Docker Compose : Introduction

Table des matières

I. Docker Compose : les bases

Durée : 1 h 30

Prérequis :

- **Notion** : Docker
- **Notion** : Conteneur

Contexte

Pourquoi utiliser Docker Compose alors que Docker existe déjà ? Docker affiche ses limites lorsqu'on souhaite créer et gérer les interactions entre plusieurs conteneurs. Docker Compose répond à ce besoin en permettant de créer et gérer des applications multi-conteneurs.

Concrètement, Docker compose est un outil qui s'appuie sur Docker pour gérer des applications complexes avec plusieurs conteneurs. Nous allons découvrir ici comment installer et configurer Docker Compose.

A. Introduction à Docker Compose

Méthode **Installation de Docker Compose**

Prérequis : avoir installé Docker.

Pour installer Docker Compose sous un environnement Linux, exécutez les commandes suivantes.

Étape 1 : téléchargez la dernière version de Docker Compose :

```
# sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Étape 2 : modifiez les droits pour pouvoir exécuter le binaire :

```
# sudo chmod +x /usr/local/bin/docker-compose
```

Étape 3 : vérifiez la version installée :

```
# docker-compose version
```

Le fichier de configuration Docker Compose

Le « ***docker-compose.yml*** » est le fichier de configuration indispensable au fonctionnement de Docker Compose. Dans ce fichier sont listés les conteneurs gérés par Docker Compose et sont définis leurs interactions.

La nomenclature « ***docker-compose.yml*** » est à respecter obligatoirement, il s'agit d'un standard sans lequel Docker Compose ne peut trouver le fichier de configuration. Ce fichier doit également respecter le format **Yaml**.

Exemple **Fichier de configuration Docker-Compose**

```
----------------------------------------------------------------------------------------------------
version: '3'
services:
   web:
      image: nginx
      ports:
      - "80:8080" redis: image: redis:alpine
   Redis :
    Image : redis : alpine
----------------------------------------------------------------------------------------------------
```

Dans cet exemple, notre application est composée de deux services : un serveur web et une base de données. Cette architecture est reflétée dans le fichier de configuration où l'on déclare deux services : « ***Web*** » et « ***Redis*** ».

Pour chacun des services déclarés dans le bloc « ***Services*** », on peut détailler des paramètres tels que le nom de l'image utilisé pour créer le conteneur, le port exposé, etc.

Ici, le « ***docker compose configuration file*** » va créer le conteneur « ***Web*** » à partir de l'image « ***Nginx*** » et exposer le port 8080.

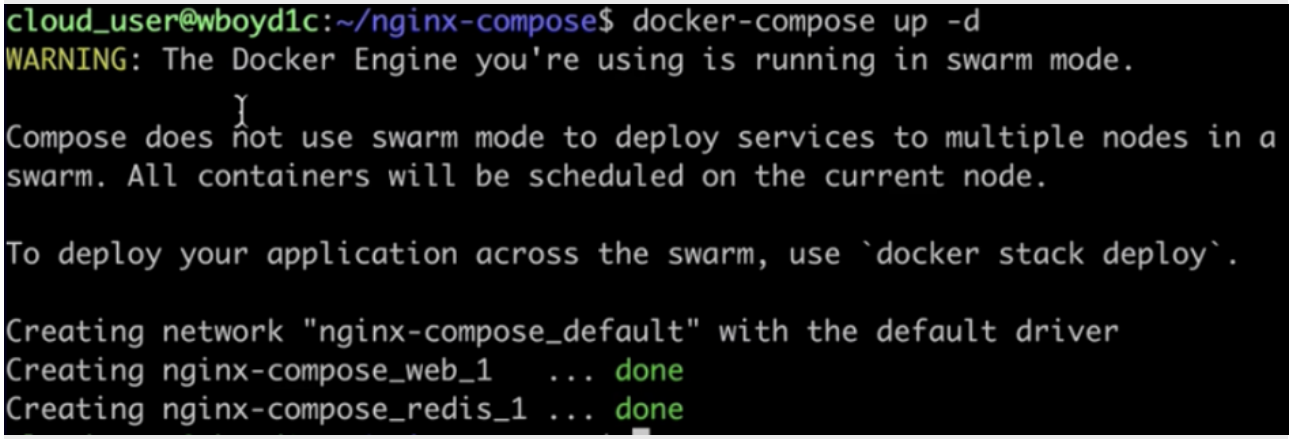
Les commandes Docker Compose

```
1 # docker-compose up
```

Cette commande permet de créer les ressources listées dans le fichier « ***docker-compose.yml*** ».

Exemple

Output de la commande « ***docker-compose up*** » appliquée au fichier de configuration de l'exemple précédent :



L'option « ***-d*** » (comme avec Docker) permet de lancer les conteneurs en mode détaché.

Remarque

Exécutez cette commande à partir du même répertoire où se trouve votre fichier « ***docker-compose.yml*** ».

