Projet SQUARE

Documentation Technique

Mélissa DA COSTA
&
Judicaël KOFFI

Table des matières

Introduction

Le projet square permet de déployer et gérer des applications vivant dans des conteneurs Docker.

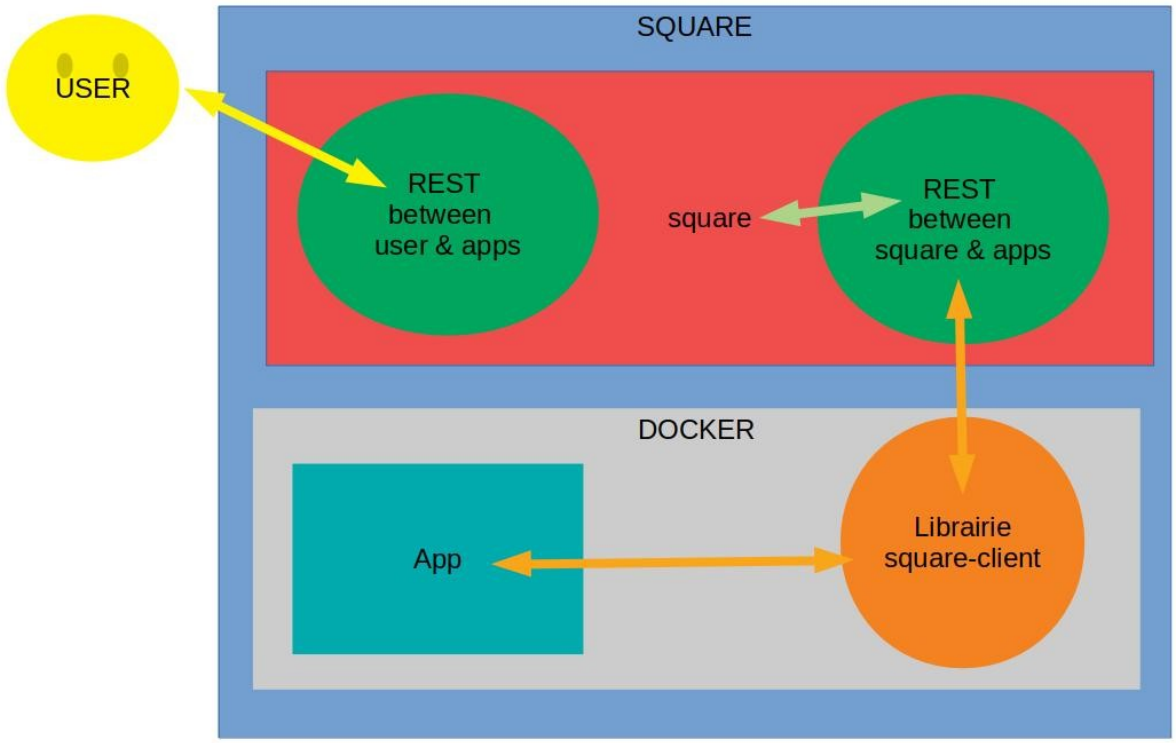
Docker est un outil permettant de lancer des applications dans des conteneurs. L'avantage est la portabilité des applications conteneurisées. Celles-ci ont déjà un socle de dépendance dans le conteneur et peuvent donc être exécutées sur n'importe qu'elle machine.

Le projet square est composé de deux entités :

Une application serveur square contenant des services REST. Parmi ces services REST, il y a des services REST permettant de contrôler les applications déployées par un utilisateur. Ainsi que des services REST permettant la discussion entre `square` et les applications déployées.

Une librairie cliente square-client qui est ajoutée dans chaque conteneur. Elle est accessible par chaque application déployée et peut discuter avec les services REST de discussion entre `square` et les applications déployées.

