Marquer comme terminé

Programme de la séance

Toutes les manipulations de cette séance sont à effectuer sur la VDI (bureau-distant) ou sur une machine équipée de `Docker`.

Il s'agit de faire tourner `postgres` dans un container `Docker`, ainsi qu'un serveur `postgraphile` qui constitue un *endpoint* `graphql` pour la base de données.

Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé. Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

Un container possède un système de fichier et s'appuie sur les fonctionnalités du système d'exploitation fournies par la machine hôte. Il utilise l'isolation de ressources (comme le processeur, la mémoire, les entrées et sorties et les connexions réseau).

Un container peut-être vu comme une mini-machine virtuelle, ou plutôt comme un *parasite*. Il est beaucoup plus léger (en termes de consommation d'espace) qu'une machine virtuelle car il utilise les éléments de système de l'hôte. Un container est instancié à partir d'une *image*, définie à partir d'une autre image. Par exemple, on peut définir un container qui héberge un serveur `graphql` à partir d'un container `node`.

`PostGraphile` permet de créer rapidement un serveur `GraphQL` qui expose une base de données `PostGres`. Il analyse pour cela le schéma de la base de données et génère automatiquement une API complète, en particulier des requêtes de jointure, qui permettent d'obtenir des données dénormalisées.

`Docker` sur la VDI (bureau distant)

Bureau distant - VDI

L'université met à disposition des machines virtuelles Linux, auxquelles vous pouvez vous connecter depuis chez vous (et depuis une machine de TP, sous Linux ou Windows)\ :

- si besoin, installer le client `VMware Horizon`
- lancer le client `VMware Horizon` (`vmware-view` en console) ou se rendre à l'URL https://bureau-distant.unicaen.fr/ avec le navigateur
- l'adresse du serveur est https://bureau-distant.unicaen.fr/
- depuis chez vous ou depuis le réseau `eduroam`, il peut vous être nécessaire de procéder à une double authentification. Voir ici pour les détails
- se connecter au serveur
- rentrer ses coordonnées de *persopass*
- choisir la machine virtuelle `Pédagogie Ubuntu 22`.

Attention Lorsque vous fermez la session de la VDI, le contenu de votre répertoire HOME est effacé. Il ne faut donc rien y stocker, et les paramètres des logiciels (entre autres l'historique et les identités gérés par le navigateur, mais également le fichier ~/.bashrc seront systématiquement effacés).

`docker` sur la VDI

Pour exécuter `docker` sur la VDI, vous devez être dans le groupe `docker`. Chaque commande `docker` ou `docker-compose` qui suit doit donc être précédée de `sudo -g docker` (demander des privilèges consistant à être membre du groupe `docker`).

Par exemple, pour faire tourner le container de test `hello-world`\ :

```
sudo -g docker docker run hello-world
```

C'est quand-même très fastidieux de faire précéder toutes vos commandes `docker` de `sudo -g docker`, donc je recommande de créer un *alias*\ :

```
$ alias docker='sudo -g docker docker'
$ alias docker-compose='sudo -g docker docker-compose'
```

Attention : les alias ne fonctionnent pas dans les scripts ! dans un script, `bash` n'interroge pas sa liste d'alias pour exécuter une commande. En revanche, les alias peuvent être paramétrés dans certains scripts.

Pour éviter de définir cet alias à chaque lancement de VDI, vous pouvez créer un script `~/Document/bashrc` contenant cette définition et utiliser `~/Documents/x11_user_startup_jammy-vdi.sh`, qui est exécuté au lancement de la VDI (voir https://faq-etu.unicaen.fr/x11_user_startup), pour installer ce `bashrc` à sa bonne place\ : `~/.bashrc`. Voici comment procéder\ :

?

```
$ cat ~/Documents/bashrc
alias docker='sudo -g docker docker'
alias docker-compose='sudo -g docker docker-compose'
$ cat ~/Documents/x11_user_startup_jammy-vdi.sh
if [ ! -f ~/.bashrc ]; then
  cp -f ~/Documents/bashrc ~/.bashrc
fi
```

Attention\ : lorsqu'on se déconnecte de la VDI, la machine n'est détruite que 15 mn après déconnexion. Le temps que le fichier `~/Documents/x11_user_startup_jammy-vdi.sh`, installez votre `~/.bashrc` à la main.