```
1 #docker-compose ps
```

Cette commande liste tous le conteneurs (services) opérationnels et gérés par Docker Compose.

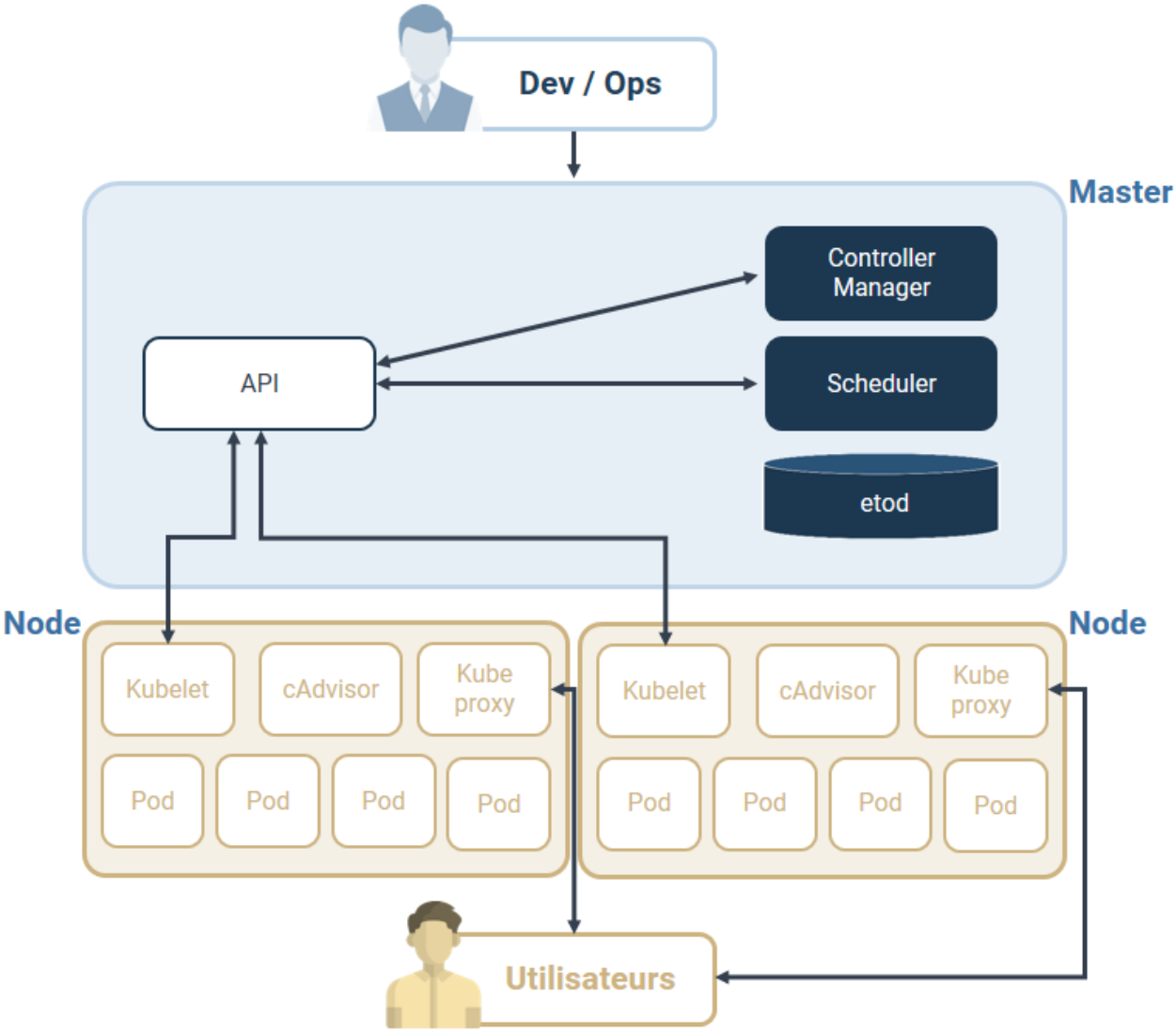
```
1 #docker-compose down
```

Cette commande permet d'arrêter et de supprimer les ressources crées par la commande « ***docker-compose up*** ».

Comparaison entre Docker Compose et Kubernetes

Tout comme Docker Compose, Kubernetes permet la gestion de plusieurs conteneurs et services.

Kubernetes se distingue par sa capacité à déployer des clusters et à gérer les conteneurs présents dans différents hôtes, contrairement à Docker Compose qui se limite à un seul hôte. Kubernetes apporte donc une dimension supplémentaire par rapport à Docker, en introduisant la notion de « ***cluster*** » et en apportant d'autres fonctionnalités telles que le « ***load balancing*** ».



Exercice : Quiz

[solution n°1 p.21]

Question 1

Quel argument est utilisé dans le fichier de configuration « ***docker-compose*** » pour configurer les ports du conteneur et les associer à un port du localhost pour qu'ils puissent communiquer les deux ?

- Host :
- remap :
- ports :
- redirect :

Question 2

Dans quel format doit être écrit le fichier de configuration « ***docker-compose*** » ?

- ○ html
- ○ Json
- ○ Yaml
- ○ xml

Question 3

Quelle commande permet de lister tous les process créés par Docker Compose ?

- ○ docker-compose ps
- ○ docker-compose env list --ps
