Manuel utilisateur de la stack SMACK

Ait El Kadi Ouiame, Ansari Othmane, Boudouin Philippe,
Boukrouh Insaf, Giroux Baptiste, Gouttefarde Léo,
Kodjo Edoh, Nahyl Othmane

Mardi 24 Janvier 2017

Table des matières

1 Mise en route

Pour commencer rapidement, un fichier README.md décrivant les éléments essentiels est fourni.

2 Installation / désinstallation

2.1 Première installation

Avant l'installation initiale de la stack sur le cluster, il faut lancer la commande suivante :

```
ZIP=~/setup.zip; cd ~; wget -O $ZIP \
  https://raw.githubusercontent.com/leogouttefarde/smack/master/setup.zip; \
  sudo apt -y install unzip; unzip -o $ZIP
```

Une fois la stack installée, les scripts de gestion deviennent disponibles sur l'ensemble des machines du cluster.

2.2 Installation

Pour lancer l'installation de la stack, il suffit de lancer le script `~/scripts/install_all.sh`

2.3 Désinstallation

Pour désinstaller la stack, il faut lancer le script `~/scripts/uninstall_all.sh`

2.4 Mise à jour des scripts

Pour mettre à jour les scripts, il suffit de lancer le script `~/scripts/update_setup.sh`

3 Gestion du cycle de vie

3.1 Démarrage

run_cassandra_cluster.sh

Permet de lancer les processus Cassandra sur tous les noeuds du cluster en appellant `run_cassandra_local.sh` sur chaque noeud. Le script inverse est `stop_cassandra_cluster.sh` .

run_cassandra_local.sh

Permet de lancer les processus Cassandra sur un hôte, il détecte aussi l'état du noeud Cassandra pour décider de la manière du lancement (start normal ou remplacement). Le script inverse est `kill_mesos_processes.sh` .

run_kafka_cluster.sh

Permet de lancer le scheduler kafka et de démarrer les brokers. Le nombre de ceux-ci est passé en argument. Le script inverse est `stop_kafka_cluster.sh` .

run_mesos_cluster.sh

Permet de démarrer le cluster mesos sur tous les maîtres et esclaves du cluster. Le script inverse est `stop_mesos_cluster.sh` .

run_mesos_local.sh

Permet de démarrer les daemons mesos. La commande de démarrage est différente selon le type de l'hôte. Le script inverse est `kill_mesos_processes.sh` .

3.2 Arrêt

stop_cassandra_cluster.sh

Arrête le cluster Cassandra.

stop_kafka_cluster.sh

Arrête le cluster Kafka.

stop_mesos_cluster.sh

Arrête le cluster Mesos.

kill_cassandra_processes.sh

Tue les processus Cassandra localement.

kill_cassandra_server.sh

Tue les processus Cassandra sur le serveur fourni en paramètre.

kill_mesos_server.sh

Tue les processus Mesos sur le serveur fourni en paramètre.

kill_mesos_processes.sh

Tue les processus Mesos localement.

kill_kafka_broker.sh

Tue le brokers Kafka de numéro fourni en paramètre.

3.3 Utilitaires

cassandra_status.sh

Affiche l'état du cluster Cassandra.

install_hosts.sh

Permet de configurer le fichier `/etc/hosts` sur toutes les machines en appellant le script `declare_hosts.sh` sur chacune de celles-ci. Le script inverse est `uninstall_hosts.sh` .

update_setup.sh

Met à jour les différents scripts sur tous les noeuds du cluster.

declare_hosts.sh

Permet de configurer, ou de rétablir l'ancienne configuration des hôtes en cas de désinstallation.

version.sh

Affiche le hash MD5 de l'archive actuellement installée.

run_cmd_all.sh

Lance la commande fourni en paramètre sur tous les noeuds.

4 Supervision du cluster

L'interface M/Monit permet de superviser l'état du cluster depuis le port 8080 de chaque machine, qui fournit une interface web dédiée.

Depuis cette interface, il est possible d'observer le cluster et de voir si un service est en panne par exemple. Si c'est le cas, le service est automatiquement redémarré sous 2 minutes maximum.

Il est également possible de forcer le démarrage d'un service, l'arrêter, le redémarrer ou lancer un monitoring de service manuellement au lieu d'attendre.

