Conteneurisation: Docker Compose

Licence Professionnelle Métiers des Réseaux Informatiques et Télécommunications, Administration et Sécurité des Réseaux

Sebastien.Kramm@univ-rouen.fr

IUT R&T Rouen, site d'Elbeuf

2021-2022

(version du 21 janvier 2022)

Licence

Ce document est placé sous licence [CC BY-NC-SA]
(Attribution - Pas d'utilisation commerciale - Partage dans les mêmes conditions)

Pour plus de détails, voir la page Creative Commons.

Production

Ce document est généré à partir du fichier source LaTeXen 3 versions :

- Une version "diapos" pour le cours lui-même
  → suffixée par "_B"
- Une version pour l'impression, avec 4 diapos par page A4
  → suffixée par "_P"
- Une version pour la lecture à l'écran, similaire à la première mais sans les animations
  → suffixée par "_H"

Information

- Ce document contient des liens vers des pages ressources, qui apparaissent avec une couleur distinctive.
- Page du cours : universitice.univ-rouen.fr/course/view.php?id=17569

Persistance via les "*Bind mounts*"

- Cours précédent : "*Docker volumes*"
- Autre approche possible : "*bind mounts*"
- Comparatif :
  - "*Docker volumes*" : Docker gère l'emplacement sur la machine hôte
    → portable !
  - "*bind mounts*" : on peut lier n'importe quel dossier de la machine hôte avec n'importe quel dossier du conteneur.
- Exemple de création :

```
docker run -d \
    -it \
    --name devtest \
    --mount type=bind,source=$(pwd)/mon_app,target=/app \
    nginx:latest
```


Volume vs. Bind mounts

- Docker recommande les "volumes" (sauf dans des cas particuliers...)
- Inconvénient des *bind mounts*" : pas portable !
  Exemple : si on indique `/etc/share/local/mon_app`
  et que sur le server c'est `/opt/local/mon_app`
- Mais : on peut utiliser cette astuce pour avoir le code applicatif d'un service sur l'hôte
  → utile en phase de dev, évite d'avoir à relancer le conteneur à chaque fois (voir TP)

doc : https://docs.docker.com/storage/bind-mounts/

Docker Compose

- Permet de gérer un ensemble de conteneurs comme un tout
- Gère l'ensemble du cycle de vie :
  - construction/téléchargement des images ;
  - gestion des volumes et réseaux ;
  - lancement/arrêt des conteneurs.
- Le système entier est défini dans un fichier de description écrit en YAML : `docker-compose.yml`

YAML ? ? ?

Yet Another Markup Language / *YAML Ain't Markup Language* (WP)

- format de représentation de données par sérialisation
- permet de représenter des informations plus élaborées que le simple CSV (WP: "Comma separated values")
- meilleure lisibilité que le XML (moins verbeux)
- alternative au JSON

```
%YAML 1.2
---
YAML: YAML Ain't Markup Language™

What It Is:
  YAML is a human-friendly data serialization
  language for all programming languages.

YAML Resources:
  YAML Specifications:
  - YAML 1.2:
    - Revision 1.2.2       # Oct 1, 2021 *New*
    - Revision 1.2.1       # Oct 1, 2009
    - Revision 1.2.0       # Jul 21, 2009
  - YAML 1.1
```

- JSON plus adapté aux échanges de données entre logiciels
- YAML plus adapté à des fichiers de configuration

YAML : exemple

```
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    given:   Dorothy
    family:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     100.27
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
```


Syntaxe YAML

- Commentaires :#
- Les éléments de listes sont dénotés par le tiret (-), suivi d'une espace, à raison d'un élément par ligne.
- Arborescence représentée par l'indentation (2 espaces)
- Tableaux : sont de la forme `clé: valeur`, à raison d'une paire par ligne.

Principales commandes

- Syntaxe : `$ docker-compose [options] <commande>`
  → on retrouve les commandes des containers et images, mais qui vont concerner un "stack" entier
- A exécuter dans le même dossier que le fichier `docker-compose.yml`
  (on peut spécifier un autre fichier : `-f monFichier.yml`)
- Commandes "globales" :
  - `up`: Create and start containers
  - `down`: Stop and remove containers, networks
- On peut décomposer les tâches :
  - `build` : création des images
  - `create` : création des conteneurs
  - `start` : démarrage des services (conteneurs)
  - `stop` : arrêt des services
  - `rm` : suppression des conteneurs arretés
- Et bien d'autres choses, voir `$ docker-compose --help`

Syntaxe du fichier Yaml

- 2 clés obligatoires : `version` et `services`
- Sous "`services`" on précise la liste des conteneurs a démarrer.
- Pour chaque conteneur, on indique image source, ou "build" s'il faut la construire.
- Options : nom, volumes, mappage des ports, etc.

```
services:
  traefik:
    image: traefik:latest
    container_name: traefik
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /DOCKER/volumes/traefik/traefik.toml:/traefik.toml
```

(note : ici les volumes sont en fait des "*bind mounts*")

Autres directives

- On peut préciser qu'un service a besoin d'un autre pour fonctionner (ou plusieurs) :

```
services:
  monapp:
    depends_on:
      - service1
      - service2
```

- On peut préciser des variables d'environnement :

```
services:
  monapp:
    environment:
      MYSQL_ROOT_PASSWORD: test
      MYSQL_DATABASE: test
      MYSQL_USER: test
```


`docker-compose.yml` : exemple

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

Définition de deux services (conteneurs) nommés `web` et `redis`

- Le 1er est construit à partir d'un fichier `Dockerfile`, qui doit exister dans le dossier courant, et ouvre le port 5000.
- le 2e utilise une image existante
  (téléchargée si besoin depuis le registry)

Persistance

- On peut indiquer à la fois des "*named volumes*" et des "*bind mounts*".

```
services:
  monapp:
    volumes:
      - type: volume
      source: mydata
      target: /data

      - type: bind
      source: ./static
      target: /opt/app/static
```


Persistance

Une forme abrégée permet de spécifier les deux types :

```
services:
  db:
    image: postgres:latest
    volumes:
      - "/var/run/postgres/postgres.sock:/var/run/postgres/postgres.sock"
      - "dbdata:/var/lib/postgresql/data"

# création des "named volumes"
volumes:
mydata:
dbdata:
```

Réf : https://docs.docker.com/compose/compose-file/compose-file-v3/