- ○ docker ps
- ○ docker swarm service ps

Question 4

Docker Compose peut faire appel à des « ***dockerfile*** » pour créer les services.

- ○ Vrai
- ○ Faux

Question 5

Lorsque vous partagez un projet Docker Compose, vous partagez les « ***dockerfile*** », le code Python si tel est le cas, et le fichier « ***docker-compose.txt*** ».

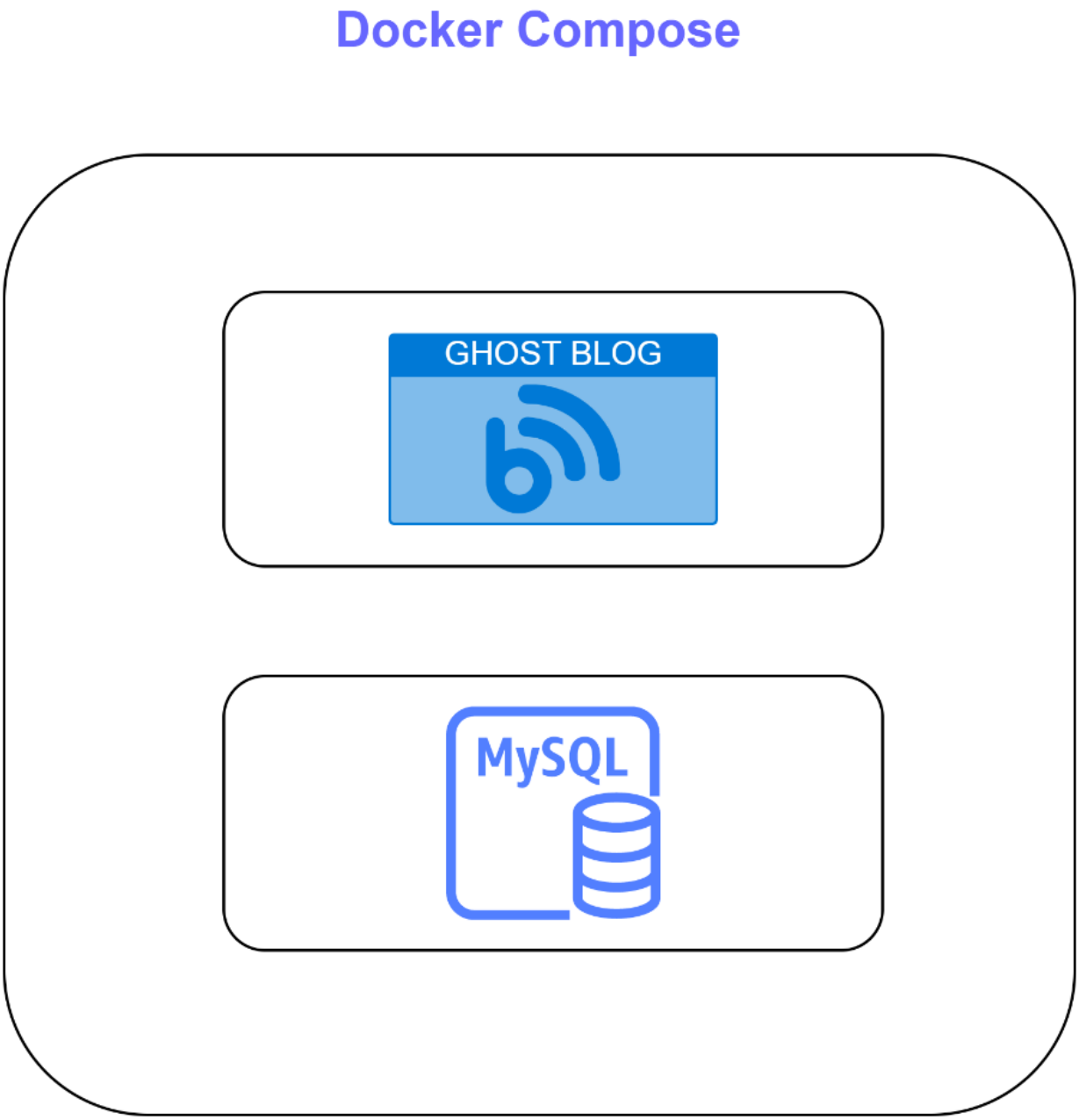
- ○ Vrai
- ○ Faux

III. Lab : création d'un blog Ghost sur le web

A. Présentation et architecture de l'application

Dans ce lab, nous allons créer une plateforme pour un blog sur Internet. Nous allons utiliser « *Ghost* », une application web *open source* qui permet de créer des blogs. Pour y accéder : ghost.org[1]

Pour le bon fonctionnement de la plateforme, nous allons également configurer une base de données MySQL qui va stocker le contenu du blog.



1 https://ghost.org/

B. Déploiement de la solution

Méthode **Préparation de l'environnement**

Comme indiqué précédemment, Docker Compose fait appel à Docker afin de créer et gérer les services décrits dans le fichier « ***docker-compose.yml*** ».

Il est donc obligatoire d'installer Docker avant de pouvoir utiliser Docker Compose. Le lab a été réalisé dans un environnement Linux (Ubuntu). Vous pouvez adapter les commandes CLI en fonction de l'environnement dans lequel vous travaillez.

Étape 1 : pour installer Docker, commencez par récupérer et installer le package Docker :