Aussi, il possible d'effectuer différentes statistiques sur le cluster et de consulter le journal des événements.

5 Application de démonstration

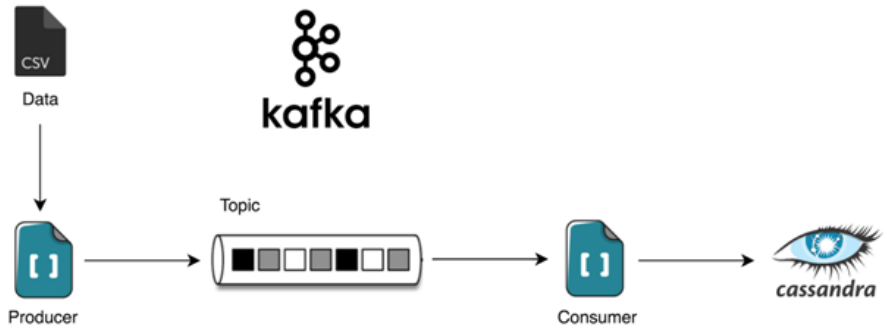
Pour le développement de l'application de démonstration, nous avons utilisé le langage de programmation Scala.

L'application concerne un dataset qui contient des données sur les voyages des taxis de NYC en date de 2013, et dont les champs sont les suivants :

medallion, hack_license, vendor_id, pickup_datetime, payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, total_amount

L'application peut être décomposée en deux parties :

5.1 L'intégration de données



Le producteur Kafka est un programme Scala qui permet de lire un fichier CSV à l'aide d'un itérateur (puisque le fichier est très volumineux et ne peut pas être chargé complètement en une seule fois). Chaque ligne lue est par la suite envoyée à un broker Kafka. Pour exécuter un programme Scala, il suffit de taper la commande "scala + nom du fichier".

Le consommateur Kafka est un programme écrit en Scala et exécuté avec Spark. Pour exécuter un programme en Spark, on aura besoin de produire un fichier jar relatif au programme à l'aide de SBT. La communication entre le producteur et le consommateur se fait au moyen de topic qui permet de séparer les catégories de données. Dans notre architecture, nous n'avons utilisé qu'un seul topic pour garantir le flux de communication entre le producteur Kafka et le consumer Kafka.

Pour ceci, on met le programme dans un dossier main qui lui même est contenu dans un dossier src, ensuite, à la racine du projet, on crée un fichier SBT qui contient toutes les dépendances dont on aura besoin lors de la compilation du programme (dépendances de Kafka, Cassandra, etc.) avec leurs versions spécifiées. La commande "sbt package" permet alors de générer ce jar, qui sera exécuté ensuite avec spark-submit (le connecteur de Cassandra est spécifié après –packages, et les jars externes après –jars et séparés par des virgules).

Chaque 40ms, le consommateur "consulte" le broker Kafka pour voir s'il y'a des lignes à "consommer" (un offset l'aide à savoir où il en est). Les lignes du fichier des données sont ensuite converties en dataframes et insérées dans la base de données Cassandra (qui a été créée au début).

5.2 Le traitement des données

La partie "Traitement" consiste en un programme Spark écrit en Scala, sa compilation et son exécution sont les mêmes que le consommateur (avec la seule différence qu'on aura pas besoin des jars externes).

Le programme lit les données de Cassandra et les met dans des dataframes. Ces dataframes pourront êtres mis dans des tables temporaires pour pouvoir effectuer des requêtes SQL brutes, ou bien on peut en effectuer des transformations Spark directement comme map, flatmap, groupby, etc. Les résultats des traitements sont stockées dans d'autres tables de notre base de données Cassandra.

On propose plusieurs types de traitement, par exemple :

— Nombre de transactions et chiffre d'affaire pour chaque méthode de paiement.
— Chiffre d'affaire pour chaque vendeur.
— Nombre de transactions pour chaque type de voyage.

Remarques :

— Les brokers de Kafka doivent être indiqués dans les deux premiers programmes (pour l'écriture et la lecture).
— Dans Cassandra, on a 2 datacenters avec une stratégie de réplication "Network Topology Strategy" et un degré de 3