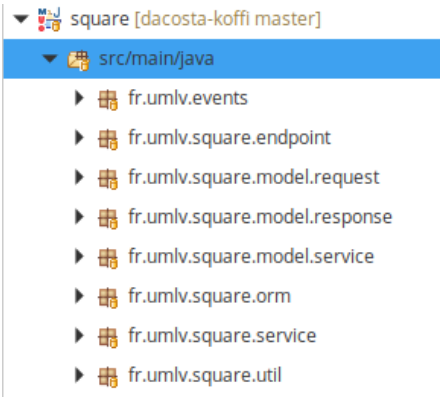
Voici un schéma du fonctionnement du projet square :



Choix d’architecture

Square

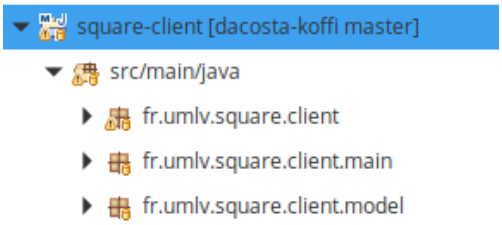
Nous avons décomposé le serveur square en plusieurs package :



- **event** contient la classe AppLifecycleBean. Cette nous permet de contrôler les actions effectuées au démarrage de l’application et à son arrêt grâce aux méthodes onStart et onStop.
- **endpoint** contient les classes correspondant aux différentes routes de l’API REST créée.
- **request** contient des classes correspondant aux requêtes envoyées aux routes de l’API. Ces requêtes sont sous forme d’objet JSON et doivent respecter certaines normes pour être reconnu.
- **response** contient des classes correspondant aux réponses renvoyées après un appel aux routes de l’API. Ces réponses sont aussi sous forme d’objet JSON.
- **service** contient la classe ImageInfo qui correspond à une image docker. Elle stocke toutes les informations récupérable via la commande *docker ps*. Ce package contient aussi la classe ReceivedLogModel qui correspond à un log envoyé par square-client.
- **orm** pour Object Relational Mapping correspond à une interface entre l’application square et une base de données. La base de données créée sera expliqué plus en détail dans une prochaine partie. Ce package contient la classe LogTable correspondant à notre table dans la base de données et la classe LogEntity correspondant aux tuples présent dans la table LogTable.
- **service** contient des classes qui permettent de gérer la partie principal de l’application. Comme la classe DockerService qui gère toutes les interactions avec les container docker. Ces classes « service » sont utilisées comme des interfaces entre l’utilisateur de la classe et l’intérieur du fonctionnement de l’application.
- **util** regroupe des classes dont les champs sont utilisés dans plusieurs classes. Ce package nous permet de factoriser le code