Mise en oeuvre rapide de `postgraphile`

1. créer un répertoire `graphile` et s'y rendre

2. créer un fichier `.env` comme suit\ :

```
POSTGRES_DB=commerce
POSTGRES_USER=postgres
POSTGRES_PASSWORD=change_me
```

3. dans un fichier `stack.yml`, définir une pile de container pour `PostGres`\ :

```
services:
  db:
    container_name: commerce-example-db
    image: postgres:16.1
    volumes:
      - datadir:/var/lib/postgresql/data
    env_file:
      - ./.env
    networks:
      - network
    ports:
      - 5432:5432

  adminer:
    image: adminer:4.8.1
    depends_on:
      - db
    ports:
      - 8080:8080

networks:
  network:

volumes:
  datadir:
```

Cette pile contient deux services :

- un service `db`, dont le container est nommé `commerce-example-db`, basé sur une image `postgres:16.1`. Il est pourvu d'un *volume persistant*, qui établit une liaison entre le système de fichier de l'hôte et celui du container. Il y a également un transfert de port entre l'hôte et le container, sur le port 5432. Ceci permet de contacter le container sur le port 5432, le port de `postgres`, à l'adresse http://127.0.0.1:5432, sans être obligé de connaître l'adresse IP du container.
- un service `adminer`, qui permet d'interagir avec la base de données dans le navigateur, à l'adresse http://127.0.0.1:8080

Sauf nommage explicite (ce qui est le cas du container `postgres` ci-dessus), les containers d'une pile sont automatiquement nommés à partir du nom du dossier qui héberge la pile et du nom de l'image. Ici, pour le service `adminer`, le nom du container sera `graphile_adminer_1`. De même, le volume sera nommé `graphile_datadir`.

Pour `Docker`, une notion importante est celle de *volume*. Il s'agit d'une liaison dynamique (ou montage) entre un point du système de fichier de l'hôte et celui du container. Les volumes permettent d'assurer la persistance des données modifiées dans un container. Par exemple, `postgres` enregistre ses données dans `/var/lib/postgresql/data` dans le container, qui est lié au volume `graphile_datadir` sur l'hôte.

Les volumes sont compliqués à gérer, d'autant plus sur la VDI où ils seront effacés lors de l'extinction. Plus de détails ici.

On peut maintenant démarrer la pile de containers\ :

4. démarrer la pile de containers

```
docker-compose -f stack.yml up -d
```

?

5. rentrer dans le container (y exécuter `bash`), examiner le système de fichier, switcher sur l'utilisateur `postgres`, se connecter à `postgres`

```
$ docker exec -it commerce-example-db bash
fab64a83ec0d:/# ls
...
su - postgres
fab64a83ec0d:~$ psql
psql (16.1)
Type "help" for help.

postgres=# \c
You are now connected to database "postgres" as user "postgres".
```

6. sortir du container et insérer le dump des données

```
docker exec -i commerce-example-db psql -U postgres -d commerce < commerce.sql
```

7. lancer le service `PostGraphile` sur la VDI

```
npx postgraphile -c postgres://postgres:change_me@127.0.0.1:5432/commerce --watch --enhance-graphiql --dynamic-json
```

8. à http://127.0.0.1:5000/graphiql, explorer l'interface fournie, tester les requêtes et les mutations.

Vous pouvez consulter une liste de commandes `Docker` ici.

Graphile dans un container

d'après https://www.graphile.org/postgraphile/running-postgraphile-in-docker/

Nous allons maintenant faire tourner `postgraphile` dans un container, intégré à la pile contenant le serveur `postgres`.