```
1 # sudo apt-get install docker.io
```

```
newuser@newuser-VirtualBox:~$ sudo apt-get install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd git git-man liberror-perl pigz runc ubuntu-fan
```

Étape 2 : une fois l'installation effectuée, vérifiez que Docker fonctionne correctement :

```
1 # sudo service docker status
```

```
newuser@newuser-VirtualBox:~$ sudo service docker status
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor prese
     Active: active (running) since Sun 2021-12-05 14:01:29 CET; 39s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 4850 (dockerd)
      Tasks: 8
     Memory: 40.3M
     CGroup: /system.slice/docker.service
             └─4850 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/con
```

Étape 3 : pour installer Docker Compose, exécutez les commandes listées précédemment. Une fois l'installation terminée, vérifiez la version Docker Compose installée.

```
newuser@newuser-VirtualBox:~$ docker-compose version
docker-compose version 1.23.2, build 1110ad01
docker-py version: 3.6.0
CPython version: 3.6.7
OpenSSL version: OpenSSL 1.1.0f  25 May 2017
```

Méthode **Analyse du fichier de configuration Docker Compose**

Nous allons examiner de plus près le contenu fichier « ***docker-compose.yml*** » qui sera utilisé pour créer notre application.

Étape 1 : le fichier de configuration commence toujours par l'argument « ***version*** ». Pour notre lab, nous allons travailler avec la version 3

Étape 2 : dans le bloc « ***services*** », déclarez l'ensemble des services que vous souhaitez créer. Chaque conteneur (service) commence par un nom unique dans le fichier de configuration. Dans notre exemple, le premier conteneur se nommera : « ***ghost*** »

```
version: '3'
services:
  ghost:
    image: ghost:1-alpine
```

Étape 3 : vous devez maintenant détailler pour chaque conteneur l'ensemble des paramètres à utiliser lors de sa création. Le premier argument est « ***image*** ». Il spécifie l'image que nous allons récupérer (comme lorsqu'on utilise la commande « ***pull*** » avec Docker) et que nous allons utiliser pour créer le conteneur.

Remarque

Il existe un autre argument que l'on peut utiliser à la place de « ***image*** » : c'est l'argument « ***build*** ». Il est utilisé lorsqu'on veut spécifier le chemin vers un fichier « ***dockerfile*** » et créer le conteneur en se basant sur ce fichier.

Méthode

Étape 4 : l'argument suivant est « ***restart*** ». Cet argument est important pour la continuité et la disponibilité d'un service. Lorsque cet argument est configuré avec la valeur « ***always*** », le service concerné redémarre automatiquement en cas d'erreur.

```
version: '3'
services:
  ghost:
    image: ghost:1-alpine
    container_name: ghost-blog
    restart: always
```

Étape 5 : un autre argument important permet au conteneur de communiquer avec l'hôte local (*localhost*) et les autres conteneurs : l'argument « ***ports*** ». Dans notre exemple, le conteneur exposera le port 2368 pour communiquer avec le *localhost* qui, de son côté, va utiliser le port 80.

```
version: '3'
services:
  ghost:
    image: ghost:1-alpine
    container_name: ghost-blog
    restart: always
    ports:
      - 80:2368
```

Remarque

La déclaration des ports se fait généralement suivant la règle suivante : on déclare un couple de ports « ***(HOST :CONTAINER)*** ». Si aucun port n'est spécifié côté *localhost*, ce dernier choisit un port aléatoirement pour communiquer avec le conteneur.

Méthode

Étape 6 : l'image « ***ghost :1-alpine*** » que nous allons utiliser dispose de plusieurs variables d'environnement. On peut considérer ces variables d'environnement comme des paramètres à configurer pour garantir le bon fonctionnement du service. Dans notre exemple, les variables d'environnement à configurer vont permettre au service « ***ghost*** » de communiquer avec la base de données. La base de données MySQL est spécifié, et un login et un mot de passe sont configurés. Ces informations vont permettre au service « ***ghost*** » de stocker ses données dans la *database* MySQL. Vous pouvez bien sûr modifier les valeurs de ces paramètres et choisir le login et mot de passe qui vous conviennent.

```
version: '3'
services:
  ghost:
    image: ghost:1-alpine
    container_name: ghost-blog
    restart: always
    ports:
      - 80:2368
    environment:
      database__client: mysql
      database__connection__host: mysql
      database__connection__user: root
      database__connection__password: P4sSw0rd0!
      database__connection__database: ghost
```

Étape 7 : utilisez ensuite l'argument « ***volumes*** » pour faire persister les données utilisées et générées par le conteneur. Dans notre exemple, nous avons besoin de préserver le contenu du dossier « ***/var/lib/ghost*** ». Pour ce faire, Docker crée un volume nommé « ***ghost-volume*** » qui permet d'écrire les données du conteneur en local sur le disque du *localhost*.

```
version: '3'
services:
  ghost:
    image: ghost:1-alpine
    container_name: ghost-blog
    restart: always
    ports:
      - 80:2368
    environment:
      database__client: mysql
      database__connection__host: mysql
      database__connection__user: root
      database__connection__password: P4sSw0rd0!
      database__connection__database: ghost
    volumes:
      - ghost-volume:/var/lib/ghost
```

Remarque

Les volumes à créer doivent également figurer dans le bloc « ***volumes*** » en plus d'être spécifié dans la description du service.

