LocalKube

Ruben SERO / Mohamed YAHYAOUI

Répartition du projet :

Le projet est divisé en deux parties principales :

- Gestion des applications : se fait par le package “app”, dont la responsabilité est de générer une image docker, lancer et arrêter un conteneur et lister l’ensembles des instances
- Gestion des logs : se fait par le package “log”, il s’occupe d’insérer les logs de l’api cliente dans la base de données

Caractéristiques techniques:

Au lancement de LocalKube, on recrée la liste des applications présentent dans le docker.

L’arrêt du localKube n’a pas d’impacte sur le fonctionnement des conteneurs .

Le nom de chaque conteneur est composé par l’id de conteneur, le nom de l’image et le numéro d’instance, par exemple “2-toto_1” est le conteneur généré à partir d’une image toto, qui a comme id d’application 2 et numéro d’instance 1.

On peut modifier la configurations interne de projet en modifiant le fichier “application.properties”, cela assure une meilleure flexibilité pour le déploiement de localKube.

La couche du base des conteneurs utilise “openjdk:15-alpine”, (la documentation dans dockerHub). Comme cette image est basée sur la distribution Alpine Linux, notre image docker aura une taille minimale (documentation de l’image Alpine).

Chaque docker généré par localKube est ouvert sur 2 ports:

- Un port service : qui permet la communication entre l’application localKube et la lib Cliente (présente à l’intérieure du conteneur)
- Un port client : qui permet la communication entre l’application d’utilisateur.

Si l’application utilisateur a besoin d’un port, elle peut utiliser le port 8080 à l'intérieure du conteneur.

Gestion des applications :

Première étape:

L’application LocalKube permet un monitoring sur les conteneurs docker en cours d'exécution. L’application connaît donc l’etat des différents conteneur sur le docker (Up, Down, Exited ..).

Il est donc nécessaire de recréer tous les applications qui ont était lancées séparament par une commande docker, par l’utilisateur, ou par localKube aprés une redémarrage de l’application.

On considére que tout les conteneurs lancés dans ce contexte respectent le format d'un conteneur lancé par localKube (contient l'api cliente, ouvert sur un port service et un port client, le nom de conteneur respecte le format "conteneur ID_image- Num instance" …).

Un exemple d'un conteneur lancé par LocalKube qui contient l'api Cliente lancé sur le port service 15001, et l'application utilisateur "timerticks" sur le port 8080 :

IMAGE : timerticks
ENTRY POINT : java -Dserver.port=15001 -jar local-kube-api.jar
PORTS : 0.0.0.0:15001→15001/tcp
0.0.0.0:8092→8080/tcp
NAMES : 1_timerticks-1

Gestion des endPoints : Applications

Le package "app" contient un end-Point "ApplicationController" ouvert à trois types de requêtes (Start, Stop et List).

une requête de type Start accepte les données de l'application à lancer, ces informations suivent le modéle de départ définit par le record "ModelStart".

On commence par sérialiser les données de l'application envoyer sous forme d'objet "ModelStart" en utilisant le parsing de Jackson 2.11.3 et les annotations "@JsonProperty" et "@JsonCreator".

La classe Application Handler applique la process de la création et de lancement d'un conteneur docker qui contient l'api cliente et l'application envoyé par l'utilisateur.

Pour créer un conteneur docker, on utilise la méthode "generateJar" de la classe "ImageFactory", dont la responsabilité est de créer l'image docker.

On utilise la librairie jib core 0.16.0 pour générer l'image docker sous forme de fichier tar, un fois l'image a été générée, on peut lancer le conteneur en deux étapes :

1- Charger l'image sur docker : par la méthode "loadImage" de la classe "ApplicationLoader"

2- Lancer le conteneur : par la méthode "LaunchApp" de la classe "ApplicationLauncher"

Chaque conteneur contient une Api-cliente qui s'agit d'un spring boot avec un seule end-Point, l'api cliente nous permet de :

- Lancer l'application utilisateur à l'intérieure de docker
- Récupérer le output de l'application et l'envoyer au localKube
- Signaler a localKube que l'application s'est arretée à cause d'une erreur

