2. Est-il possible de creer des ensembles de conteneurs dans des sous reseaux separes sur un meme serveur physique ?

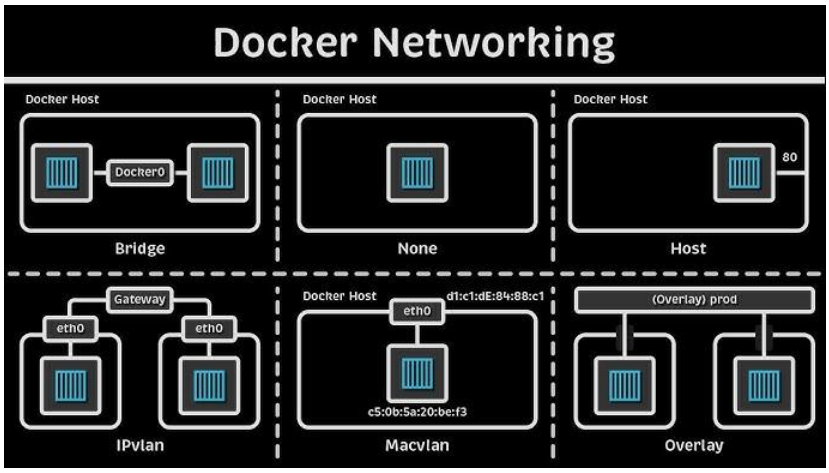
Si oui, expliquez comment.

Oui, il est tout à fait possible de créer des ensembles de conteneurs dans des sous-réseaux séparés sur un même serveur physique.

C'est une pratique courante et recommandée pour l'organisation, la sécurité et la performance des applications conteneurisées et cela repose sur la gestion réseau au niveau de l'hôte (serveur physique) qui exécute les conteneurs.

Les principaux types de réseaux Docker :

- **Bridge** : réseau par défaut qui permet aux conteneurs de communiquer entre eux sur un même hôte.
- **Host** : réseau qui permet à un conteneur de partager directement le réseau de l'hôte, en utilisant l'adresse IP de la machine hôte.
- **Overlay** : réseau qui connecte plusieurs conteneurs sur différents hôtes, idéal pour les clusters Docker.
- **None** : réseau qui désactive tout réseau pour un conteneur, isolant totalement ce dernier.



- **Macvlan** : réseau qui attribue à chaque conteneur une adresse MAC unique.
- **Ipvlan** : réseau qui permet aux conteneurs de partager l'interface réseau de l'hôte tout en attribuant à chaque conteneur une adresse IP distincte.

3. Prise en main de Docker

3.1. Installation de Docker

L'installation de Docker va être exécutée sur un serveur Debian 11 bullseye.

a. Configuration du référentiel apt de Docker.

 # Ajout Docker's official GPG key:

```
sudo apt update

sudo apt install –y ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc
```

 # Ajout du repository aux sources Apt:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
```

b. Installation Docker packages.

```
sudo apt install –y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

c. Vérification de l'installation

docker --version	`fabrice@debian11:~$ docker --version` `Docker version 28.1.1, build 4eba377`

sudo docker run hello-world
 	
Cette commande télécharge une image de test et l'exécute dans un conteneur.
 	
Une fois le conteneur exécuté, il affiche un message de confirmation et se ferme.
	`fabrice@debian11:~$ sudo docker ps -a`
	`CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES`
	`a9b23f0b64e0 hello-world "/hello" 24 seconds ago Exited`
	
	La commande **docker ps** permet de lister les conteneurs actifs et avec l'option **-a** , tous les conteneurs

3.2. LANCER UN CONTENEUR NOMME TOTO AVEC NGINX

- Exécution de la commande suivante : docker run -d --name toto -p 8080:80 nginx:latest

```
fabrice@debian11:~$ sudo docker run -d --name toto -p 8080:80 nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
61320b01ae5e: Pull complete
670a101d432b: Pull complete
405bd2df85b6: Pull complete
cc80efff8457: Pull complete
2b9310b2ee4b: Pull complete
6c4aa022e8e1: Pull complete
abddc69cb49d: Pull complete
Digest: sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Status: Downloaded newer image for nginx:latest
c6e094d74116225818166b2bd0f3acc761dfb99d9a3cefacca80acb784375cd5
```