```
volumes:
  ghost-volume:
  mysql-volume:
```

Méthode

Étape 8 : un dernier argument est utilisé dans notre fichier : « ***depends_on*** ». Cet argument établit une dépendance entre les services et un ordre à respecter lors de la création des conteneurs. Dans notre exemple, le service « ***base de données*** » sera créé en premier avant de créer le service « ***blog*** ». De la même façon que le premier service est décrit dans le bloc « ***services*** », le deuxième service « ***mysql*** » est également décrit mais avec des valeurs et des paramètres différents :

```
mysql:
  image: mysql:5.7
  container_name: ghost-db
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: P4sSw0rd0!
  volumes:
    - mysql-volume:/var/lib/mysql
```

Étape 9 : maintenant que le fichier « *docker-compose.yml* » est prêt, vous allez pouvoir créer votre application. Commencez par créer un répertoire où vous allez mettre le fichier « ***docker-compose.yml*** » et copiez la version finale du fichier dedans :

```
-------------------------------------------------------------------------------------------------------------
version: '3'
services:
  ghost:
    image: ghost:1-alpine
    container_name: ghost-blog
    restart: always
    ports:
      - 80:2368
    environment:
      database__client: mysql
      database__connection__host: mysql
      database__connection__user: root
      database__connection__password: P4sSw0rd0!
      database__connection__database: ghost
    volumes:
      - ghost-volume:/var/lib/ghost
```

```
    depends_on:
      - mysql

  mysql:
    image: mysql:5.7
    container_name: ghost-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: P4sSw0rd0!
    volumes:
      - mysql-volume:/var/lib/mysql

volumes:
  ghost-volume:
  mysql-volume:
-------------------------------------------------------------------------------------------
--------------------------
```

Étape 10 : executez la commande :

```
# sudo docker-compose up -d
```

```
newuser@newuser-VirtualBox:~/application$ sudo docker-compose up -d
Creating network "application_default" with the default driver
Creating ghost-db ... done
Creating ghost-blog ... done
```

Remarque

Lors de l'exécution de la commande « ***docker-compose up*** », vous pouvez visualiser le process Docker pour récupérer les images « ***ghost :alpine-1*** » et « ***mysql*** ». Docker commence par vérifier si les images sont présentes en local avant de les télécharger à partir d'un « *Registry* Docker » sur Internet.

```
ffbb094f4f9e: Pull complete
df186527fc46: Pull complete
fa362a6aa7bd: Pull complete
5af7cb1a200e: Pull complete
949da226cc6d: Pull complete
bce007079ee9: Pull complete
eab9f076e5a3: Pull complete
c7b24c3f27af: Pull complete
6fc26ff6705a: Pull complete
bec5cdb5e7f7: Pull complete
6c1cb25f7525: Pull complete
Pulling ghost (ghost:1-alpine)...
1-alpine: Pulling from library/ghost
aad63a933944: Pull complete
976f06839970: Downloading [=======================================>
  17.09MB/22.54MBnload complete
18316e90c190: Download complete
7aba797547c3: Download complete
ef529ab4d1ec: Download complete
96e7ecd230d9: Download complete
59586d3e4b30: Downloading [=========>
  9.423MB/51.39MBting
```

Méthode **Analyse des logs Docker Compose**

Étape 1 : commencez tout d'abord par vérifier l'état des conteneurs avec la commande suivante :

```
# sudo docker ps
```

```
newuser@newuser-VirtualBox:~/application$ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATU
S          PORTS                                  NAMES
d3ea3f1da07c   ghost:1-alpine     "docker-entrypoint.s…"   54 minutes ago   Up 54
 minutes   0.0.0.0:80->2368/tcp, :::80->2368/tcp  ghost-blog
1d43a1aa15ab   mysql:5.7          "docker-entrypoint.s…"   54 minutes ago   Up 54
 minutes   3306/tcp, 33060/tcp                    ghost-db
```

Cette commande permet de visualiser l'état des différents conteneurs créés et fournit d'autres informations telles que le nom du conteneur, l'identifiant du conteneur et les ports utilisés.

Étape 2 : pour avoir plus de visibilité sur le fonctionnement du conteneur, vous pouvez consulter les logs docker en utilisant la commande suivante :

```
# docker logs  « Identifiant du conteneur »
```

```
newuser@newuser-VirtualBox:~/application$ sudo docker logs d3ea3f1da07c
(node:54) [DEP0096] DeprecationWarning: timers.unenroll() is deprecated. Please
 use clearTimeout instead.
(node:54) [DEP0095] DeprecationWarning: timers.enroll() is deprecated. Please u
se setTimeout instead.
[2021-12-05 19:12:17] INFO Finished database migration!
(node:1) [DEP0096] DeprecationWarning: timers.unenroll() is deprecated. Please
use clearTimeout instead.
(node:1) [DEP0095] DeprecationWarning: timers.enroll() is deprecated. Please us
e setTimeout instead.
[2021-12-05 19:12:18] WARN Theme's file locales/en.json not found.
[2021-12-05 19:12:19] INFO Ghost is running in production...
[2021-12-05 19:12:19] INFO Your blog is now available on http://localhost:2368/
[2021-12-05 19:12:19] INFO Ctrl+C to shut down
[2021-12-05 19:12:19] INFO Ghost boot 1.552s
```

Dans la capture précédente, nous avons vérifié les logs qui correspondent au conteneur « ***ghost*** ». Les logs confirment la création du service « ***ghost*** » sans erreurs.

Étape 3 : de même, on peut vérifier les logs du deuxième service, la base de données « ***MySQL*** » :

```
newuser@newuser-VirtualBox:~/application$ sudo docker logs 1d43a1aa15ab
2021-12-05 19:12:15+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Serv
er 5.7.36-1debian10 started.
2021-12-05 19:12:15+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mys
ql'
2021-12-05 19:12:15+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Serv
er 5.7.36-1debian10 started.
2021-12-05T19:12:16.038981Z 0 [Warning] TIMESTAMP with implicit DEFAULT value i
s deprecated. Please use --explicit_defaults_for_timestamp server option (see d
ocumentation for more details).
2021-12-05T19:12:16.045269Z 0 [Note] mysqld (mysqld 5.7.36) starting as process
 1 ...
```

Les logs retracent les différentes étapes de création de la base de données « ***MySQL*** » : création des tables, chiffrement, etc. Vers la fin des logs, on peut clairement distinguer les messages indiquant le succès de la configuration et la disponibilité de la base de données :

