Orchestration Docker

Docker-compose

DERNIÈRE MISE À JOUR : 21/01/2024

Motivations

- Jusqu’à présent, nous avons travaillé avec une application monolithique, sur un seul hôte Docker.
- Configurer correctement une application multi-services peut être complexe
  - Démarrage de l'ensemble des conteneurs avec les bonnes options
  - Configuration du réseau
  - Configuration des volumes
  - Etc.
- Nous voulons déployer et gérer des applications de type micro-services

- Orchestration de conteneurs
- Conteneurs indépendants
- Réseau défini par logiciel
- Équilibrage de charge
- Observabilité
- Déploiement continu

Solution

- Une approche Infrastructure-as-code
- Docker Compose

Docker Compose

- Un outil complémentaire du Docker Engine
- Compose vous permet d'éviter de gérer individuellement des conteneurs qui forment les différents services de votre application.
  - Outil qui définit et exécute des applications multi-conteneurs
  - Utilise un fichier de configuration YAML dans lequel vous définissez les services de l'application
  - A l'aide d'une simple commande, vous contrôlez le cycle de vie de tous les conteneurs qui exécutent les différents services de l'application.
  - Déploiement sur une seule machine

Principes de Docker Compose

- L'utilisateur décrit son application (multi) conteneurs dans un fichier YAML appelé dockercompose.yml
- Exécuter docker-compose up pour démarrer l'application
  - Compose télécharge automatique les images, les build (si nécessaire), et démarre les conteneurs
  - Compose peut configurer des volumes, le réseau, et toutes autres options liées à Docker
- Compose agrège les sorties (ex logs) des différents conteneurs en un seul endroit, (si applicable)

Un exemple

```
version: "3.9"
services:
  web:
      build: .
      ports:
          - "5000:5000"
       volumes:
          - .:/code
          - logvolume01:/var/log
       depends_on:
          - redis
   redis:
       image: redis
volumes:
    logvolume01 :
```

- 2 services: web et redis
  - Par défaut, Compose créé un réseau bridge indépendant du réseau par défaut
  - La découverte de service fonctionne sur ce réseau
    - web peut contacter redis en utilisant son nom
- Un conteneur pour chaque service est créé
  - Pour web, une image est d'abord re-créé à partir du Dockerfile présent dans le répertoire courant
  - Pour redis, l'image redis est récupérée depuis Docker Hub
- Configuration du réseau
  - Le port 5000 de la machine hote est associé au port 5000 du conteneur web

Un exemple

```
version: "3.9"
services:
  web:
      build: .
      ports:
          - "5000:5000"
       volumes:
          - .:/code
          - logvolume01:/var/log
       depends_on:
          - redis
   redis:
       image: redis
volumes:
    logvolume01 :
```

- Contrôle de l'ordre de démarrage
  - Compose démarre les conteneurs dans un ordre correspondant aux dépendances introduites dans la composition
    - depends_on : permet de définir une dépendance explicite
    - D'autres balises induisent des dépendances implicites (links, volumes_from, etc.)
    - Attention: Compose attend que les conteneurs aient démarrés, mais pas que les services au sein des conteneurs soient prêts à fonctionner
- Les volumes
  - Déclaration d'un volume nommé: logvolume01
    - La déclaration est nécessaire pour partager un volume entre services
    - Le volume est ici utilisé par le conteneur web

Docker Compose?

Containeur basique

$ docker run debian /bin/echo "Salut"
Salut
$ docker run debian /bin/echo "Coucou"

Coucou

Lister les containers

$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

$ docker ps –a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
d0683f6462a5	debian	"/bin/echo Salut“	About a minute ago	Exited (0)	About a minute ago
e1794g7573b6	debian	"/bin/echo Coucou“	About a minute ago	Exited (0)	About a minute ago

Ré- exécuter un container

$ docker start -i vibrant_swanson
Salut

Logs d'un container

$ docker start -i vibrant_swanson
Salut

Ménage : suppression d'un container

$ docker rm vibrant_swanson

container en cours d’exécution

$ docker run -it debian /bin/bash
root@2c666d3ae783:/#
ps -a
PID TTY TIME CMD
6 ? 00:00:00 ps

Interrompre le container

$ docker stop $id_conteneur
$ docker kill $id_conteneur
$ docker pause $id_ conteneur

Docker Exec

$ docker exec -it vibrant_swanson /bin/bash
root@2c666d3ae783:/#

root@2c666d3ae783:/# exit