- Vérification dans le navigateur à l'adresse : http://127.0.0.1:8080

- Conteneurs actifs et inactifs de la machine : docker ps –a

```
fabrice@debian11:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
               PORTS                                       NAMES
c6e094d74116   nginx:latest   "/docker-entrypoint.…"   3 minutes ago    Up 3 min
utes           0.0.0.0:8080->80/tcp, [::]:8080->80/tcp     toto
a9b23f0b64e0   hello-world    "/hello"                 18 minutes ago   Exited (
0) 18 minutes ago                                          trusting_wiles
```

3.3. SUPPRESSION ET RELANCE DE L'IMAGE TOTO AVEC -TID

- Suppression du conteneur toto : docker rm -f toto

```
fabrice@debian11:~$ sudo docker rm -f toto
toto
```

- Relance du conteneur toto (avec –tid) : docker run -tid --name toto -p 8080:80 nginx:latest

- Liste des conteneurs actifs : docker ps

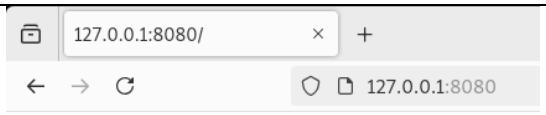
```
fabrice@debian11:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
ORTS                                       NAMES
a31fe903391f   nginx:latest   "/docker-entrypoint.…"   7 seconds ago    Up 6 sec
.0.0.0:8080->80/tcp,_[::]:8080->80/tcp   toto
```

3.4. MODIFICATION DE LA PAGE HTML DU CONTENEUR NGINX

- Entrer dans le conteneur : docker exec -ti toto bash
- Modification du fichier HTML : echo "Bonjour depuis Docker" > /usr/share/nginx/html/index.html

```
fabrice@debian11:~$ sudo docker exec -it toto bash
root@a31fe903391f:/# echo "Bonjour depuis Docker" > /usr/share/nginx/html/index.html
```

- Vérification dans le navigateur à l'adresse : http://127.0.0.1:8080



3.5. ENTRER DANS LE CONTENEUR ET LANCER INCREMENT.SH

- Création du script d'incrémentation (dans une Boucle infinie avec enregistrement dans un fichier texte)

```bash
#!/bin/bash

# Init
file="compteur.txt"

# Si le fichier n'existe pas, on initialise la variable à 0
if [ ! -f "$file" ]; then
    echo 0 > "$file"
fi

# Lecture de la dernière valeur
compteur=$(cat "$file")

# Boucle infinie
while true; do
    # Incrémenter la variable
    compteur=$((compteur + 1))

    # Sauvegarder dans le fichier
    echo "$compteur" > "$file"

    # Afficher dans le terminal
    echo "Compteur: $compteur"

    # Attendre 1 seconde
    sleep 1
done
```

- Copie du script dans le conteneur : docker cp increment.sh toto:/increment.sh

```
fabrice@debian11:~$ sudo docker cp increment.sh toto:/increment.sh
Successfully copied 1.54kB to toto:/increment.sh
```

- Exécution du script dans le conteneur : docker exec -ti toto bash /increment.sh

```
fabrice@debian11:~$ sudo docker exec -it toto bash /increment.sh
Compteur: 1
Compteur: 2
Compteur: 3
Compteur: 4
Compteur: 5
Compteur: 6
Compteur: 7
Compteur: 8
```

3.6. SUPPRESSION DE L'IMAGE

- Suppression du conteneur : docker rm -f toto

```
fabrice@debian11:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                       NAMES
a31fe903391f   nginx:latest   "/docker-entrypoint.…"   46 minutes ago   Up 46 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   toto
fabrice@debian11:~$ sudo docker rm -f toto
toto
```

