Docker compos

Introduction:

Pour suivre cet article, vous aurez besoin de :

- Accès à une machine locale Ubuntu 22.04
- Docker installé sur votre serveur ou votre machine locale.

Les étapes:

▼ Étape 1 - Installation de Docker Compose

- Pour vous assurer d'obtenir la version stable la plus récente de Docker Compose, tout d'abord, confirmez la dernière version disponible sur leur page de versions . Au moment d'écrire ces lignes, la version stable la plus récente est `2.17.3`.

Utilisez la commande suivante pour télécharger :

```
mkdir -p ~/.docker/cli-plugins/
```

```
curl -SL https://github.com/docker/compose/releases/download/v2.17.3/docker-compos
e-linux-x86_64 -o ~/.docker/cli-plugins/docker-compose
```

- Ensuite, définissez les autorisations correctes pour que la `docker compose` commande soit exécutable :

```
chmod +x ~/.docker/cli-plugins/docker-compose
```

- Pour vérifier que l'installation a réussi, vous pouvez exécuter :

```
docker compose version
```

Vous verrez une sortie semblable à celle-ci :

```
hilali@hilali-VirtualBox:~$ docker compose version
Docker Compose version v2.17.3
```

> 💡 Docker Compose est maintenant installé avec succès sur votre système. Dans la section suivante, vous verrez comment configurer un `docker-compose.yml` fichier et obtenir un environnement conteneurisé opérationnel avec cet outil.

▼ Étape 2: Test

▼ Test 1:

▼ Étape 1 - Configuration d'un `docker-compose.yml`

- Commencez par créer un nouveau répertoire dans votre dossier personnel, puis déplacez-vous dedans :

```
mkdir ~/hello
cd ~/hello
```

- Créez ensuite le `docker-compose.yml` fichier :

```
nano docker-compose.yml
```

Insérez le contenu suivant dans votre `docker-compose.yml` fichier :

```
version: '3'

services:
  app:
    image: node:14-alpine
    command: sh -c "echo 'Hello, World!'"
```

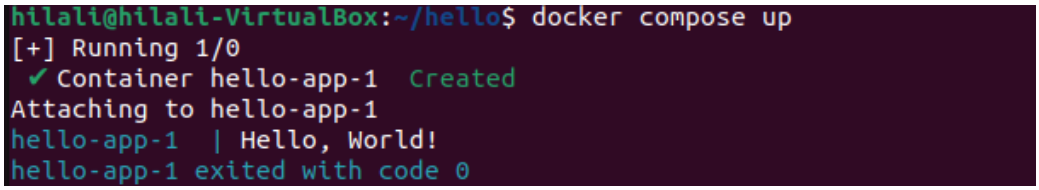
- Dans ce fichier, nous avons défini un service `app` qui utilise l'image Docker officielle de Node.js version 14 basée sur Alpine. Le service exécute ensuite une commande shell qui affiche "Hello, World!".

▼ Étape 2- Exécution de Docker Compose

- Pour lancer ce conteneur, il suffit d'exécuter la commande `docker-compose up` à partir du répertoire contenant le fichier `docker-compose.yam`.
  Docker Compose va alors télécharger l'image Docker de Node.js 14 (si elle n'est pas déjà présente sur le système) et lancer le conteneur en exécutant la

commande `sh -c "echo 'Hello, World!'"`. Le message "Hello, World!" sera alors affiché dans la sortie de la console.

```
docker compose up
```



💡 **Remarque :** Si vous avez utilisé un nom différent pour votre fichier de configuration Docker Compose, vous devez spécifier le nom complet du fichier à l'aide de l'option `-f` ou `--file` lors de l'exécution de la commande `docker-compose`. Par exemple, si vous avez nommé votre fichier de configuration `my-app.yml`, vous pouvez exécuter la commande suivante pour lancer les conteneurs :

```
docker-compose -f ./my-app.yml up
```

Cela permettra à Docker Compose de trouver votre fichier de configuration et de l'utiliser pour lancer les conteneurs. Si vous ne spécifiez pas le nom complet du fichier de configuration, Docker Compose recherchera un fichier avec le nom par défaut `docker-compose.yml` ou `docker-compose.yaml` dans le répertoire courant et générera une erreur s'il ne peut pas le trouver.

Assurez-vous que le nom du fichier que vous spécifiez avec l'option `-f` ou `--file` correspond exactement au nom du fichier de configuration que vous avez créé, y compris l'extension `.yml` ou `.yaml` si vous l'avez utilisée.

