Exécuter une application multi-conteneur avec docker compose

Dans cette section et celles qui suivront, nous allons nous baser sur l'exemple que vous pouvez retrouver ici : https://github.com/eliesjebri/getting-started.git

Ce que vous allez faire, c'est de récupérer la partie */app* de ce projet, créer un Dockerfile dans ce dossier et coller ces instructions dans ce fichier :

```
FROM node:12-alpine
WORKDIR /app
RUN apk add --no-cache python2 g++ make
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Vous allez ensuite créer un dossier nommé nginx à l'intérieur de app/ dans lequel vous allez créer deux fichiers : un fichier appelé *nginx.conf* et un autre Dockerfile.

Dans *nginx.conf*, voici les instructions à intégrer :

```
server {
    listen [::]:80;
    listen 80;
    server_name nodeserver;
    charset utf-8;
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_pass http://nodeserver:3000;
    }
}
```

Et dans le Dockerfile, il faut ajouter les lignes suivantes :

```
FROM nginx
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Vous avez maintenant les environnements nécessaires pour la suite.

Créer un docker compose

Une fois que vous avez vos bases de conteneurs, il faut créer le fichier qui va les exécuter simultanément, à savoir le fichier nommé **docker-compose.yml**. Pour ce faire, rendez-vous à la racine du projet à savoir le dossier */app* et créer ce fichier. Ajoutez à l'intérieur de celui-ci les instructions suivantes :

```
services:
  nodeserver:
    build:
      context: .
    ports:
      - "3000:3000"
  nginx:
    restart: always
    build:
      context: ./nginx
    ports:
      - "80:80"
```

Examinons un peu ce qui se trouve dans ce fichier. Tout d'abord, nous voyons la version que nous souhaitons exécuter.

Ensuite, les lignes qui suivent contiennent l'ensemble des services, c'est-à-dire la désignation de tous les conteneurs à exécuter à savoir celui de l'application node JS et du serveur Nginx. Vous êtes libre de renommer les services comme vous le souhaitez.

Chaque service possède des spécifications précises comme les informations sur l'emplacement des dépendances à exécuter. Ici, dans la partie build, nous voyons un paramètre appelé context qui désigne l'emplacement du Dockerfile de l'image docker.

Nous avons également une partie appelée ports dans laquelle on spécifie le port vers lequel pointe l'application une fois celle-ci exécutée.

Cet exemple montre un docker-compose très basique, nous allons étoffer ce dernier le long de cet article avec d'autres fonctionnalités.

Exécuter un docker compose

Avant d'ajouter d'autres informations à l'intérieur de ce fichier docker-compose, nous allons tout d'abord tester si celui-ci fonctionne correctement. Pour ce faire, nous allons exécuter l'application multiconteneur avec la commande suivante :

```
docker compose up
```

Avec cette commande, l'application se lance en avant-plan, c'est-à-dire que vous pouvez interagir avec l'application directement à partir de l'invite de commande. Cependant, cela vous empêche d'exécuter d'autres commandes jusqu'à ce que l'on arrête les conteneurs.

Pour remédier à cela, on peut lancer l'application en mode détachée, comme c'est le cas de docker run. Pour ce faire il faut juste ajouter le mot clé **-d** à la fin de la commande précédente.

Une fois que vous avez exécuté cette commande, vous pouvez taper localhost sur votre navigateur pour avoir accès à l'application.

Une autre manière de vérifier si tout s'est bien passé est de taper la commande suivante qui va lister les images disponibles :

```
docker images
```

Vous pouvez aussi ne lister que les services en cours d'exécution en tapant la commande ci-dessous :

```
docker compose ps
```

Pour arrêter les services, il vous suffit de taper la commande **docker-compose down** ou **CTRL+C** dans le cas où vous avez lancé l'application sans utiliser la mode détachée.

Autres fonctionnalités

Nous allons maintenant ajouter quelques fonctionnalités en plus à notre application.

Monter un volume et définir une variable d'environnement

Commençons par le volume. Ce dernier permet d'identifier des dossiers ou des fichiers nécessaires pour que l'on puisse modifier ou ajouter des fonctionnalités à l'application, sans pour autant réexécuter docker-compose.

Une variable d'environnement, quant à lui, permet de définir un ensemble de clés/valeurs qui peut spécifier l'état dans lequel on souhaite lancer le conteneur ou un paramètre nécessaire à l'application pour son bon fonctionnement.

Par exemple, nous allons ajouter le répertoire contenant le code source de notre application dans la section volume de *docker-compose.yml*. Nous allons également ajouter une variable d'environnement spécifiant le type d'environnement avec lequel on souhaite exécuter l'application comme ceci :

```
version: "3.8"
services:
    nodeserver:
        build:
            context: .
        volumes:
            - ./src:/app/src
        environment:
            NODE_ENV: production
        ports:
            - "3000:3000"
    nginx:
        restart: always
        build:
            context: ./nginx
        ports:
            - "80:80"
```

Ici, l'instruction – *./src:/app/src* spécifie que nous allons récupérer le code source de notre application dans le dossier *src* à la racine de notre projet. Nous allons par la suite le copier dans un autre fichier *src*. Ce dernier va être créé dans le dossier *app* que nous avons spécifié dans la section *WORKDIR* du dockerfile.

Maintenant, si l'on modifie un peu notre code source et que l'on actualise l'application, la modification sera prise en compte instantanément.

Attribuer et activer des profils

Un profil sert à activer ou non un service présent dans le fichier docker-compose selon la ou les valeurs spécifiées dans cette clause. Cela veut dire que si le profil défini n'est pas activé au moment du démarrage du conteneur, le service en question ne démarrera pas.

Considérons notre précédent exemple et ajoutons un profil dans le service nodeserver :

```
services:
  nodeserver:
    build:
      context: .
    volumes:
      - ./src:/app/src
    environment:
      NODE_ENV: production
    profiles:
      - debug
    ports:
      - "3000:3000"
  nginx:
    restart: always
    build:
      context: ./nginx
    ports:
      - "80:80"
```

Pour lancer un conteneur en activant un profil, il faut exécuter la commande suivante :

```
docker compose --profile debug up -d
```

où debug est le nom du profil que nous venons de spécifier dans le fichier docker-compose.

Vous savez maintenant utiliser docker-compose afin de lancer vos applications multiconteneurs en une seule commande. Les fonctionnalités que nous venons de voir vont vous permettre de bien paramétrer votre fichier. Cela permet de mieux gérer l'exécution de chaque conteneur inclus dans docker-compose et lancer votre application selon vos besoins.