ECOLE D'INGÉNIEUR ENSEEIHT

Cloud et Big Data - Rapport de projet

Florent DUBOIS
Florian GARIBAL
Luc SAPIN
Omar BENNANI

18 décembre 2017

Sommaire

1 Projet

La technologie Docker

Dans le cadre du projet de Cloud et Big Data, nous avons dû développer plusieurs technologies de manière distribuées. Nous devions donc les configurer correctement pour qu'elles communiquent entre elles.
Tout d'abord, nous avons dû décider sur quel support de virtualisation installer nos outils (serveur zookeeper, kafka, cluster Spark, ElasticSearch, Kibana).
Nous avons décidé d'utiliser la **technologie Docker** pour plusieurs raisons : tout d'abord le **hub Docker héberge nos images** en ligne (très pratique pour la gestion du projet) ; de plus, la technologie docker est **très dynamique** et la **gestion de nos images** est très **facile** et **intuitive** ; enfin, cette technologie nous permet de mettre en place des **containers** (i.e. des machines virtuelles) **très légers** avec **seulement les outils nécessaires au développement des services voulus**. En effet nous n'avions pas besoin d'une interface graphique ou de tout autres services offert par une machine virtuelle autre que le gestionnaire de paquet et l'architecture Linux.

Il a donc d'abord fallu se familiariser avec la technologie Docker, le **plus délicat** étant comment réussir à **lier des containers entre eux**, et faire qu'ils communiquent efficacement. Nous avons créé une **image pour chaque outil**, ainsi qu'un **fichier *.sh*** pour **lancer chaque container avec les bons paramètres**. Pour **lier les container** entre eux, la technique utilisée consiste à **renseigner un port d'écoute** au lancement d'un container (pareil qu'en séance de TP) et de lier ce même port avec les autres containers.

Cependant, après avoir effectué des recherches sur cette manière de faire communiquer les containers nous avons remarqué qu'elle était dépréciée. En effet l'utilisation de *–link* n'est plus conseillée et a été remplacée par l'utilisation des *network*. Ce nouveau moyen de fonctionner permet de créer un réseau dans lequel on lancera nos containers qui seront par la suite tous accessibles entre eux par le biais de leur IP ou de leurs noms grâce à un serveur DNS. Nous n'avons cependant pas eu le temps de mettre en place cette solution.
La première difficulté a été de spécifier aux containers de Zookeeper et de Kafka comment ils devaient communiquer.

Zookeeper et Kafka Serveur

Pour cela nous avons tout d'abord **exposé** le port **2181** du container **Zookeeper** (option *–expose=2181* lors du lancement du container) pour pouvoir ensuite y **accéder** depuis le container **Kafka** avec l'option **link** (*–link zookeeper :2181*). Ce "link" nous a permis d'obtenir, dans le container Kafka, une variable d'environnement contenant l'IP de Zookeeper, dynamiquement allouée lors du démarrage du container, et le port exposé associé. Il nous a ensuite fallu modifier le fichier $KAFKA_HOME$/*config/server.properties* afin de lui donner à chaque démarrage l'adresse IP et le port sur lesquels se connecter. Pour cela nous avons utilisé le programme *sed* comme ici :
sed -i "112s#.#zookeeper.connect=$ADDR :$PORT#" ./config/server.properties*
L'étape suivante a été de mettre en place les producteur, consommateur et serveur Kafka.

Kafka Producteur, Consomateur et Serveur

Étant donné que le **producteur** était fourni par le **simulateur** et le **consommateur** était disponible sur le **container spark**, il nous a fallu préciser à chacun d'eux où relayer les données (pour le producteur) et où les récupérer (pour le consommateur). Dans un premier temps nous avons **lié** le **serveur Kafka** avec le **simulateur**. Pour cela nous avons fait la même manipulation d'exposition des ports et de lien entre les deux containers. Ensuite il a fallu rendre accessible l'application web de gestion des quartiers depuis l'extérieur. Pour cela nous avons décidé d'utiliser la l'option **-P <container_port> :<host _port>** avec les paramètres suivant *-P 8080 :8080* qui publie/transfère (-P pour *publish*) le port 8080 du container sur le port 8080 de la machine hôte. Suite à cela nous pouvions accéder à l'interface de gestion depuis l'adresse **http ://localhost :8080** depuis un navigateur sur notre machine hôte.