▼ Test 2:

▼ Étape 1 - Configuration d'un `docker-compose.yml`

Pour montrer comment configurer un `docker-compose.yml` fichier et travailler avec Docker Compose, vous allez créer un environnement de serveur Web à

l'aide de l' image Nginx officielle de Docker Hub , le registre public de Docker. Cet environnement conteneurisé servira un seul fichier HTML statique.

- Commencez par créer un nouveau répertoire dans votre dossier personnel, puis déplacez-vous dedans :

```
mkdir ~/compose-demo
cd ~/compose-demo
```

- Dans ce répertoire, configurez un dossier d'application qui servira de racine de document pour votre environnement Nginx :

```
mkdir app1
```

- À l'aide de votre éditeur de texte préféré, créez un nouveau `index.html` fichier dans le `app` dossier :

```
nano app1/index.html
```

Placez le contenu suivant dans ce fichier :

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Docker Compose Demo</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/kognise/wate
r.css@latest/dist/dark.min.css">
</head>
<body>
    <h1>Web 1</h1>
    <p>This is a Docker Compose Demo Page 1.</p>
</body>
</html>
```

Enregistrez et fermez le fichier lorsque vous avez terminé. Si vous utilisez `nano`, vous pouvez le faire en tapant `CTRL+X`, puis `Y` et `ENTER` pour confirmer.

- Créez ensuite le `docker-compose.yml` fichier :

```
nano docker-compose.yml
```

Insérez le contenu suivant dans votre `docker-compose.yml` fichier :

```
version: '3.7'
services:
  web:
    image: nginx:alpine
    ports:
      - "8000:80"
    volumes:
      - ./app1:/usr/share/nginx/html
```

1. Le `docker-compose.yml` fichier commence généralement par la `version` définition. Cela indiquera à Docker Compose la version de configuration que vous utilisez.
2. Vous avez ensuite le `services` bloc, où vous configurez les services qui font partie de cet environnement. Dans votre cas, vous avez un seul service appelé `web`. Ce service utilise l' `nginx:alpine` image et met en place une redirection de port avec la `ports` directive. Toutes les requêtes sur le port `8000` de la machine **hôte** (le système à partir duquel vous exécutez Docker Compose) seront redirigées vers le `web` conteneur sur le port `80`, où Nginx sera exécuté.
3. La `volumes` directive créera un volume partagé entre la machine hôte et le conteneur. Cela partagera le `app1` dossier local avec le conteneur, et le volume sera situé à `/usr/share/nginx/html` l'intérieur du conteneur, qui écrasera alors la racine du document par défaut pour Nginx.

- Enregistrez et fermez le fichier.

> 💡 Vous avez configuré une page de démonstration et un `docker-compose.yml` fichier pour créer un environnement de serveur Web conteneurisé qui le servira. À l'étape suivante, vous allez créer cet environnement avec Docker Compose.

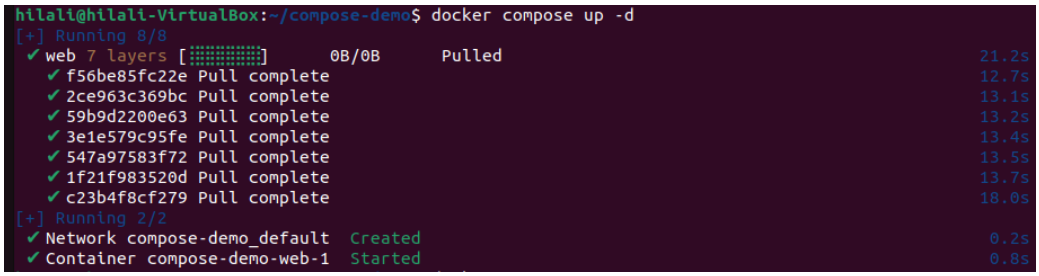
▼ Étape 2- Exécution de Docker Compose

- Avec le `docker-compose.yml` fichier en place, vous pouvez maintenant exécuter Docker Compose pour mettre votre environnement en place. La

commande suivante téléchargera les images Docker nécessaires, créera un conteneur pour le `web` service et exécutera l'environnement conteneurisé en arrière-plan (-d):