- Suppression de l'image : docker rmi nginx:latest

```
fabrice@debian11:~$ sudo docker rmi nginx:latest
Untagged: nginx:latest
Untagged: nginx@sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Deleted: sha256:be69f2940aaf64fdf50c9c99420cbd57e10ee655ec7204df1c407e9af63d0cc1
Deleted: sha256:d4b3df3b01538a659a8990a121e2f90340dec055232275eb6e121f3af230e3c4
Deleted: sha256:8b8ba0c31936a63862b7bb3f9edd4e2721da3ccf99dc0e6dcaf289bbd212ba6f
Deleted: sha256:fcef123e6b584f68369e1c45f4e582be8391b93abd1e5be6ef674ce3459ac7fd
Deleted: sha256:ee697e6373f3f1a82f1a55fc29ef5ebc2c05158dcbfbeffcaf223406f429be8b
Deleted: sha256:d6189b151727a846370cf1355a8b5276f6c2aa41f378cc2945205a0eb03ea861
Deleted: sha256:3c1060a9ec6601e87cbcb04366f323b6383b7f605153e7df398bc1f843e36141
Deleted: sha256:ace34d1d784c01e3f9d156687089e8f58f786e23ccd097bdbbf337d6d28b3783
```

4. Architecture de type Micro-Service

4.1. Creation structure de fichier de type micro-service

```
myapp/
├── serveurweb/
│   ├── default.conf
│   └── Dockerfile
├── worker1/
│   ├── affichage.py
│   ├── Dockerfile
│   └── increment.sh
└── worker2/
    ├── affichage.py
    ├── Dockerfile
    └── increment.sh
```

```
fabrice@debian11:~$ tree myapp/
myapp/
├── serveurweb
│   ├── default.conf
│   └── Dockerfile
├── worker1
│   ├── affichage.py
│   ├── Dockerfile
│   └── increment.sh
└── worker2
    ├── affichage.py
    ├── Dockerfile
    └── increment.sh
```

4.2. Dockerfile NGINX

Définir dans le service **serveurweb**, un Dockerfile créant un **conteneur Nginx** avec un volume de montage persistant.

Dockerfile

Montage persistant vers **/var/www/html**

Remplacement du **default.conf** pour pointer vers ce montage

```
FROM nginx:latest
LABEL maintainer="FT"

RUN apt update && apt install -y nginx

RUN rm /etc/nginx/conf.d/default.conf
COPY default.conf /etc/nginx/conf.d/

VOLUME ["/var/www/html"]

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Configuration

Nginx default.conf

```
server {
    listen 80;
    server_name localhost;

    location / {
        root /var/www/html;
        index index.html;
    }
}
```

4.3. SCRIPT INCREMENT.SH POUR WORKER1 ET WORKER2

Implémenter un fichier shell qui **incrémente une variable de 1 chaque seconde** et qui sauvegarde cette variable dans les fichiers **/var/www/html/wrk1.txt** et **/var/www/html/wrk2.txt** puis lance le fichier python **affichage.py**.

a. Worker1

```bash
#!/bin/bash

# fichier : increment.sh
COUNT=0                          # Cette variable va servir de compteur incrémenté à chaque seconde
FILE="/var/www/html/wrk1.txt"    # Destination où on écrit la valeur du compteur

while true; do                   # Boucle infinie qui va tourner sans arrêt (jusqu'à interruption)
  echo $COUNT > $FILE            # Écrit la valeur actuelle du compteur dans le fichier cible
  COUNT=$((COUNT + 1))           # Incrémente la variable COUNT de 1
  sleep 1                        # Pause de 1 seconde avant de répéter la boucle
  python3 /affichage.py          # Mise à jour du fichier HTML Nginx (index.html) avec les valeurs de wrk1.txt
done
```

b. Worker2

```bash
#!/bin/bash

# fichier : increment.sh
COUNT=0                          # Cette variable va servir de compteur incrémenté à chaque seconde
FILE="/var/www/html/wrk2.txt"    # Destination où on écrit la valeur du compteur

while true; do                   # Boucle infinie qui va tourner sans arrêt (jusqu'à interruption)
  echo $COUNT > $FILE            # Écrit la valeur actuelle du compteur dans le fichier cible
  COUNT=$((COUNT + 1))           # Incrémente la variable COUNT de 1
  sleep 1                        # Pause de 1 seconde avant de répéter la boucle
  python3 /affichage.py          # Mise à jour du fichier HTML Nginx (index.html) avec les valeurs de wrk2.txt
done
```

4.4. SCRIPT AFFICHAGE.PY

Ce script **lit le fichier au format texte** pour **modifier le fichier html du serveur** web.

a. Worker1