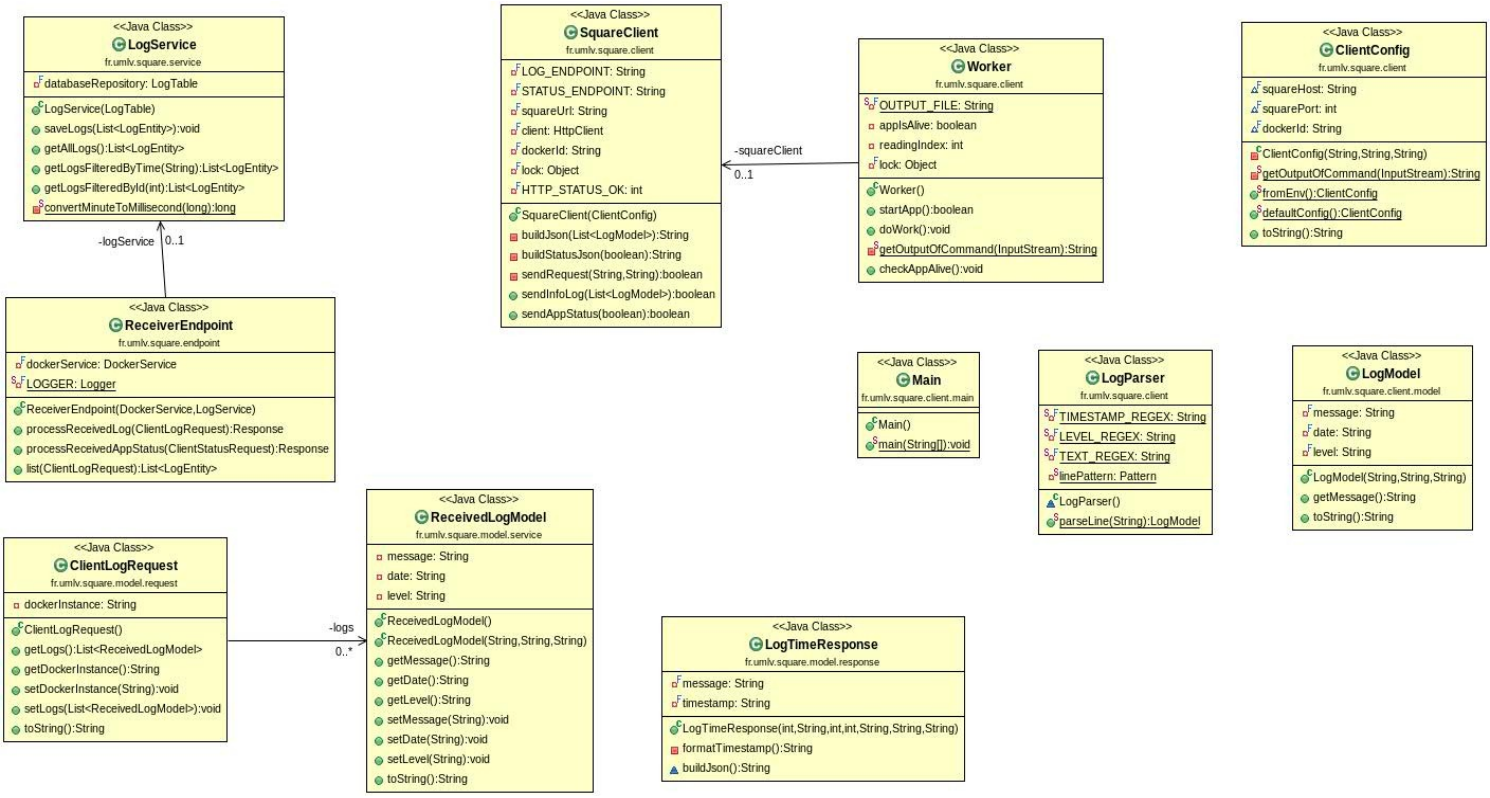
Square Client

Nous avons aussi décomposé la librairie square-client en plusieurs package :