```
docker compose up -d
```

Docker Compose recherchera d'abord l'image définie sur votre système local, et s'il ne trouve pas l'image, il la téléchargera à partir de Docker Hub. Vous verrez une sortie comme celle-ci :

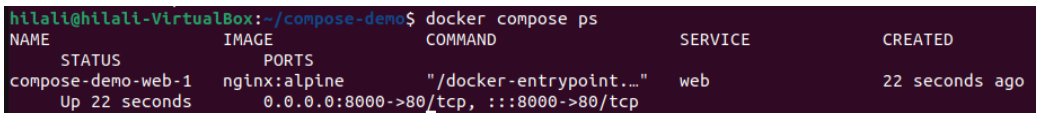


> 💡 **Remarque :** Si vous rencontrez une erreur d'autorisation concernant le socket Docker, cela signifie que vous avez ignoré l'étape 2 d'installation de Docker sur votre machine. Revenir en arrière et terminer cette étape permettra aux autorisations d'exécuter des commandes docker sans `sudo`.

Votre environnement est maintenant opérationnel en arrière-plan. Pour vérifier que le conteneur est actif, vous pouvez exécuter :

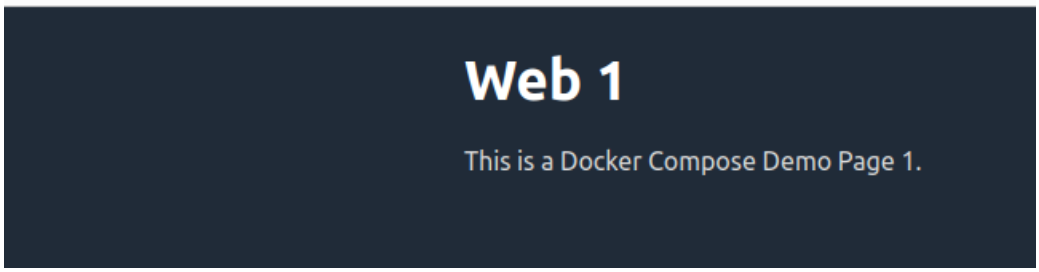
```
docker compose ps
```

Cette commande vous montrera des informations sur les conteneurs en cours d'exécution et leur état, ainsi que sur les redirections de port actuellement en place :



Vous pouvez maintenant accéder à l'application de démonstration en pointant votre navigateur soit `localhost:8000` si vous exécutez cette démo sur votre machine locale, soit `your_server_domain_or_IP:8000` si vous exécutez cette démo sur un serveur distant.

Vous verrez une page comme celle-ci :



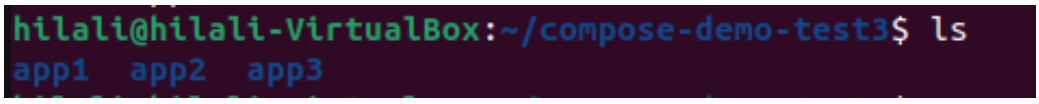
> 💡 Le volume partagé que vous avez configuré dans le fichier permet de synchroniser `docker-compose.yml` vos fichiers de dossier avec la racine du document du conteneur. `app` Si vous apportez des modifications au `index.html` fichier,elles seront automatiquement récupérées par le conteneur et donc épercutées sur votre navigateur lorsque vous rechargerez la page.Dans l'étape suivante, vous verrez comment gérer votre environnement conteneurisé avec les commandes Docker Compose.

▼ Test 3:

▼ Étape 1 - Configuration d'un `docker-compose.yml`

- dans le dossier `compose-demo` ajouter 2 dossier app (app2 et app3)

```
mkdir app2
mkdir app3
```



- configurer le fichier HTML sur chaque dossier app:

▼ dossier app2

- À l'aide de votre éditeur de texte préféré, créez un nouveau `index.html` fichier dans chaque dossier `app2` :

```
nano app2/index.html
```

Placez le contenu suivant dans ce fichier :

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Docker Compose Demo</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/kognise/
water.css@latest/dist/dark.min.css">
</head>
<body>
    <h1>Web 2</h1>
    <p>This is a Docker Compose Demo Page 2.</p>
</body>
</html>
```

Enregistrez et fermez le fichier lorsque vous avez terminé. Si vous utilisez `nano`, vous pouvez le faire en tapant `CTRL+X`, puis `Y` et `ENTER` pour confirmer.