```
Location '/var/run/mysqld' in the path is accessible to all OS users. Consider
choosing a different directory.
2021-12-05T19:12:16.288555Z 0 [Note] Event Scheduler: Loaded 0 events
2021-12-05T19:12:16.289890Z 0 [Note] mysqld: ready for connections.
Version: '5.7.36'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Co
munity Server (GPL)
```

Méthode **Vérification de l'état des services créés**

Une fois l'installation terminée, les services affichent un statut « ***UP*** ». On peut maintenant les tester et s'assurer qu'ils fonctionnent correctement.

Étape 1 : dans un premier temps, connectez-vous à la base de données « ***MysQL*** » pour vérifier la création de la base de données « ***ghost*** » et de ses tables. Pour ce faire, connectez-vous au conteneur en exécutant la commande suivante :

```
#sudo docker exec -it ghost-db bash
```

Étape 2 : une fois à l'intérieur du conteneur, connectez-vous à la base de données avec la commande suivante :

```
#mysql -u root -p
```

Utilisez le mot de passe renseigné dans le fichier « ***docker-compose.yml*** ».

```
root@1d43a1aa15ab:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 5.7.36 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statemen

mysql>
```

Étape 3 : listez ensuite les bases de données disponibles :

```
#SHOW DATABASES ;
```

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| ghost              |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Étape 4 : vous devriez voir la base de données « ***ghost*** » listée. Sélectionnez celle-ci avec la commande :

```
#USE ghost
```

```
mysql> USE ghost
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Étape 5 : vérifiez l'existence des tables de la base de données utilisées par le service blog « ***ghost*** » :

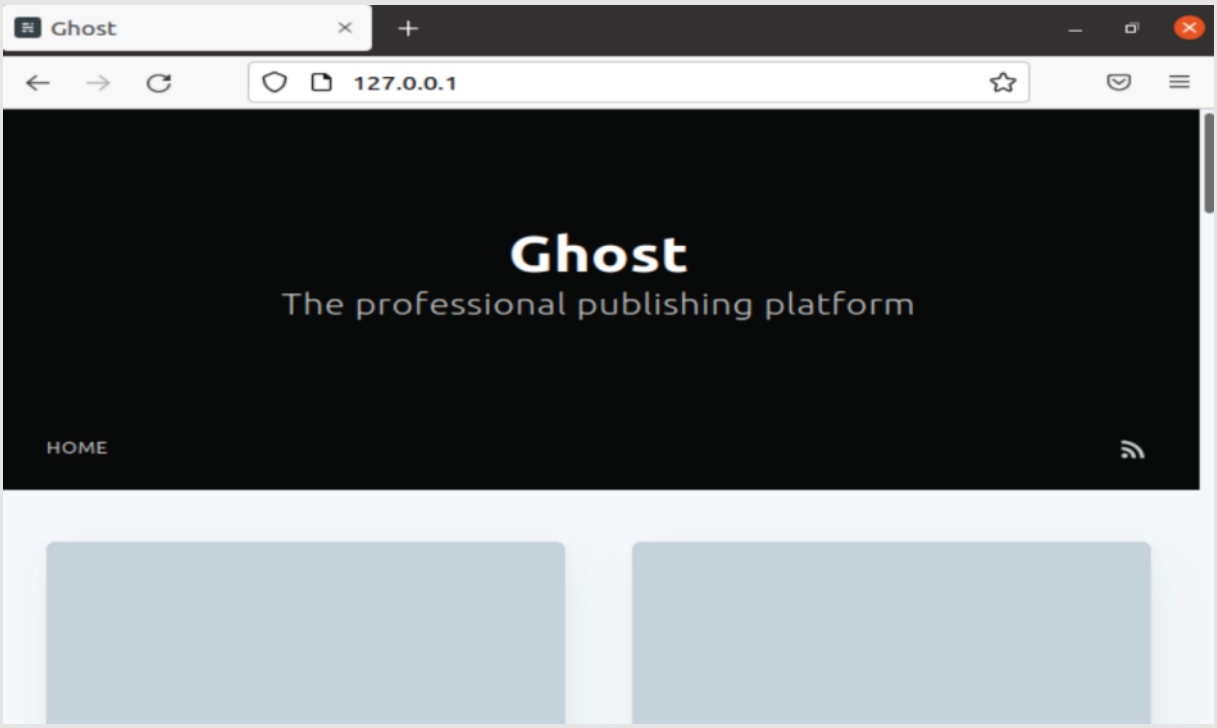
```
1 #SHOW tables ;
```

```
mysql> SHOW tables;
+----------------------------+
| Tables_in_ghost            |
+----------------------------+
| accesstokens               |
| app_fields                 |
| app_settings               |
| apps                       |
| brute                      |
| client_trusted_domains     |
| clients                    |
| invites                    |
| migrations                 |
| migrations_lock            |
| permissions                |
| permissions_apps           |
| permissions_roles          |
| permissions_users          |
| posts                      |
| posts_authors              |
| posts_tags                 |
| refreshtokens              |
| roles                      |
| roles_users                |
| settings                   |
| subscribers                |
```

Étape 6 : maintenant que votre base de données est prête, vérifiez si votre blog est accessible sur un navigateur web. Ouvrez un navigateur et entrez l'adresse suivante : http://127.0.0.1[1]

1 http://127.0.0.1/

Vous devriez voir la page suivante s'afficher :



Exercice : Quiz

[solution n°2 p.22]

Les questions font référence à un fichier Python « ***app.py*** », un « ***dockerfile*** » et le fichier « ***docker-compose*** ».