```python
wrk_path = "/var/www/html/wrk1.txt"          # fichier contenant le compteur de worker1
output_path = "/var/www/html/index.html"     # fichier HTML généré que Nginx

def read_count(path):
    try:
        with open(path, "r") as f:           # ouvre le fichier en lecture
            return f.read().strip()          # lit tout le contenu du fichier et enlève les espaces ou retours à la ligne
    except:                                  # si le fichier n'existe pas ou si une erreur se produit, on retourne "N/A"
        return "N/A"

count = read_count(wrk_path)                 # Lecture du compteurs du worker, et on le stocke dans la variable count

with open(output_path, "w") as f:            # Ouverture du fichier index.html en écriture ("w")
    f.write(f"""
    <html>
    <head><title>Worker 1 Status</title></head>
    <body>
        <h1>Worker 1 Count: {count}</h1>
    </body>
    </html>
    """)
```

b. Worker2

```python
wrk_path = "/var/www/html/wrk2.txt"          # fichier contenant le compteur de worker2
output_path = "/var/www/html/index.html"     # fichier HTML généré que Nginx

def read_count(path):
    try:
        with open(path, "r") as f:           # ouvre le fichier en lecture
            return f.read().strip()          # lit tout le contenu du fichier et enlève les espaces ou retours à la ligne
    except:                                  # si le fichier n'existe pas ou si une erreur se produit, on retourne "N/A"
        return "N/A"

count = read_count(wrk_path)                 # Lecture du compteur du worker, et on le stocke dans la variable count

with open(output_path, "w") as f:            # Ouverture du fichier index.html en écriture ("w")
    f.write(f"""
    <html>
    <head><title>Worker 2 Status</title></head>
    <body>
        <h1>Worker 2 Count: {count}</h1>
    </body>
    </html>
    """)
```

4.5. Dockerfile Worker1 et Worker2

Définir dans les services worker1 et worker2, un Dockerfile créant un **conteneur Ubuntu** avec une gestion des fichiers de code dans le cadre d'un binding avec le volume de montage persistant du serveur web.

- Pull de l'image Ubuntu
- Installation de python3
- Copie des scripts dans le conteneur
- Lancement au démarrage du script d'incrémentation du compteur

```
FROM ubuntu:latest
LABEL maintainer="FT"

RUN apt update && apt install -y python3

COPY increment.sh /increment.sh
COPY affichage.py /affichage.py

RUN chmod 755 /increment.sh

ENTRYPOINT ["/increment.sh"]
```

4.6. Lancer les services (en ANNEXE)

- **Build les images**

```
docker build -t serveurweb .
docker build -t worker1 .
docker build -t worker2 .
```

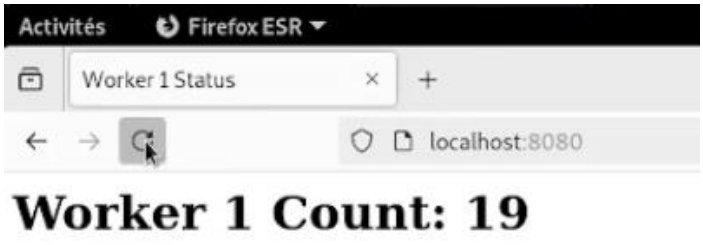
- **Lancer les conteneurs**

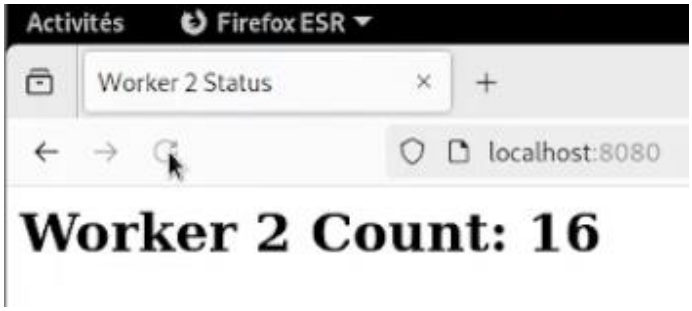
```
docker run -tid --name toto -p 8080:80 serveurweb
docker run -tid --name worker1 --volumes-from toto worker1
docker run -tid --name worker2 --volumes-from toto worker2
```