▼ dossier app3

- À l'aide de votre éditeur de texte préféré, créez un nouveau `index.html` fichier dans chaque dossier `app3` :

```
nano app3/index.html
```

Placez le contenu suivant dans ce fichier :

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Docker Compose Demo</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/kognise/
water.css@latest/dist/dark.min.css">
</head>
<body>
```

```
    <h1>Web 3</h1>
    <p>This is a Docker Compose Demo Page 3.</p>
</body>
</html>
```

> 💡 Enregistrez et fermez le fichier lorsque vous avez terminé. Si vous utilisez `nano`, vous pouvez le faire en tapant `CTRL+X`, puis `Y` et `ENTER` pour confirmer.

- modifier par la suit le fichier `docker-compose.yml` :

```
nano docker-compose.yml
```

Insérez le contenu suivant dans votre `docker-compose.yml` fichier :

```
version: '3.7'
services:
  web1:
    image: nginx:alpine
    ports:
      - "8001:80"
    volumes:
      - ./app1:/usr/share/nginx/html
  web2:
    image: nginx:alpine
    ports:
      - "8002:80"
    volumes:
      - ./app2:/usr/share/nginx/html
  web3:
    image: nginx:alpine
    ports:
      - "8003:80"
    volumes:
      - ./app3:/usr/share/nginx/html
```

- Enregistrez et fermez le fichier.

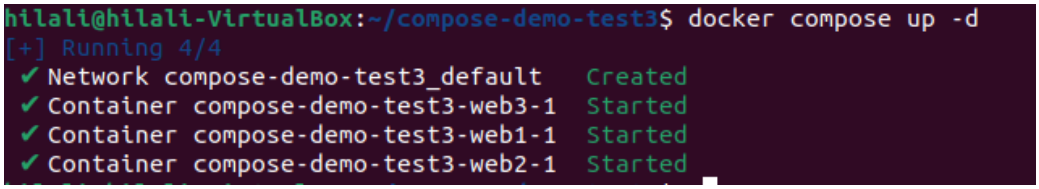
> 💡 les sévices web1 et web2 et web3 ne doit pas avoir le même port!

▼ Étape 2- Exécution de Docker Compose

- Avec le `docker-compose.yml` fichier en place, vous pouvez maintenant exécuter Docker Compose pour mettre votre environnement en place.

```
docker compose up -d
```

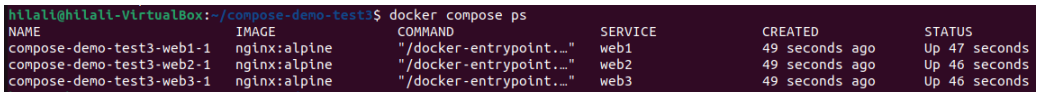
Docker Compose recherchera d'abord l'image définie sur votre système local, et s'il ne trouve pas l'image, il la téléchargera à partir de Docker Hub. Vous verrez une sortie comme celle-ci :



Votre environnement est maintenant opérationnel en arrière-plan. Pour vérifier que le conteneur est actif, vous pouvez exécuter :

```
docker compose ps
```

Cette commande vous montrera des informations sur les conteneurs en cours d'exécution et leur état, ainsi que sur les redirections de port actuellement en place :



Vous pouvez maintenant accéder à l'application de démonstration de votre choix en pointant votre navigateur soit `localhost:8001` (pour l’app1 8002→app2,8003→app3)

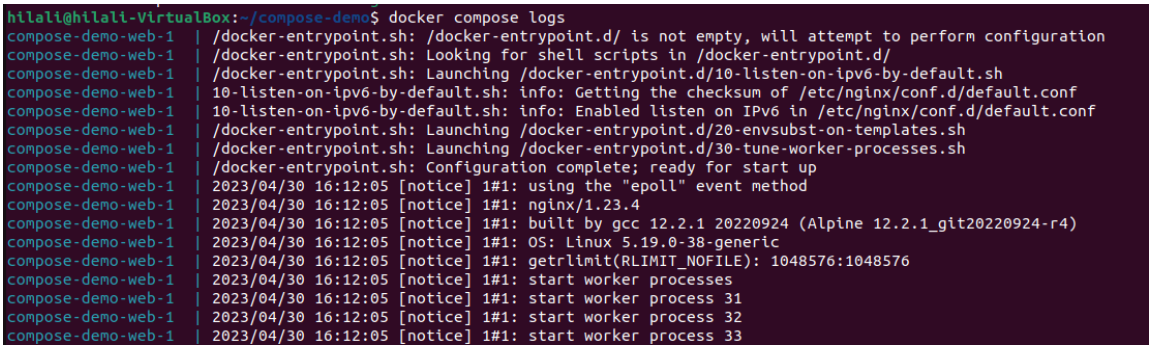
▼ Étape 4 - Se familiariser avec les commandes Docker Compose

- Vous avez vu comment configurer un `docker-compose.yml` fichier et mettre en place votre environnement avec `docker compose up`. Vous allez maintenant voir comment utiliser les commandes Docker Compose pour gérer et interagir avec votre environnement conteneurisé.Pour vérifier les logs produits par votre conteneur Nginx, vous pouvez utiliser la `logs` commande :

