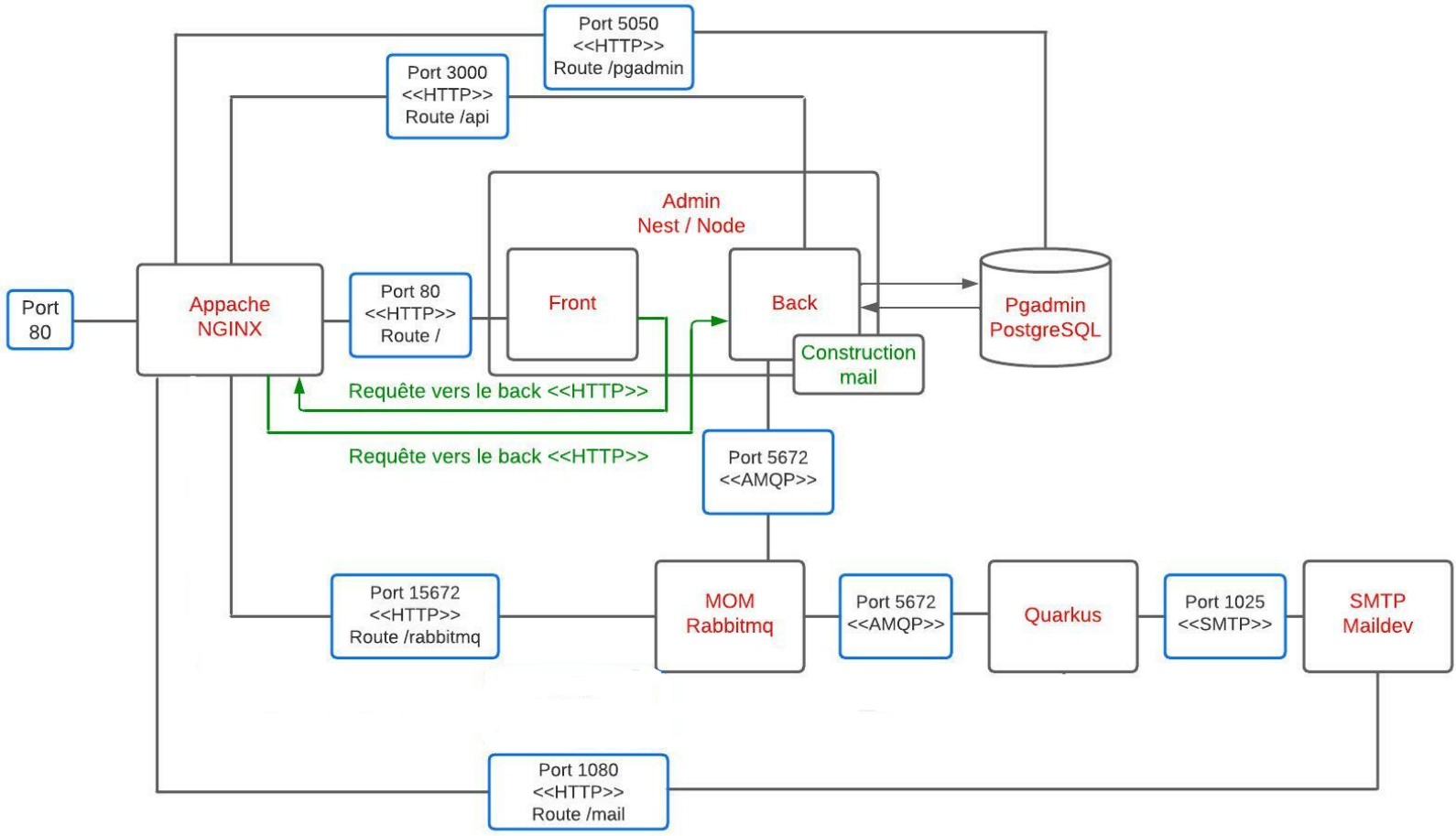
Yanis Bouger
Marius Le Douarin

Projet d'Architecture Logicielle

Architecture du projet

Schéma des différents liens entre les microservices et quelques redirections vers le reverse proxy



L'architecture en détail

Tout d'abord, il est bon de savoir que nous avons intégré du reverse proxy à notre projet, grâce à NGINX qui est un serveur web et connu pour le fait que ce soit un reverse proxy open-source, pour plusieurs raisons : tout d'abord d'un point de vue sécurité le reverse proxy peut faire office de pare-feu, en bloquant tout genre de requêtes malveillantes avant qu'elles ne puissent atteindre le serveur cible, le reverse proxy gère également les certificats SSL, pour aussi une question d'anonymisation puisque cela permet de masquer l'identité des serveurs cibles pour des raisons de confidentialité, c'est aussi pratique pour sa gestion des URLs puisqu'il peut gérer les redirection et les ré-écriture d'URL pour un serveur cible.

Ici, NGINX va servir de pièce maîtresse dans notre projet car il va permettre de tout superviser : il redirige le chemin /pgadmin/ vers le serveur pgadmin, le chemin /api vers le back, le chemin /rabbitmq vers l'interface rabbitmq, le chemin /mail vers maildev, et la route par défaut vers le front.

Ce dernier a besoin d'effectuer des requêtes au back, comme nous avons dû l'effectuer pour notre projet de Web, ici le front passe par le reverse proxy pour parvenir à ses fins, encore une fois pour les mêmes raisons d'un point de vue sécurité précisées précédemment. Tandis que le back lui peut aisément effectuer ses requêtes à la base de données donc nous avons fait le choix de ne pas avoir à repasser par NGINX pour une même requête.

