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

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Table des matières

1 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

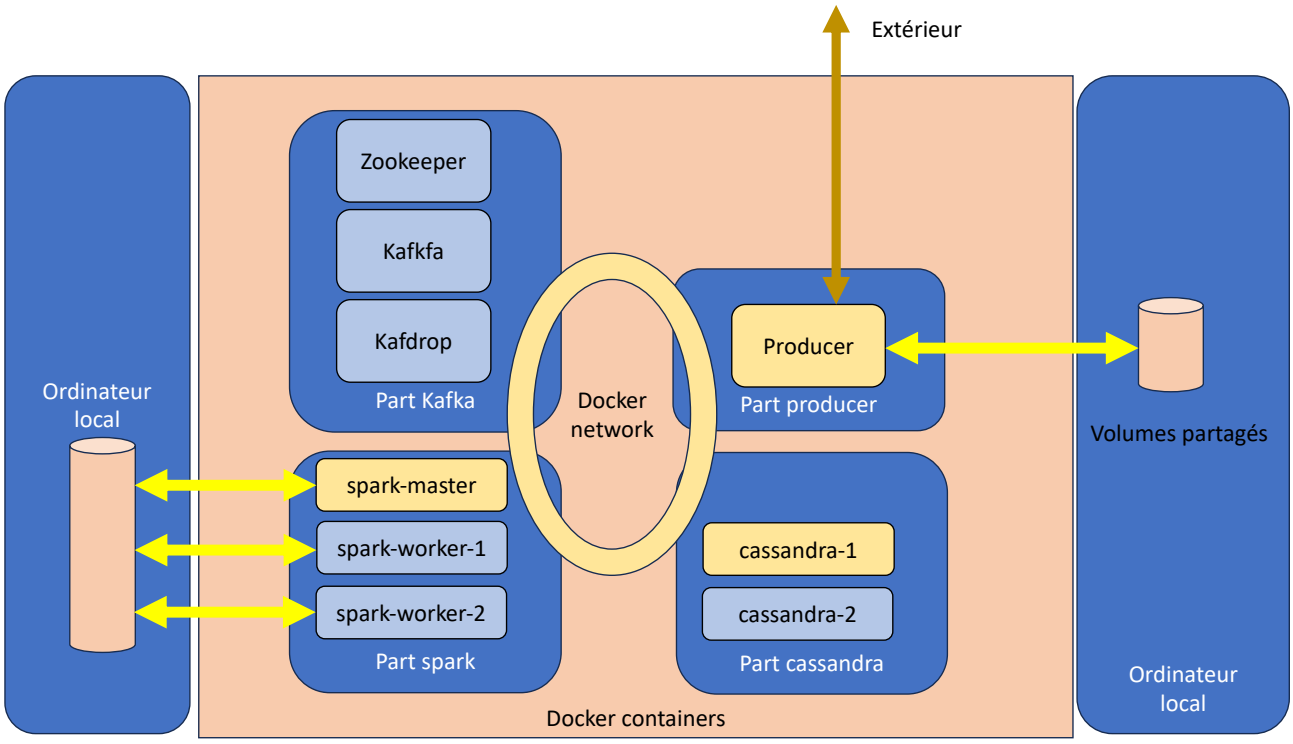
- Le driver python ne permet pas de créer de table. Il faudra le faire dans `cqlsh`.

2 Installation des services

2.1 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



2.2 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)

3 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
```

3.1 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 ;
```