Guide d’installation

Déploiement du projet

Le projet a été conçu afin d’être déployé 100% de manière locale. Afin d’arriver à ce but, nous avons utilisé Docker afin de pouvoir déployer l’application sur Windows, Mac et Linux . Dans le projet, nous pouvons trouver deux fichiers Dockerfile, un à la racine du projet et l’autre à l’intérieur d’un répertoire cpp-invoker. Afin de déployer l’application efficacement, nous avons utilisé le système d’orchestration Docker compose. À ce fait, vous trouverez deux fichiers .yml qui servent pour le déploiement de Docker compose, un de ces fichier est `docker-compose.yml` afin de déployer l’application à partir des images pré-construites.

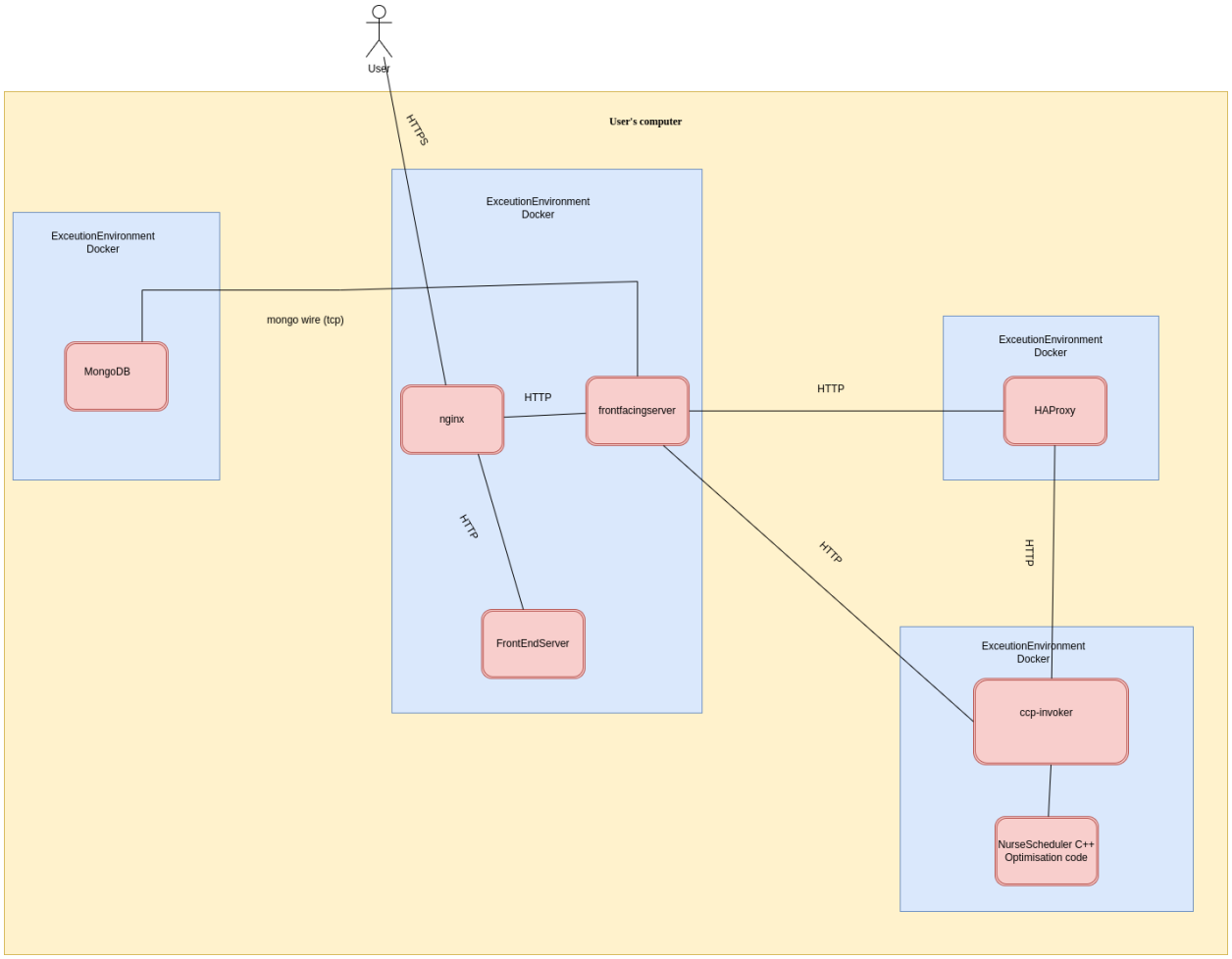
```
docker compose up
```

L’autre fichier est docker-compose-dev.yml il est utilisé pour des fins de développement comme expliqué dans le document [1].

Spécification de l’ordinateur

Afin d’assurer que tous les conteneurs Docker s’exécutent ensemble, nous recommandons d’avoir au moins 8 GB de mémoire vive.

Diagramme de déploiement



Description des services

Comme indiqué dans le diagramme, nous avons 4 services différents.

Service de la base de données

Le premier service est utilisé pour rouler la BD MongoDB.

Service principal

Le deuxième service inclut trois serveurs, Nginx, un serveur appelé frontfacingserver et le serveur du Front-End.

Le serveur Nginx assure la communication HTTPS avec le navigateur via un certificat de type «self-signed» , il s'assure aussi du reverse proxy afin d'acheminer les communications avec le frontfacingserver et le serveur du Front-End.
Le serveur de Front-End est un serveur Angular dont le code peut se trouver dans la sous-directoire front-end-application.
Le serveur de frontfacingserver communique avec la base de données et avec le serveur de Front-End.

Service de répartiteur de charge