Pour le système de mail, nous nous basons sur un protocole AMQP de messagerie ouvert afin de permettre à notre application de communiquer de manière asynchrone avec Quarkus en utilisant des fils d'attentes de messages. Ici nous utilisons alors RabbitMQ qui est une implémentation open-source d'un de ces serveurs de messagerie AMQP, fiable, efficace et scalable au besoin, et puisque c'est du AMQP le back comique directement avec tout comme Quarkus. Quarkus qui est dont un framework Java open-source qui va nous être très utile ici puisqu'il s'agit de notre interface permettant d'établir une connexion directement avec Maildev en SMTP, le protocole de communication permettant la transmission de mails comme il est demandé dans le projet.

Enfin, nous utilisons pgAdmin qui est un logiciel open-source de gestion de base de données pour PostgreSQL, notamment pour configurer, gérer (il permet la création, la modification et la suppression d'utilisateurs, et gère ces données avec des requêtes SQL) et maintenir notre base de données PostgreSQL. pgAdmin ne sera également pas dépendant de passer par nginx pour ses requêtes à la base de données.

Liste et statut des services

Nous avons implémenté les services suivants :

- Un back-end NestJS
- Un front-end Angular
- Une base de donnée PostgreSQL
- Un adminer pgAdmin
- Un reverse-proxy NGINX
- Un MOM RabbitMQ
- Un service d'envoie de mails sous Quarkus
- Un client mail Maildev

Aucun de ces services n'a été testé ni n'est monitoré, de plus nos mails ne contiennent pas d'iCalendar.

Les options de configuration de notre application

Il n'y a pas beaucoup de paramètres qu'il est possible de configurer sans que cela ne pose problème à notre application. Il est néanmoins possible de changerles paramètres suivants dans les fichiers docker-compose :

- PGADMIN_DEFAULT_EMAIL qui vaut par défaut admin@admin.com
- PGADMIN_DEFAULT_PASSWORD qui vaut par défaut admin
- MAILDEV_WEB_USER qui vaut par défaut maildev
- MAILDEV_WEB_PASS qui vaut par défaut maildev

L'existence d'un mode de production et d'un mode de développement, les différences

L'intérêt ?

Il y a un grand intérêt de distinguer une version production d'une version développement lorsqu'il s'agit d'une telle application Web. Tout d'abord d'un point de vue sécurité, les paramètres de sécurité sont plus stricts en production, puisqu'elle sera mise entre les mains du grand public il s'agit donc de protéger les données sensibles des utilisateurs, comme ici les mots de passe par exemple.

On cherche également de bonnes performances en production ainsi qu'une bonne optimisation afin de garantir une bonne expérience utilisateur, qu'elle soit fluide et sans délai, tandis qu'en développement nous pouvons nous permettre que les délais soient plus longs en développement, bien que cela ralentisse également les développeurs, mais nécessaire pour certaines étapes de débug ou test. La version pour le développement reste tout de même une solution facile pour les développeurs, pour s'assurer de pouvoir travailler efficacement, assurer une maintenance aisée sans perturber les utilisateurs en sortie.

En production

Nous faisons usage de dockers temporaires pour build le front, le back ainsi que Quarkus, qui s'enchaînent avec un node qui contient juste le dist et les nodes modules pour le back, un nginx qui contient que le dist pour le front, et un Quarkus qui continent que l'application Quarkus compilée en natif pour Graalvm, c'est-à-dire l'environnement qui permet d'exécuter notre application avec une performance élevée, une faible empreinte mémoire ainsi qu'une utilisation efficace des ressources. Dans le mode production notre front est sur le port 80, qui est d'ailleurs le seul port exposé du reverse proxy.

En développement

Dans ce mode-ci nous faisons usage de volumes pour utiliser le code présent directement sur l'ordinateur, de telle sorte à pouvoir lancer en watch mode, c'est-à-dire qu'on

recompile lorsqu'un changement dans un fichier a été effectué, sans refaire pour autant l'image, que ce soit pour le front, le back et Quarkus. Cette fois-ci, le front est sur le port 4200, et il y a beaucoup de ports qui sont exposés, mais c'est logique puisque nous sommes en mode développement et que la nécessité de tester les microservices un à un en cas de problème s'importe.

De potentiels feedbacks pour l'unité d'enseignement Architecture Logicielle

Cette unité d'enseignement était enrichissante, tant sur le plan acquisition de connaissances pour plus tard que sur le plan pratique, notamment vis-à-vis de l'apprentissage de la dockerisation d'un projet à plus ou moins grande échelle, ou bien encore de la gestion de microservices et les interconnexions, la mise en place de différentes "versions" d'une application vouée au déploiement, pour faciliter dans un cas le travail des développeurs, dans l'autre d'assurer une agréable expérience utilisateur.

Peut-être qu'il faudrait néanmoins que le projet d'AL devrait être moins dépendant du projet de l'unité d'enseignement Web Mobile, bien qu'il soit possible d'avancer l'un sans l'autre, j'imagine que des groupes moins avancés au cours du projet ont dû se retrouver dans une impasse, notamment pour l'intégration du front, voire du back. Une conséquence étroitement liée est les deadlines un peu trop proches, bien qu'une faveur nous ait été faite en l'occurrence ici. Il ne s'agit ici que d'une remarque mineure qui n'enlève en rien l'utilité du module et la découverte de ces nouveaux outils importants pour l'intégration et le développement d'applications.

Le lancement de l'application

Pour lancer le mode dev et prod il suffit de rentrer les commandes :

- npm install pour le back et le front
- tar xvf volumes.tar.gz à la racine
- sudo chown -R 5050:5050 volumes/pgadmin à la racine
- docker-compose -f docker-compose.prod.yml up –build –force-recreate ou bien docker-compose -f docker-compose.dev.yml up –build –force-recreate à la racine