Fichier « ***app.py*** » :

```python
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Bonjour studi! j'ai été incrémenté {} fois.\n'.format(count
```

« ***dockerfile*** » :

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Fichier « ***docker-compose*** » :

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
  redis:
    image: "redis:alpine"
```

Question 1

Le fichier « ***requirements*** » est un fichier qui doit être placé dans le répertoire courant lorsque vous exécutez la commande Docker Compose.

- Vrai
- Faux

Question 2

Dans cette configuration, « ***build : .*** » signifie « construit l'image si le « ***dockerfile*** » n'existe pas ».

- Vrai
- Faux

Question 3

Dans cette configuration, l'image « ***redis*** » est récupérée du « ***Docker Hub*** ».

- Vrai
- Faux

Question 4

L'application Flask, bien que non spécifiée expressément, va utiliser le port 5000.

- Vrai
- Faux

Question 5

Dans l'application Flask, la ligne « ***cache = redis.Redis(host='redis', port=6379)*** » ne renseigne pas une adresse IP pour l'hôte. C'est une erreur.

- ○ Vrai
- ○ Faux

V. Essentiel

Docker compose est un outil qui permet de définir et de créer des applications composées de plusieurs conteneurs.

Le fichier de configuration **docker-compose.yml** est le fichier indispensable au fonctionnement de docker compose où sont définis l'ensemble des services (conteneurs) en plus d'établir la communication entre les différents conteneurs (exposition de ports).

Docker compose présente l'avantage de pouvoir créer et lancer l'ensemble des services définis dans le fichier de configuration yaml avec une seule commande : **docker-compose up**.

VI. Auto-évaluation

A. Exercice

Vous devez concevoir une architecture permettant à un drone de devenir autonome. Vous aurez besoin d'un module pour la détection de la piste et d'un module de *tracking* pour les autres objets. À chaque instant la position du drone est renseignée dans une base de données.

Question 1

[solution n°3 p.23]

À quelle architecture pensez-vous ? Quels outils de conteneurisation allez-vous utiliser ?

Vous devez créer un fichier « ***docker-compose.yml*** » permettant de déployer un environnement composé d'une base de données « ***Mysql*** » et d'un serveur « ***Apache PHP*** ».

Question 2

[solution n°4 p.23]

Rédigez le fichier correspondant, en prenant en considération les points suivants :

- Utilisez l'image « ***mysql :5.7*** » pour créer le conteneur « ***db*** ».
- Utilisez l'image « ***php :7.0-apache*** » pour créer le conteneur « ***webapp*** ».
- Assurez-vous que les services redémarrent automatiquement en cas de problème.
- Persistez le contenu des répertoires « ***/var/lib/mysql*** » du conteneur « ***db*** ».
- Persistez le contenu des répertoires « ***/var/www/html*** » et « ***/var/log/apache2*** » du conteneur « ***webapp*** ».

B. Test

Exercice 1 : Quiz

[solution n°5 p.24]

Vous trouvez ce fichier. Il est sauvegardé sous le nom de « ***unknown.dat*** ».