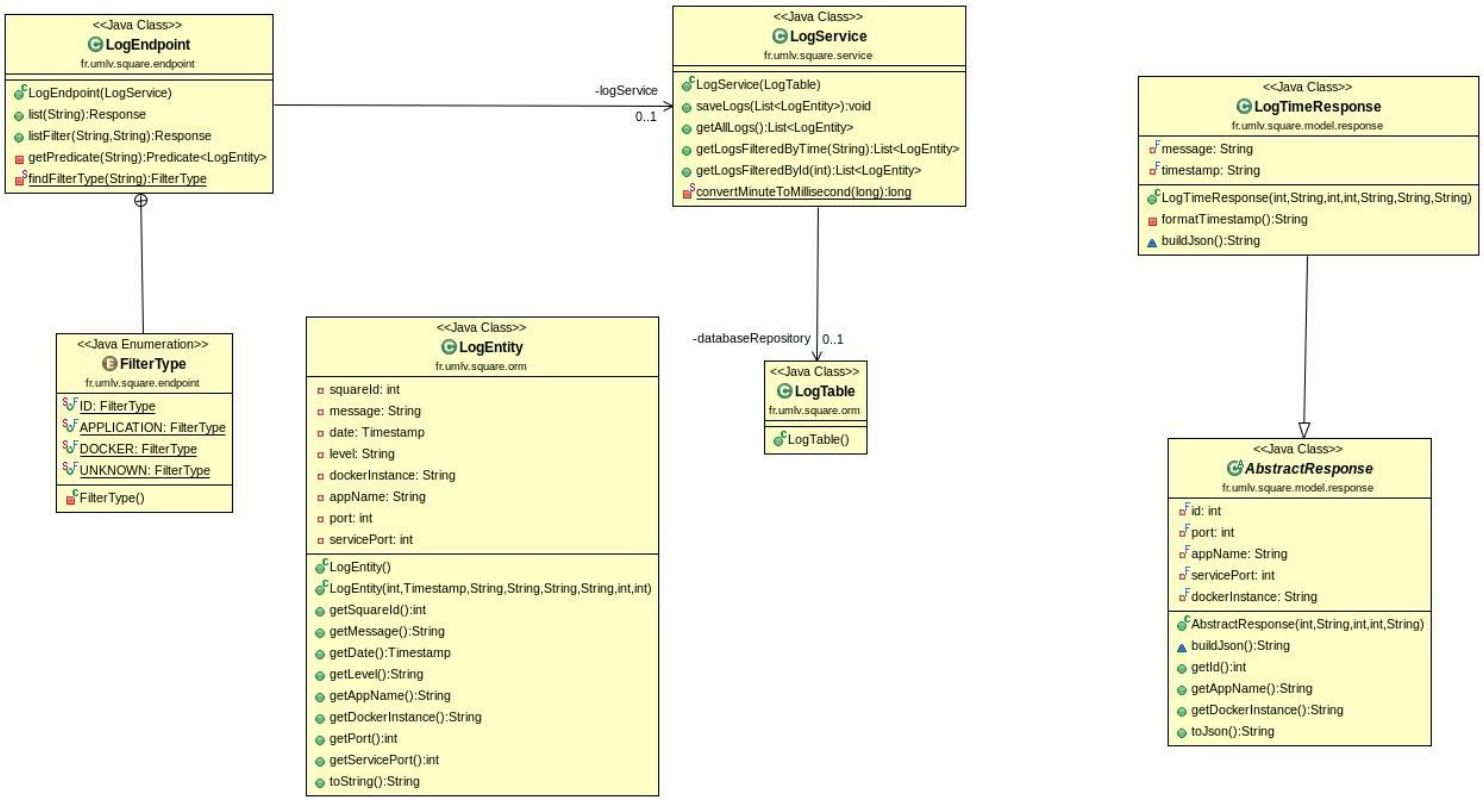
- **client** contient les classes qui forment l'application square-client.
- **main** contient la classe Main de square-client. Elle lance des Threads qui envoient les logs des application au serveur square.
- **model** contient la classe LogModel qui représente un log à envoyer au serveur square.

Voici un diagramme de classe de square-client :