En créant un ***topic* nommé *conso-maison*** dans le server Kafka, le **producteur envoie les données** du simulateur dans les **brokers** stockées sur le serveur, sur les ports **9092** et **9093**. Le **consommateur**, lui, **se connecte et s'abonne au topic** *conso-maison* afin de récupérer les dernières données envoyées.
Ces données récupérées par le consommateur Kafka sont délivrées au cluster Spark. Pour configurer le consommateur, il a fallu modifier le fichier */sparkvip/src/ain/java/conf.java* de Spark, en récupérant l'adresse IP du server Kafka. Cette **adresse IP**, accessible comme **variable d'environnement grâce au link** avec le container Kafka, permet au consommateur (par le biais de la classe *System* de Java) d'accéder au *broker*qui contient le topic *conso-maison*.

Pour manipuler les données plus facilement, le container de Spark est lancé avec la commande *-v volume1 :/spark-vip*. On renseigne également les port **9092** et **9093** avec *–link*.

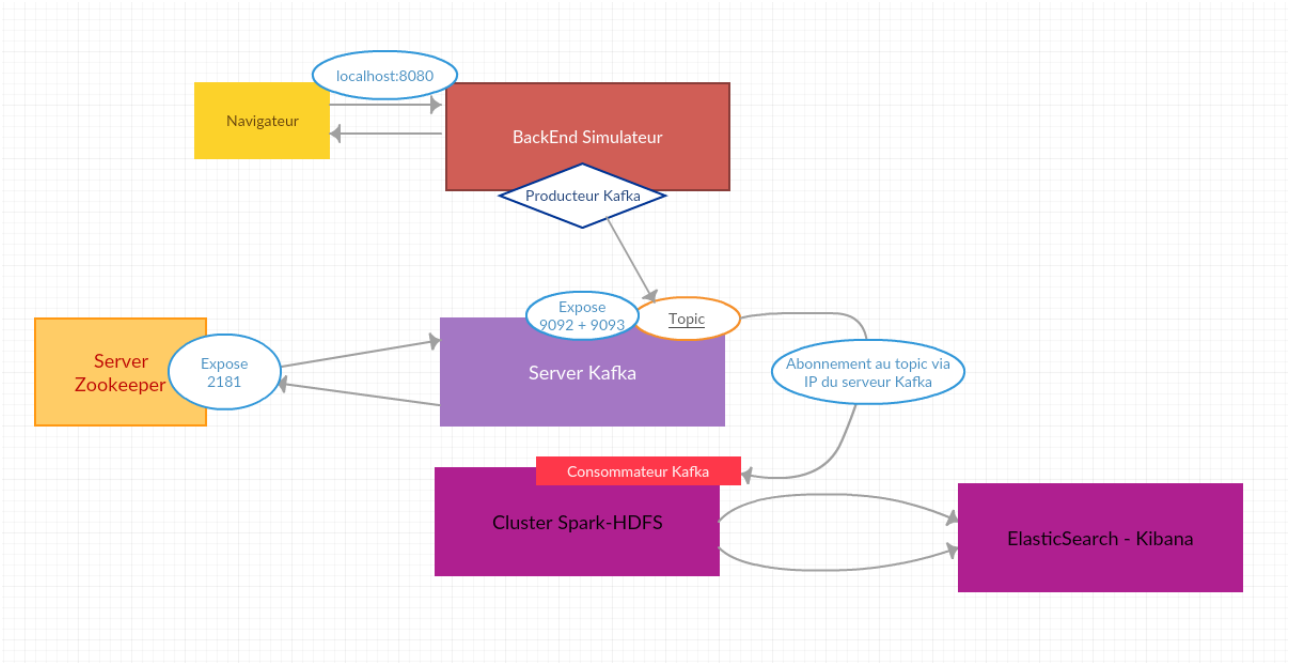

FIGURE 1 – Architecture : port d'écoute et abonnement

Par la suite, il a fallu configurer les outils de gestion de données comme HDFS puis ElasticSearch avec Kibana.