```
version: "2"
services:
    databases:
      image: mysql
      ports:
      - "3306:3306"
      environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=demodb
    web:
      image: nginx
```

Question 1

Ce fichier est un « ***dockerfile*** ».

- ○ Vrai
- ○ Faux

Question 2

Ce fichier respecte le format « ***yaml*** ».

- ○ Vrai
- ○ Faux

Question 3

Ce fichier Docker Compose comporte deux services.

- ○ Vrai
- ○ Faux

Question 4

Le port pour le service web est associé à celui de l'hôte : c'est le port 3306.

- ○ Vrai
- ○ Faux

Question 5

Les variables d'environnement sont définies dans le « ***dockerfile*** ».

- ○ Vrai
- ○ Faux

Solutions des exercices

Exercice p. 5 Solution n°1

Question 1

Quel argument est utilisé dans le fichier de configuration « ***docker-compose*** » pour configurer les ports du conteneur et les associer à un port du localhost pour qu'ils puissent communiquer les deux ?

- ○ Host :
- ○ remap :
- ◉ ports :
- ○ redirect **:**

Q Les ports permettent de faire communiquer les containers entre eux et avec l'hôte (c'est-à-dire votre ordinateur).

Question 2

Dans quel format doit être écrit le fichier de configuration « ***docker-compose*** » ?

- ○ html
- ○ Json
- ◉ Yaml
- ○ xml

Q Le fichier « **docker-compose** » doit être écrit au format yaml, à la différence du « ***dockerfile*** » qui est un simple fichier texte.

Question 3

Quelle commande permet de lister tous les process créés par Docker Compose ?

- ◉ docker-compose ps
- ○ docker-compose env list --ps
- ○ docker ps
- ○ docker swarm service ps

Q De la même façon que vous pouvez écrire « ***docker ps*** », dans ce cas vous utiliserez « ***docker compose ps*** ».

Question 4

Docker Compose peut faire appel à des « ***dockerfile*** » pour créer les services.

- ◉ Vrai
- ○ Faux

Q Dans Docker Compose, vous pouvez installer des services via leur « ***dockerfile*** » et / ou via l'image déjà présente dans le registre des images Docker.

Question 5

Lorsque vous partagez un projet Docker Compose, vous partagez les « ***dockerfile*** », le code Python si tel est le cas, et le fichier « ***docker-compose.txt*** ».

- ○ Vrai
- ◉ Faux

Dans un projet Docker Compose, vous partagez les fichiers de votre application, les « ***dockerfile*** » s'ils existent et le fichier « ***docker-compose.yaml*** ». Ce n'est pas un fichier de type txt.

Exercice p. 16 **Solution n°2**

Question 1

Le fichier « ***requirements*** » est un fichier qui doit être placé dans le répertoire courant lorsque vous exécutez la commande Docker Compose.

- ◉ Vrai
- ○ Faux

Le « ***dockerfile*** » en réalise une copie, du répertoire courant vers le container.

Question 2

Dans cette configuration, « ***build : .*** » signifie « construit l'image si le « ***dockerfile*** » n'existe pas ».

- ○ Vrai
- ◉ Faux

L'image docker construit l'image en utilisant le répertoire courant où se trouve le « ***dockerfile*** ».

Question 3

Dans cette configuration, l'image « ***redis*** » est récupérée du « ***Docker Hub*** ».

- ◉ Vrai
- ○ Faux

L'image « ***redis*** » est dépourvue de « ***dockerfile*** » dans ce cas, c'est d'ailleurs pourquoi on voit le mot-clé « ***images*** ». L'image est téléchargée depuis le registre Docker.

Question 4

L'application Flask, bien que non spécifiée expressément, va utiliser le port 5000.

- ◉ Vrai
- ○ Faux

Le mot clé « ***EXPOSE*** » signifie que le port 5000 sera exposé. De plus, l'application Flask tourne souvent sur ce port par défaut. D'après le fichier « ***docker-compose*** », on observe un mapping de port : « ***hôte (8000) - container (5000)*** ».

Question 5

Dans l'application Flask, la ligne « ***cache = redis.Redis(host='redis', port=6379)*** » ne renseigne pas une adresse IP pour l'hôte. C'est une erreur.

- ○ Vrai
- ◉ Faux

Dans Docker Compose, les services sont nommés. Il se crée un réseau interne à Docker Compose où chacun des services a un identifiant. Dans ce cas, le service « ***redis*** » est appelé « ***: 'redis' !*** ».

p. 18 **Solution n°3**

Pour gagner en compartimentation, il est conseillé d'utiliser des micro-services. Vous pourrez avoir un micro-service pour détecter la piste, un micro-service pour détecter les objets, et un autre micro-service pour héberger la base de données.

Vous pourriez écrire trois « ***dockerfile*** » et créer trois images. Vous pourriez ensuite créer un réseau et lancer chacun des containeurs de façon à ce qu'ils soient sur ce réseau. Le code maître pourra alors interagir avec ces services.

Vous pouvez opter aussi pour une solution beaucoup plus flexible et maintenable : l'usage de « ***docker-compose*** ». Automatiquement, les services pourront communiquer entre eux grâce à leur nom de service. Vous pourrez augmenter un ou plusieurs services de manière beaucoup plus simple.

p. 18 **Solution n°4**

```
version: '3.3'
services:
db:
image: mysql:5.7
volumes:
- db_data:/var/lib/mysql
restart: always
environment:
MYSQL_ROOT_PASSWORD: R00tP4ssw0rd
MYSQL_DATABASE: app_name
MYSQL_USER: db_user
MYSQL_PASSWORD: 4pp_n4meP4sswrd0!
webapp:
image: php:7.0-apache
volumes:
- app_data:/var/www/html
- app_logs:/var/logs/apache2
restart: always
links:
- db:newdb
volumes:
db_data:
app_data:
app_logs:
```

Dans le fichier « ***yaml*** » de la solution, l'argument « ***Link*** » a été utilisé. Cet argument permet de créer un alias « ***newdb*** » que le service « ***webapp*** » utilise pour communiquer avec le service « ***db*** ». L'emploi de links n'est pas nécessaire parce que « ***webapp*** » utilise l'alias par défaut « ***db*** » pour communiquer.

Exercice p. 18 Solution n°5

Question 1

Ce fichier est un « ***dockerfile*** ».

- ○ Vrai
- ◉ Faux

Le fichier « ***dockerfile*** » est un fichier texte dépourvu de formatage. Le fichier ci-dessus est un fichier « ***yaml*** » : c'est un fichier de type « ***docker-compose*** ».

Question 2

Ce fichier respecte le format « ***yaml*** ».

- ◉ Vrai
- ○ Faux

La structure « ***yaml*** » et son indentation sont caractéristiques.

Question 3

Ce fichier Docker Compose comporte deux services.

- ◉ Vrai
- ○ Faux

Les deux services sont « ***database*** » et « ***web*** ».

Question 4

Le port pour le service web est associé à celui de l'hôte : c'est le port 3306.

- ○ Vrai
- ◉ Faux

Attention, il s'agit du service « ***database*** ». Le port 3306 est associé au port 3306 de la machine hôte.

Question 5

Les variables d'environnement sont définies dans le « ***dockerfile*** ».

- ○ Vrai
- ◉ Faux

Le fichier n'est pas un « ***dockerfile*** ». Par ailleurs, les variables d'environnement sont utilisées par les containeurs et non par les images.