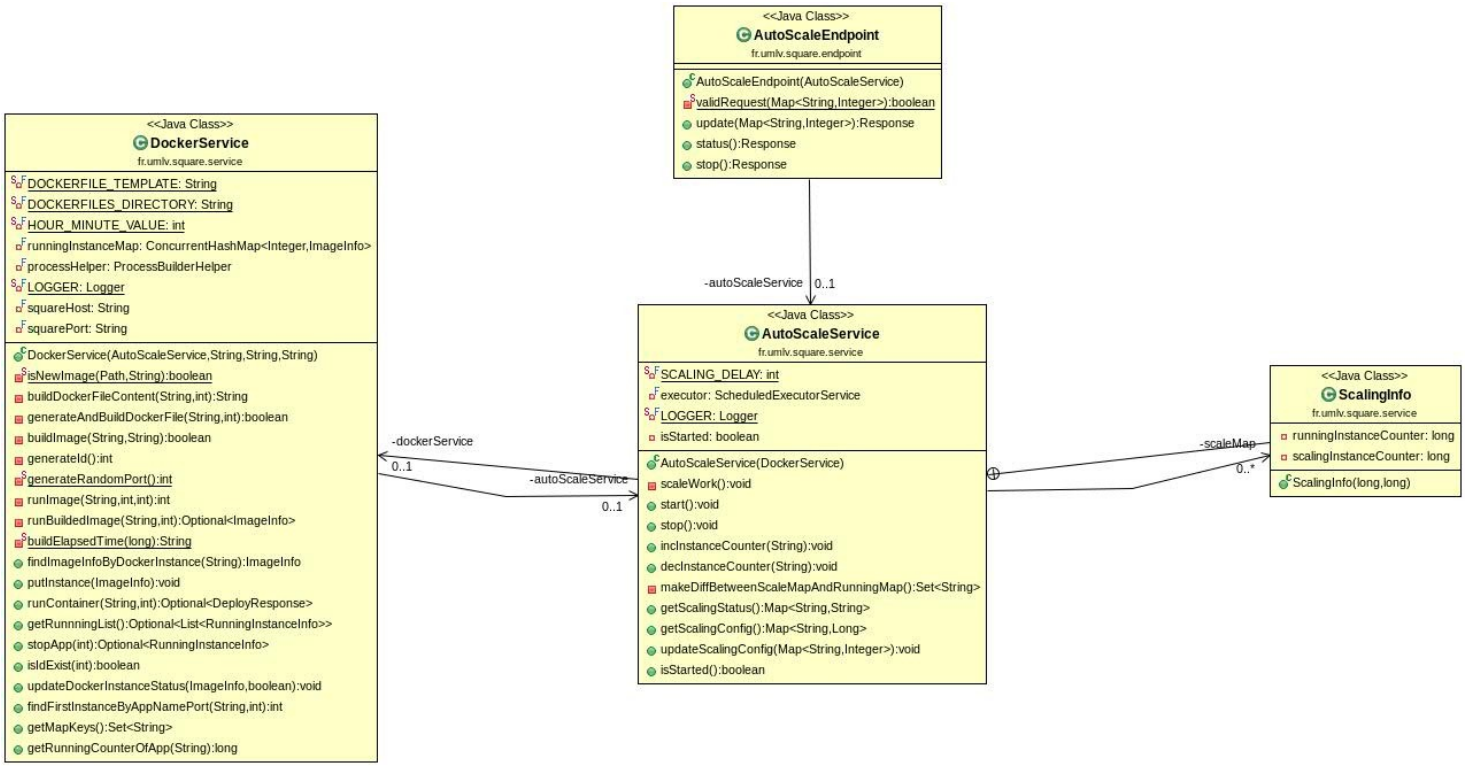
Gestion des logs

Voici un diagramme de classe représentant la gestion des logs :



Auto Scale

Voici un diagramme de classe représentant l'auto scale :



Dans le développement de ce projet, certaines décisions ont été prises notamment pour faciliter le travail. Par exemple, nous avons choisi que les images docker aient le même nom que le jar qui leur correspond.
Nous avons aussi décidé qu'au démarrage de Square, nous lançons un *docker ps* pour connaître les images qui tournent afin de reconstituer la structure de données interne de notre service qui manage les dockers. On a remarqué qu'il y a une limitation à cela, s'il y a d'autres docker lancer sur autre que ceux lancer par Square, il y a des infos qui sont fausses, une solution est de rajouter un tag sur les noms des images lancer par Square afin de filtrer par rapport qu'aux images ayant ce tag mais ce n'est pas forcément la méthode la plus propre.

Utilisation d'une base de données:

Nous avons utilisé une base de données postgres sql pour faire persister les logs des applications déployer dans les dockers.

Voici les points que nous n'avons pas réaliser. Nous avons fais le choix de ne pas prioriser ces points pour assurer la qualité du projet :

Persistance de l'auto-scale :

Lors de l'arrêt de l'application square, si par exemple il y avait une configuration d'auto-scale active alors elle est perdu. Au redémarrage de square, l'ancien configuration d'auto-scale n'est pas ré-appliqué.