Yanis Bouger
Marius Le Douarin

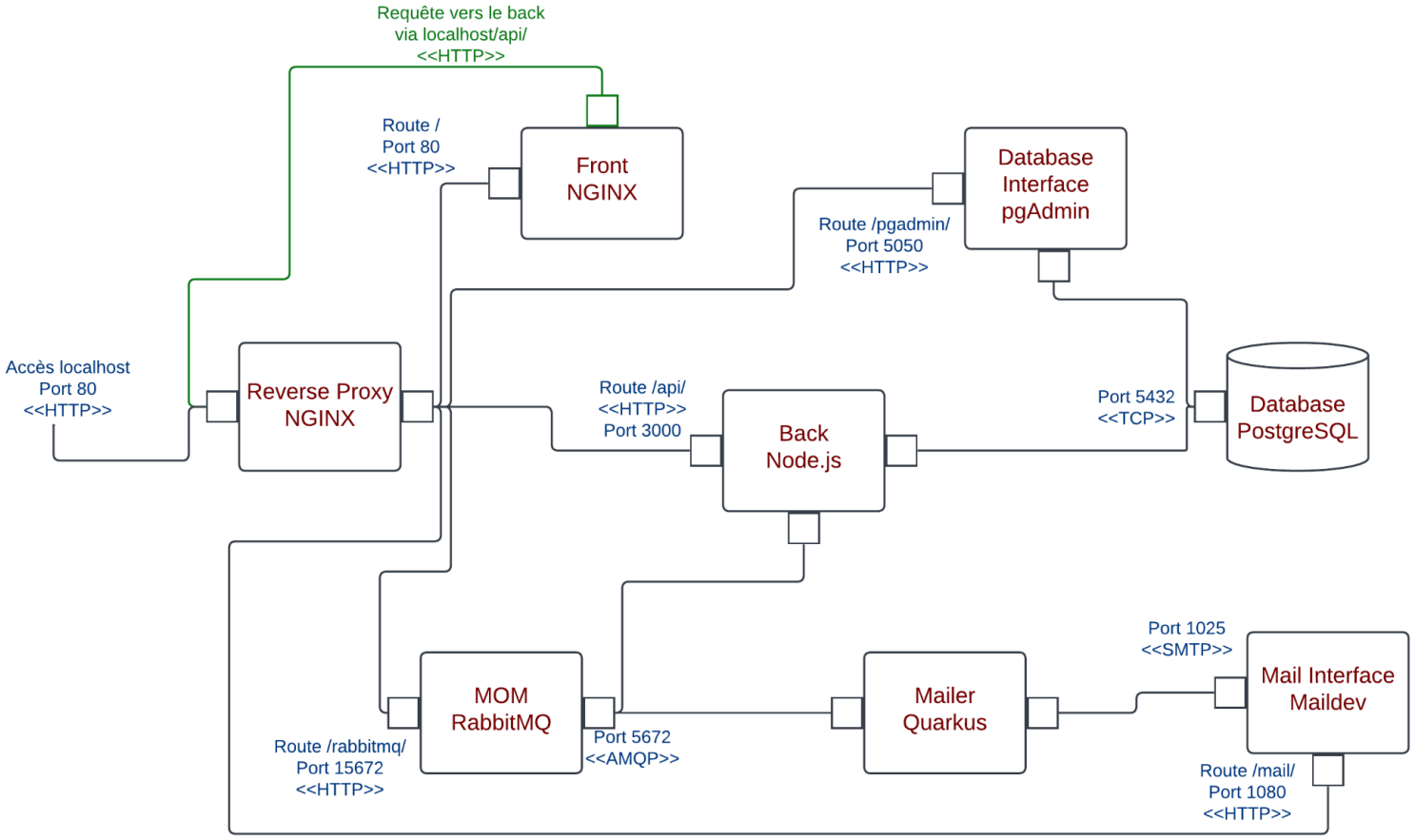
Université de Rennes

Projet d'Architecture Logicielle

Dernier commit fait le dimanche 15 janvier 2023 à 19h01 UTC+1

Architecture du projet

Schéma des différents liens entre les microservices en mode production.



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 la route /pgadmin/ vers le serveur pgAdmin, la route /api/ vers le back, la route /rabbitmq/ vers l'interface RabbitMQ, la route /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

Nous n'avons pas réalisé de load-testing, ni de monitoring. De plus, nos mails ne contiennent pas de pièces jointes sous la forme de fichiers .ics.

Les options de configuration de notre application

Il est possible de configurer les identifiants pour les différents services de notre application. Cela se fait dans le fichier .env à la racine du projet.

Pour RabbitMQ, RMQ_USER et RMQ_PASS servent à la fois au back et à Quarkus pour accéder au MOM, ainsi qu'à l'utilisateur s'il souhaite accéder à la page de management de RabbitMQ (route /rabbitmq/). RMQ_VHOST sert à indiquer quel hôte virtuel utiliser.