Pour cela, nous devons procéder en deux étapes\ :

1. faire en sorte que le serveur `postgres` charge automatiquement les données. Pour cela, on définira un container personnalisé basé sur l'image `postgres:11.0-alpine` dont le dossier `/docker-entrypoint-initdb.d/` contiendra les fichiers `.sql` permettant de reconstituer les données.
2. définir un container personnalisé pour `postgraphile`
3. intégrer ces containers à la pile

Container personnalisé `postgres`

- créer un répertoire `db`
- y enregistrer un fichier nommé `Dockerfile` de définition du nouveau container `postgres`\ :

```
FROM postgres:16.1-alpine
COPY ./init/ /docker-entrypoint-initdb.d/
```

- ce fichier de paramétrage indique qu'il faut créer un container à partir de l'image `postgres:11.0-alpine` et y copier le contenu du répertoire `init` dans le répertoire `/docker-entrypoint-initdb.d/`
- créer un répertoire `init` dans le répertoire `db`
- à partir du fichier `commerce.sql`, définir un fichier `00-database.sql` qui contient le schéma des données (les requêtes de création de tables)\ :

```
\connect commerce

CREATE TABLE ...
```

- dans un fichier `01-data.sql` contenant les données (requêtes d'insertion)\ :

```
\connect commerce

INSERT INTO ...
```

Container personnalisé `postgraphile`

1. ajouter l'URL dans le fichier `.env`

```
DATABASE_URL=postgres://postgres:change_me@db:5432/commerce
```

2. dans un répertoire `graphql`, créer un fichier de configuration `Dockerfile` pour le container qui va accueillir le serveur `postgraphile`\ :

?

```
FROM node:alpine
LABEL description="Instant high-performance GraphQL API for your PostgreSQL database https://github.com/graphile/postgraphile"

# Install PostGraphile and PostGraphile connection filter plugin
RUN npm install -g postgraphile
RUN npm install -g postgraphile-plugin-connection-filter

EXPOSE 5000
ENTRYPOINT ["postgraphile", "-n", "0.0.0.0"]
```

Définition de la pile de containers personnalisés

La structure de votre répertoire doit être la suivante\ :

```
.
├── db
│   ├── Dockerfile
│   └── init
│       ├── 00-database.sql
│       └── 01-data.sql
├── .env
├── graphql
│   └── Dockerfile
├── stack.yml
```

Le nouveau fichier de définition de la pile de containers est désormais comme suit\ :

```
services:
  db:
    container_name: commerce-example-db
    image: commerce-example-db
    build:
      context: ./db
    volumes:
      - datadir:/var/lib/postgresql/data
    env_file:
      - ./.env
    networks:
      - network
    ports:
      - 5432:5432

  adminer:
    image: adminer
    depends_on:
      - db
    ports:
      - 8080:8080

  graphql:
    container_name: commerce-example-graphql
    image: commerce-example-graphql
    restart: always
    build:
      context: ./graphql
    env_file:
      - ./.env
    depends_on:
      - db
    networks:
      - network
    ports:
      - 5433:5433
    command: ["--enhance-graphiql", "--connection", "${DATABASE_URL}", "--port", "5433", "--schema", "public", "--append-
plugins", "postgraphile-plugin-connection-filter"]

networks:
  network:

volumes:
  datadir:
```

Lorsque les containers n'existent pas, ils seront créés au premier démarrage de la pile. Pour forcer leur reconstruction, il faudra lancer\ :

?

```
    docker-compose -f stack.yml up -d --build
```

Notez que le service `graphql` est paramétré de façon à redémarrer toujours `restart: always`, i.e. tant qu'une erreur l'empêche de démarrer. C'est le cas ici, car bien qu'il dépende du service `db`, il faut certainement un peu de temps pour que la base de données initiale soit insérée et `postgraphile` ne démarre pas du premier coup.

Vous pouvez maintenant accéder à `postgraphile` à http://127.0.0.1:5433/graphiql

Modifié le: mardi 5 novembre 2024, 11:49

◄ Tests ChatGPT

Choisir un élément

Aller à...

Gestion des volumes Docker ►

mentions légales . vie privée . charte utilisation . unicaen . cemu . moodle