HDFS & ElasticSearch

La configuration du cluster spark-HDFS passe par l'installation d'hadoop dans le même container. Pour traiter les données, ElasticSearch vient les récupérer sur le cluster. Pour configurer HDFS, nous avons d'abord dû installer *ssh server* pour lui permettre de gérer les instances sur les différentes machines. Par la suite, nous avons suivi les sujets de TP en effectuant les mêmes manipulations d'hdfs. Par la suite nous avons créé un dossier */user/root* pour stocker les résultats des *Map-Reduce* dans */user/root/log/part-00000*. Pour que

2 Florent Dubois

Dans ce projet, j'ai dans un premier temps travaillé sur l'installation et la mise en place des premier dockers, que nous avons ensuite configurés en liaison avec Florian.
J'ai ensuite participé à la mise en place , à la liaison et à l'écriture du traitement du consommateur Kafka dans un nouveau docker qu'il a fallut lier dynamiquement. En effet le traitement qui ajoute un booléen signifiant si un quartier est vip ou non est effectué par Spark grâce au fichier conf.java qui précise également le chemin vers le docker du server Kafka. Nous avons rencontré quelques problèmes dans l'écriture de ce traitement, notamment sur la lecture du fichier txt contenant les quartiers vip ce qui nous a demandé du debuggage.
Enfin j'ai aidé Florian dans la mise en place et l'installation de HDFS.

3 Omar Bennani

Pour ma part, j'ai travaillé sur la mise en place de **Elasticsearch** et **Kibana**. Pour pouvoir écrire des données sur un index Elasticsearch, il m'a fallu écrire un **Job** en Java qui map les données stockées sur **HDFS** et qui les exporte (dans un format elasticsearch) dans ce que l'on appelle une **resource**. Tout ceci fut possible grâce aux différentes librairies de Hadoop (**mapred**, **conf**, **io**) et à la librairie **elasticsearch-hadoop-mr** qui nous fournit le format de sortie elasticsearch. J'ai ensuite suivi la procédure du TP Hadoop pour exporter le Job en jar embarquant les différents imports.
Ensuite, il m'a fallu démarrer les différents container (dont elasticsearch qui crée un node où seront indexées les données en sortie du Job) en exposant les ports et en les liant entre eux, et démarrer la simulation. Enfin, j'ajoutais elasticsearch-hadoop-mr-6.0.0.jar au classpath de hadoop : *export HADOOP_CLASSPATH=/elasticsearch-hadoop-6.0.0/dist/elasticsearch-hadoop-mr-6.0.0.jar*.
Cependant, en démarrant le Job, je tombais sur l'erreur suivante :

Exception in thread "main" org.elasticsearch.hadoop.EsHadoopIllegalArgumentException : Cannot detect ES version - typically this happens if the network/Elasticsearch cluster is not accessible or when targeting a WAN/Cloud instance without the proper setting 'es.nodes.wan.only'

Cela était dû à un problème de configuration de elasticsearch. La configuration des noms du node et du cluster se faisait via le fichier **elasticsearch.yml**. Le but était de les faire coincider avec ceux que j'ai fourni dans le Job, mais je n'arrivais pas à connecter la sortie du Job au node d'elasticsearch, et ce malgré avoir lié elasticsearch à spark/hdfs au bon port.

```java
@Override
public int run(String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), SimpleJob.class);

    job.setJobName("test-job");
    job.setInputFormat(TextInputFormat.class);
    job.setOutputFormat(EsOutputFormat.class);
    job.setMapperClass(Tokenizer.class);
    job.setMapOutputValueClass(MapWritable.class);
    job.setSpeculativeExecution(false);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    job.set("es.resource", args[1]);
    job.set("es.nodes", "172.17.0.6:9200");

    JobClient.runJob(job);
    return 0;
}
```

Malheureusement je n'ai pas eu le temps de creuser un peu plus, et on n'a donc pas pu afficher les données sur Kibana, malgré le travail en amont de Florent et Florian.

4 Luc Sapin

Dans ce projet, j'ai participé à l'élaboration des containers de Kafka et de Zookeeper. J'ai exploré les fichiers de configuration et j'ai pu ainsi récupérer l'adresse IP des containers, et les lier dynamiquement via les ports d'écoute et de diffusion. Dans un deuxième temps, avec Omar nous nous sommes attaqués à l'élaboration de la partie ElasticSearch. J'ai ainsi du comprendre comment utiliser cet outil et comment le faire travailler avec Hadoop, avant de l'inclure dans un container. Pour faire fonctionner ElasticSearch il faut que le container Spark-HDFS soit bien configuré. J'ai donc aidé Florent à débugger cette configuration.