```
docker compose logs
```

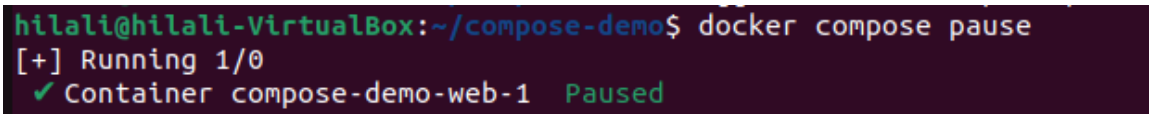
Cette commande peut être utile pour déboguer des problèmes de conteneurs et pour surveiller l'état des conteneurs dans un projet Docker Compose.

Vous verrez une sortie semblable à celle-ci :



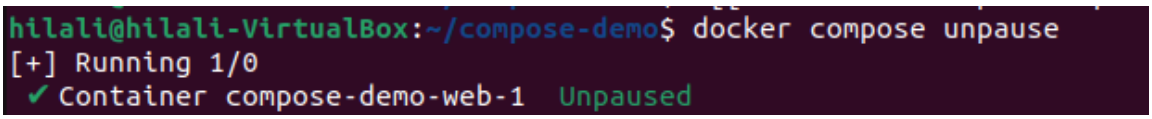
- Si vous souhaitez suspendre l'exécution de l'environnement sans modifier l'état actuel de vos conteneurs, vous pouvez utiliser :

```
docker compose pause
```



- Pour reprendre l'exécution après avoir émis une pause :

```
docker compose unpause
```



- La `stop` commande mettra fin à l'exécution du conteneur, mais elle ne détruira aucune donnée associée à vos conteneurs :

```
docker compose stop
```

```
hilali@hilali-VirtualBox:~/compose-demo$ docker compose stop
[+] Running 1/1
 ✔ Container compose-demo-web-1  Stopped
```

- Si vous souhaitez supprimer les conteneurs, les réseaux et les volumes associés à cet environnement conteneurisé, utilisez la `down` commande :

```
docker compose down
```

```
hilali@hilali-VirtualBox:~/compose-demo$ docker compose down
[+] Running 2/2
 ✔ Container compose-demo-web-1  Removed
 ✔ Network compose-demo_default  Removed
```

Notez que cela ne supprimera pas l'image de base utilisée par Docker Compose pour faire tourner votre environnement (dans votre cas, `nginx:alpine`). De cette façon, chaque fois que vous relancerez votre environnement avec un `docker compose up`, le processus sera beaucoup plus rapide puisque l'image est déjà sur votre système.

```
hilali@hilali-VirtualBox:~/compose-demo/app$ docker compose up -d
[+] Running 1/0
 ✔ Container compose-demo-web-1  Running
                                                    0.0s
```

en 0.0s

- Si vous souhaitez également supprimer l'image de base de votre système, vous pouvez utiliser :

```
docker image rm nginx:alpine
```

```
hilali@hilali-VirtualBox:~/compose-demo$ docker image rm nginx:alpine
Untagged: nginx:alpine
Untagged: nginx@sha256:dd2a9179765849767b10e2adde7e10c4ad6b7e4d4846e6b77ec93f080cd2db27
Deleted: sha256:8e75cbc5b25c8438fcfe2e7c12c98409d5f161cbb668d6c444e02796691ada70
Deleted: sha256:f301a4112756ab559d9c78e8ed3625dab81f91803dfeabbc4f9184c878b1f3b1
Deleted: sha256:d794631f2dea08ec92bc28f93ac8c1079c505aef791e86cc2bd5566904d2d581
Deleted: sha256:0d2f2fd89d17527e0808abf8debc4d22c1b3670894eeb12ecee580fe05719dec
Deleted: sha256:13ec71ce1944eb1de9f7fe3bbee31e4355476075b70f8b395a68d90ca849f111
Deleted: sha256:8997729a28eb948a9a00aa56b19143cff03e0ced25280473fff15c461860dfa7
Deleted: sha256:193b9708a46833ccb84791d3d58bf0d5428c37fc8fd80c951d3ca15cca5091c6
Deleted: sha256:f1417ff83b319fbdae6dd9cd6d8c9c88002dcd75ecf6ec201c8c6894681cf2b5
```