```
fabrice@debian11:~$ sudo docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED              STATUS              PORTS                                   NAMES
4ad834f24b8c   worker2      "/increment.sh"          36 seconds ago       Up 35 seconds                                               worker2
1d3160624312   worker1      "/increment.sh"          About a minute ago   Up About a minute                                           worker1
53d876695bfc   serveurweb   "nginx -g 'daemon of…"   2 minutes ago        Up 2 minutes        0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   toto
```

5. RESULTAT

Lancement du navigateur à l'adresse http://localhost:8080





- Commit vers https://github.com/fabrice-git-hub/ec7.git
- Vidéo du résultat ici :

https://github.com/fabrice-git-hub/ec7/raw/refs/heads/main/ECF7_RenduWeb-Compteur.mp4

6. Annexe

Lancer les services (Build des images)

a. Serveur NGINX

```
fabrice@debian11:~/myapp/serveurweb$ sudo docker build -t serveurweb .
[sudo] Mot de passe de fabrice :
[+] Building 3.0s (9/9) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                          0.1s
 => => transferring dockerfile: 273B                                          0.0s
 => [internal] load metadata for docker.io/library/nginx:latest               0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/4] FROM docker.io/library/nginx:latest                                 0.0s
 => [internal] load build context                                             0.2s
 => => transferring context: 179B                                             0.0s
 => CACHED [2/4] RUN apt update && apt install -y nginx                       0.0s
 => [3/4] RUN rm /etc/nginx/conf.d/default.conf                               1.8s
 => [4/4] COPY default.conf /etc/nginx/conf.d/                                0.3s
 => exporting to image                                                        0.2s
 => => exporting layers                                                       0.1s
 => => writing image sha256:236819072d4c18f73592aeb75d4bc77ff393ce3e187bf     0.0s
 => => naming to docker.io/library/serveurweb                                 0.0s
```

b. Worker1

```
fabrice@debian11:~/myapp/worker1$ sudo docker build -t worker1 .
[+] Building 2.0s (10/10) FINISHED                                  docker:default
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 257B                                          0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest              0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/5] FROM docker.io/library/ubuntu:latest                                0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 1.01kB                                           0.0s
 => CACHED [2/5] RUN apt update && apt install -y python3                     0.0s
 => CACHED [3/5] COPY increment.sh /increment.sh                              0.0s
 => [4/5] COPY affichage.py /affichage.py                                     0.1s
 => [5/5] RUN chmod 755 /increment.sh                                         1.3s
 => exporting to image                                                        0.2s
 => => exporting layers                                                       0.1s
 => => writing image sha256:5f460f1981557c53bfee7569604abe2270cae6115bf3f     0.0s
 => => naming to docker.io/library/worker1                                    0.0s
```

c. Worker2

```
fabrice@debian11:~/myapp/worker2$ sudo docker build -t worker2 .
[+] Building 1.6s (10/10) FINISHED                                  docker:default
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 257B                                          0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest              0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 1.01kB                                           0.0s
 => [1/5] FROM docker.io/library/ubuntu:latest                                0.0s
 => CACHED [2/5] RUN apt update && apt install -y python3                     0.0s
 => CACHED [3/5] COPY increment.sh /increment.sh                              0.0s
 => [4/5] COPY affichage.py /affichage.py                                     0.1s
 => [5/5] RUN chmod 755 /increment.sh                                         0.8s
 => exporting to image                                                        0.2s
 => => exporting layers                                                       0.2s
 => => writing image sha256:2ba3fea8c3c1aeb11a6652b7dfbe3c95719fc9a09bc47     0.0s
 => => naming to docker.io/library/worker2                                    0.0s
```