- Libérer les ressources et arrêter l'application avant d'arretêr le conteneur en cas de request shutDown de la part de localKube

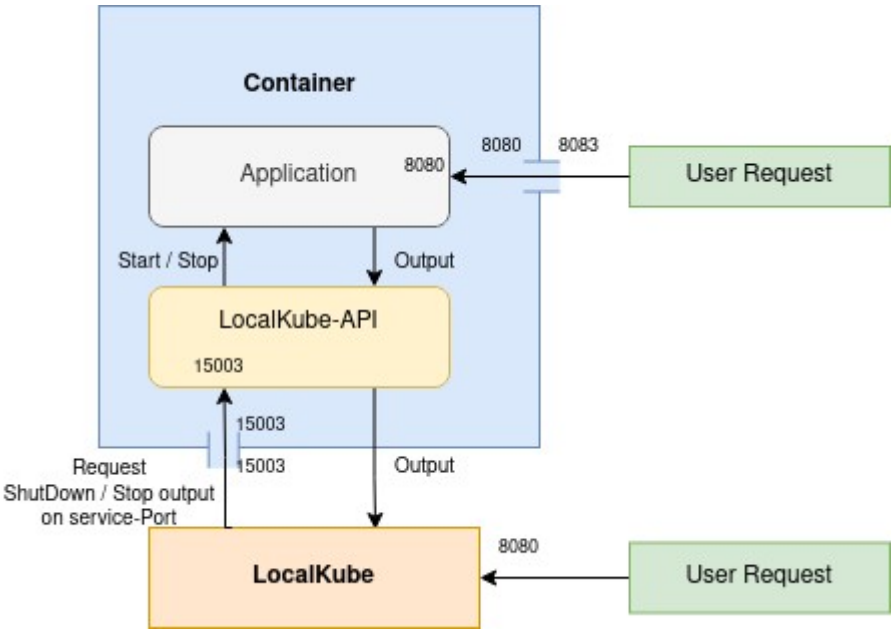

Schéma de comunication entre LocalKube et le conteneur

Gestion des endPoints : Logs

Les logs sont sauvegardés à deux endroits. Une première sauvegarde est effectués dans une base de données Sqlite JDBC 3.32.3.2 à l'aide de la librairie jdbi 3.16.0. Une deuxième sauvegarde est effectuée dans un fichier csv permettant de retrouver les logs même si l'application a été arrêté.

- Sauvegarde dans la base de donnée

La classe permettant le traitement de la base de donnée (DataBase) est implémenté par l'interface LogObserver car la base de données est créé lors de la création de l'observeur et des éléments y sont ajoutés lors des passages dans le thread. Les champs de la base de donnée sont l'id de l'application, le timestamp du log, ainsi que son message. A l'aide de la map, il sera possible de retrouver toutes les informations de l'application émettant le log vu que l'on connaît son id qui est unique. La clé primaire de la base sera un enum vu qu'une application peut emmètre plusieurs log, donc son id n'est pas unique.

- Sauvegarde dans le fichier csv

La classe permettant la lecture et l'écriture du fichier csv (CsvFile) est implémenté par l'interface LogObserver car ses méthodes doivent avoir lieu à la création et à la fermeture de l'observeur.

Un fichier csv contenant les logs sera créé à chaque nouvelle heure. Dans chacun de ses fichiers, on stocke l'id de l'application émettant le log, le timestamp et le message. Si l'application redémarre, l'écriture se fait à la suite du fichier de log si on est toujours dans la même heure.

Lors du redémarrage de l'application, tout les fichiers contenus dans le dossiers /logs seront lu et ajouté dans la base à l'aide de la méthode openFileLog qui parcours tout les fichiers csv.

- Filtrage des logs dans la base donnée

Comme tout les logs sont dans une base de données, il va être possible de transformer la base de donnée sous forme de map de record afin de lui appliquer la méthode filter de map. Dans ces records, toutes les informations de l'application seront retenu. De cette façon, les fonctions appelées suite aux appels des requêtes JSON appellent la méthode filterLog contenant le bon predicat en paramètre. Ainsi, elle renvoie une liste de tout les records étant vrai pour le prédicat.