5 Florian Garibal

Durant ce projet, j'ai participé à **l'interconnexion des containers Docker de façon dynamique**. En effet il était nécessaire, vu qu'à chaque lancement, l'IP des containers était attribuée dynamiquement, de modifier les fichiers de configuration selon l'IP et port dynamique. Pour cela j'ai dans un premier temps dû localiser les différents fichiers de configuration à modifier et par la suite écrire un script shell permettant la modification de la ligne concernée.
Dans un deuxième temps j'ai aussi participé à la **connexion entre Spark, Kafka et HDFS**. Le but était de récupérer les données du topic *conso-maison*, stocké dans les brokers Kafka, depuis le fichier *Conf.java* de Spark. J'ai donc aidé Florent à débugger cette classe afin d'arriver à transmettre. Le plus gros problème que l'on a eu fut la communication entre les différents containers Docker. Pour finir j'ai participé à **l'écriture des données dans HDFS**. Après que Florent a effectué le traitement, l'ajout à la ligne de données du quartier VIP ou non, il a fallu stocker la ligne obtenue dans HDFS. Pour cela il a fallu configurer les droits, propriétaire et le lancement de HDFS au démarrage du container.
A côté, je me suis beaucoup renseigné sur les bonnes manières de faire quant à l'utilisation de plusieurs containers (liens entre eux, lancement, ...) et j'ai remarqué notamment le fait que l'option *link* était dépréciée ou encore que pour un meilleur lancement il fallait utiliser *docker compose*. Cependant je n'ai pas eu le temps de mettre en place ces solutions.

Conclusion

Dans ce projet nous avons réussi à déployer, grâce à la technologie Docker, la majeure partie des outils conseillés. Dans ce nouveau contexte de systèmes distribués, Docker nous a permis de surmonter la plupart des difficultés rencontrées. Cependant nous avons rencontré quelques difficultés dûes au fait que nous n'avons jamais installé Spark, Hadoop et ElasticSearch. En effet les documentations accessibles sur internet sont souvent spécifiques à l'infrastructure utilisée et ne correspondaient pas tout le temps à la notre.

Si nous avions réussi à mettre en place tout l'infrastructure, nous aurions pu améliorer certains points quant à cette dernière :

Dans un premier temps nous aurions pu changer la manière de lier les containers Docker en utilisant les *docker network* plutôt que les options *expose* et *link*. En effet, comme dit dans le rapport, la fonctionnalité *link* est désormais dépréciée et il est fortement déconseillé de l'utiliser. Afin de mettre en place les network il aurait fallu créer un docker network et spécifier que tous nos containers se lancent dans ce même network. Suite à cela, ils seraient donc tous connecter entre eux et nous n'aurions plus à utiliser les *link*. De plus, lorsque les containers sont dans le même réseau, un serveur DNS est disponible, ce qui permet d'utiliser le nom du container que l'on souhaite contacter au lieu de son adresse IP.

Dans un deuxième temps, on aurait pu améliorer le lancement du service *ssh-server* dans le container Spark. En effet il est nécessaire d'avoir un serveur SSH sur ce container pour que HDFS puisse communiquer avec toutes ses instances et récupérer les fichiers/informations demandés. Cependant nous n'avons pas réussi à démarrer ce service au démarrage du container avec *systemctl* ou *init.d*. De ce fait, à l'heure actuelle des choses, le démrrage du service se fait dans le */.bashrc* ce qui n'est pas du tout une bonne manière de faire. Il nous aurait donc fallu changer cette façon de lancer le service SSH en se renseignant de façon plus approfondie sur *systemctl* car il semble que cela soit la manière la plus appropriée de faire.

Dans un troisième temps, il nous aurait fallu mieux gérer les permissions d'accès et les *users* des containers. En effet, à l'heure actuelle des choses, dés que nous lançons un container nous sommes connectés en tant que *root*. Or, il est très déconseillé d'avoir un accès *root* permanent sur un serveur car, si jamais un attaquant arrive à le pénétrer, il aura alors tous les droits sur le serveur. Il nous aurait donc fallu créer un *user* pour chaque container qui aurait le strict minimum des droits dont il a besoin afin de remédier à ce problème.

Pour finir, pour un lancement plus propre et plus paramétrable de notre projet, nous aurions pu utiliser *docker compose* qui, à partir d'un fichier de configuration *YAML*, lance les dockers avec les paramètres souhaité. En effet on peut lui spécifier les ports à exposer, une dépendance sur un autre lancement de container, une ou des commandes à exécuter, etc. Nous avons essayé d'écrire ce genre de fichier mais cependant nous ne sommes pas arrivés à un résultat pertinent quant à son utilisation.