|  |
| --- |
| DaSci – UE ARCHIBIGD  Février 2024  Année scolaire 2023/2024  Laurent Lecornu |
|  |

UE Architecture Big Data (UE ARCHIBIGD)

**TP7 : Architecture Kappa**

Modifié le : 07/03/2024

# Table des matières

Table des matières 2

1 Objectif : 3

2 Installation des services 3

2.1 Présentation des services 3

2.2 Services spark 4

3 Architecture Kappa 8

3.1 Lancer les containers 8

# ****Objectif :****

* Déploiement d’un exemple d’architecture Kappa
* Test de Cassandra (cqlsh)
  + Création de keyspaces
  + Réglage
  + Création de tables
  + Opération sur les tables
* Interaction avec Spark
* Utilisation avec pyspark
* Utilisation avec python
* Le driver python ne permet pas de créer de table. Il faudra le faire dans cqlsh.

# Installation des services

## Présentation des services

L’objectif est de créer 3 services (3 containers) :

* Services de l’architecture Kappa
  + container spark (master)
  + 2 containers spark (workers)
  + 2 containers cassandra
  + Containers kafka, zookeeper & kafdrop.
* Services producer



## Services spark

cd kappa

Voici le docker-compose.yml

version: "3"

services:

zookeeper:

image: confluentinc/cp-zookeeper:7.4.3

container\_name: zookeeper

networks:

- network1

ports:

- "2181:2181"

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

kafka:

image: confluentinc/cp-kafka:7.4.3

container\_name: kafka

networks:

- network1

depends\_on:

- zookeeper

ports:

- "9092:9092"

- "9091:9091"

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ADVERTISED\_HOST\_NAME: kafka:9092

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT\_HOST://localhost:9092

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT

# KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

# KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

# KAFKA\_CONFLUENT\_LICENSE\_TOPIC\_REPLICATION\_FACTOR: 1

# KAFKA\_CONFLUENT\_BALANCER\_TOPIC\_REPLICATION\_FACTOR: 1

# KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1

# KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR: 1

kafdrop:

image: obsidiandynamics/kafdrop:3.27.0

container\_name: kafdrop

networks:

- network1

depends\_on:

- kafka

- zookeeper

ports:

- 19000:9000

environment:

KAFKA\_BROKERCONNECT: kafka:29092

spark-master:

build: .

container\_name: spark-master

networks:

- network1

ports:

- "9080:8080"

- "7077:7077"

volumes:

- ./apps:/opt/spark-apps

- ./data:/opt/spark-data

environment:

- SPARK\_LOCAL\_IP=spark-master

- SPARK\_WORKLOAD=master

spark-worker-a:

build: .

container\_name: spark-worker-a

networks:

- network1

ports:

- "9081:8080"

- "7002:7000"

depends\_on:

- spark-master

environment:

- SPARK\_MASTER=spark://spark-master:7077

- SPARK\_WORKER\_CORES=1

- SPARK\_WORKER\_MEMORY=1G

- SPARK\_DRIVER\_MEMORY=1G

- SPARK\_EXECUTOR\_MEMORY=1G

- SPARK\_WORKLOAD=worker

- SPARK\_LOCAL\_IP=spark-worker-a

volumes:

- ./apps:/opt/spark-apps

- ./data:/opt/spark-data

spark-worker-b:

build: .

container\_name: spark-worker-b

networks:

- network1

ports:

- "9082:8080"

- "7001:7000"

depends\_on:

- spark-master

environment:

- SPARK\_MASTER=spark://spark-master:7077

- SPARK\_WORKER\_CORES=1

- SPARK\_WORKER\_MEMORY=1G

- SPARK\_DRIVER\_MEMORY=1G

- SPARK\_EXECUTOR\_MEMORY=1G

- SPARK\_WORKLOAD=worker

- SPARK\_LOCAL\_IP=spark-worker-b

volumes:

- ./apps:/opt/spark-apps

- ./data:/opt/spark-data

cassandra1:

image: cassandra:latest

container\_name: cassandra1

ports:

- "9042:9042"

environment:

- CASSANDRA\_SEEDS=cassandra1,cassandra2

networks:

- network1

cassandra2:

image: cassandra:latest

container\_name: cassandra2

environment:

- CASSANDRA\_SEEDS=cassandra1,cassandra2

networks:

- network1

networks:

network1:

name: mynetwork

driver: bridge

Construire les images, démarrer les containers et vérifier.

(attention sur le decompose-docker.yml du producer, le chemin d’un volume doit être modifé)

cd kappa

docker-compose up -d

cd ../producer

docker-compose up -d

docker ps

Vérifiez que tous les containers sont bien en mode running.

(Regarder le dashboard)

# Architecture Kappa

A force de créer des images et des containers, le cache se remplit et sature le disque ce qui empêche le bon fonctionnement des containers.

Il faut alors effectuer une purge du cache.

docker buildx prune -f

## ****Lancer les containers****

On va utiliser 3 terminaux bash : sur le producer, sur spark-master et sur cassandra1.

Ouvrir un premier terminal. Il servira à créer le container producer.

docker exec -ti producer bash

Ouvrir un deuxième terminal. Il servira à créer les containers nécessaires à l’architecture kappa.

docker exec -ti spark-master bash

Sur un troisième terminal, on va se connecter sur un terminal bash du container cassandra1.

docker exec -ti cassandra1 bash

puis vous y exécuterez la commande suivante.

root@fbce05e80d97:/# cqlsh

On va y créer la KEYSPACE test

CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': 1 };

Sur le deuxième terminal, on peut maintenant se connecter sur le terminal bash de spark-master

On se place sur le répertoire ../spak-apps, puis on excécute la commande suivante qui permet de créer la table demo.transactions1 sur cassandra1.

root@e460143c2821:/opt/spark-apps# cd ../spark-apps

root@e460143c2821:/opt/spark-apps# python3 TP7\_cassandra.py

Attention, il faut indiquer dans TP7\_cassandra.py le numéro ip du container cassandra1.

Vous pouvez facilement le trouver en utilisant la commande suivante (ou en utilisant le dashboard)

docker network inspect mynetwork

On vérifie la création de la table en exécutant la commande suivante sur le cqlsh (ouvert précédemment)

cqlsh> describe tables

Keyspace demo

-------------

transactions1

….

Puis

cqlsh> select \* from demo.transactions1 ;

date | capteur | numero | valeur

------+---------+--------+--------

(0 rows)

On peut maintenant démarrer l’alimentation de kafka sur le terminal bash du producer par

c9511fefdb8b6:/publisher/app# python generate-random-events.py

(des messages vont apparaître)

Puis le traitement des messages reçus par spark-streaming,

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1,com.datastax.spark:spark-cassandra-connector\_2.12:3.5.0 --conf spark.cassandra.connection.host=cassandra1 --conf spark.cassandra.connection.port=9042 data-pipeline-streaming\_cassandra.py kafka-demo-events

On attend que spark-streaming se lance et en retournant sur cqlsh on observe le remplissage de la table demo.transaction1 par

cqlsh> select \* from demo.transactions1 ;