Pour Maildev, MD_IN_USER et MD_IN_PASS servent à Quarkus pour pouvoir envoyer les mails, MD_WEB_USER et MD_WEB_PASS servent à l'utilisateur pour accéder à la page web de visualisation des mails (route /mail/).

Pour pgadmin, PGADMIN_EMAIL et PGADMIN_PASS servent à accéder à l'interface web (route /pgadmin/).

Enfin, pour la base de données postgres, PG_USER et PG_PASS servent au back pour y accéder, de plus la valeur de PG_USER doit être reportée dans le fichier servers.json du dossier pgadmin, et PG_PASS est nécessaire depuis pgadmin pour accéder à la base de données.

Leurs valeurs par défaut sont les suivantes :

maildev	
MD_IN_USER	quarkus
MD_IN_PASS	quarkus
MD_WEB_USER	maildev
MD_WEB_PASS	maildev

rabbitmq	
RMQ_USER	guest
RMQ_PASS	guest
RMQ_VHOST	mail

pgadmin	
PGADMIN_EMAIL	admin@admin.com
PGADMIN_PASS	admin

postgres	
PG_USER	postgres
PG_PASS	postgres

Utiliser notre application

Pour lancer notre application il faut se placer à la racine du projet, et lancer la commande `docker-compose -f docker-compose.dev.yml up --build` pour lancer en mode dev, ou `docker-compose -f docker-compose.prod.yml up --build` pour lancer en mode prod.

Une fois les conteneurs lancés, nous pouvons accéder à http://localhost/ pour accéder au front-end. Depuis cette page, il est possible de se connecter, pour cela deux identifiants sont déjà disponibles : nom d'utilisateur 1, mot de passe azertyuiop ; nom d'utilisateur 2, mot de passe qwertyuiop. Sinon il est possible de créer un nouvel utilisateur, une fenêtre pop-up donnera l'identifiant servant de nom d'utilisateur. Une fois connecté, dans le menu de navigation, il est possible d'accéder à la liste des associations, puis à la

page d'information d'une association, depuis laquelle il est possible d'envoyer un mail à ses membres.

Il est possible d'accéder à l'interface Swagger sur http://localhost/api/swagger/, nous avons laissé cette possibilité, car c'est la seule façon que nous avons de créer une association ou un rôle.

Il est possible d'accéder à l'interface pgadmin sur http://localhost/pgadmin/, les identifiants par défaut sont : mail admin@admin.com, mot de passe admin. La base de donnée devrait déjà être présente dans le menu déroulant de la liste des serveurs en haut à gauche, pour y accéder le mot de passe par défaut est : postgres. Les tables sont présentes dans le menu Databases → postgres → Schemas → public → Tables. On peut ensuite accéder au contenu d'une table en faisant : clique-droit → View/Edit Data → All Rows.

Il est possible d'accéder à l'interface de management de RabbitMQ sur http://localhost/rabbitmq/, les identifiants par défaut sont : nom d'utilisateur guest, mot de passe guest.

Enfin, il est possible d'accéder à l'interface web de Maildev sur http://localhost/mail/, les identifiants par défaut sont : nom d'utilisateur maildev, mot de passe maildev.

Pour arrêter l'application, il suffit de faire ctrl+c dans le terminal qui a servi à la lancer, ou `docker-compose -f docker-compose.dev.yml down`, en mode dev, ou `docker-compose -f docker-compose.prod.yml down`, en mode prod. En mode dev, le « Live Reload » de Quarkus peut créer un conteneur supplémentaire qui ne se supprime pas tout seul.

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 uniquement le dist et les nodes modules pour le back, un nginx qui ne contient que le dist pour le front, et un Quarkus qui ne contient 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 ou Quarkus (pour Quarkus, une interaction telle qu'un envoie de mail peut être nécessaire pour activer le Live Reload). Cette fois-ci, le front est sur le port 4200, et il y a beaucoup de ports qui sont exposés, mais c'est normal puisque nous sommes en mode développement et que la nécessité de tester les microservices un à un en cas de problème s'importe.

Ce que nous avons fait pour sécuriser le mode production

Afin d'avoir le minimum possible de ports d'exposé en mode production, nous avons utilisé avidement le reverse-proxy et créé autant de routes que nous avions de service auxquels accéder. De plus, nous avons utilisé le maximum possible d'images de type « alpine » qui ont été pensées pour êtres plus sécurisées. Nous avons aussi utilisé des utilisateurs non-root dans les conteneurs, pour prévenir d'éventuelles failles. Enfin nos services nécessitent pour la majorité des identifiants pour y accéder ou communiquer avec eux.

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 faudrait-il néanmoins que le projet d'AL soit moins dépendant du projet de WM, bien qu'il soit possible d'avancer l'un sans l'autre, nous pensons 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.