Le troisième service inclut un serveur HAProxy qui a la responsabilité d'équilibrer la charge entre les différents serveurs qui exécutent le code C++. La répartition de charge se fait selon une stratégie «round-robin». Dans son fonctionnement, HAProxy s'assure de la disponibilité des serveurs et exclut les serveurs non disponibles.

Service d'exécution de la demande d'optimisation

Le quatrième service inclut un serveur Python appelé cpp-invoker qui s'occupe de gérer la file des demandes d'optimisation et d'appeler le code C++ via la ligne de commande. Ce service est conçu afin d'être déployé en grande quantité. Présentement les fichiers de docker-compose incluent trois instances de ce service. Afin de modifier le nombre d'instances de ce service, il faut modifier les configuration du HAProxy et les fichiers docker-compose. Le fichier de configuration du HAProxy se trouve dans la racine du projet et il est passé au conteneur en utilisant docker-compose.

Le certificat SSL

Comme indiqué dans le diagramme de déploiement, l'application utilise une connexion HTTPS afin de communiquer avec le navigateur de l'utilisateur. Le certificat SSL utilisé pour l'instant est de type self-signed pour des raisons de développement. Il est recommandé au temps de déploiement sur un site opérationnel d'avoir un DNS et un certificat SSL signé. Un outil qui peut être utilisé afin de générer le certificat SSL signé est l'outil certbot de la compagnie Let's Encrypt [2].

Afin de générer un nouveau certificat SSL self-signed, l'utilisateur peut utiliser la commande suivante dans la racine du projet.

```
(echo CA & echo QC & echo M & echo company & echo section & echo
someone & echo happy@example.com) | openssl req -x509 -nodes -days
365 -newkey rsa:2048 -keyout ssl/private/nginx-selfsigned.key
-out ssl/certs/nginx-selfsigned.crt
```

Il est aussi possible de modifier le défi Heilmann utilisé afin d'authentifier le client avec la commande:

```
openssl dhparam -out nginx/dhparam.pem 4096
```

Références

[1] https://docs.google.com/document/d/1n8zDzQv9TPS-QddhGwwJ-BVU7RXKsaWnZBXEOSwuHYY/edit

[2] https://letsencrypt.org/getting-started/