TP n° 2 Kafka

Nom prénom binôme 1

Nom prénom binôme 2 (Pas de trînome sauf groupe impaire).

I. Kafka

Relancer les containers dockers que vous avez : docker start hadoop-master hadoop-worker1 hadoop-worker2

Aller sur le container master : docker exec -it hadoop-master bash puis lancer hadoop : ./start-hadoop.sh

Verifier que les processus background sont bien lancés avec la commande : jps.

Quels sont les processus qui tournent sur le container master ?

Lancer maintenant kafka avec ./start-kafka-zookeeper.sh. Relancer la commande jps et inclure un screenshoot montrant que kafka est bien lancé.

II. Premier pas avec Kafka

Principaux concepts de Kafka :

Topics : Les données sont organisées en "topics", qui sont des flux de messages. Chaque message est associé à un topic spécifique.

Producers : Ce sont les entités qui envoient des données aux topics Kafka.

Consumers : Les consommateurs récupèrent les données des topics Kafka et les traitent selon les besoins de l'application.

Brokers : Les brokers Kafka sont les serveurs responsables du stockage et de la gestion des messages. Ils sont responsables de la répartition et de la gestion des données sur les différents nœuds du cluster Kafka.

Partitions : Chaque topic est divisé en un ou plusieurs "partitions". Les partitions permettent de distribuer les données de manière parallèle et d'assurer la scalabilité du système.

Fonctionnement :

Production de données : Les producteurs envoient des messages à un ou plusieurs topics Kafka.

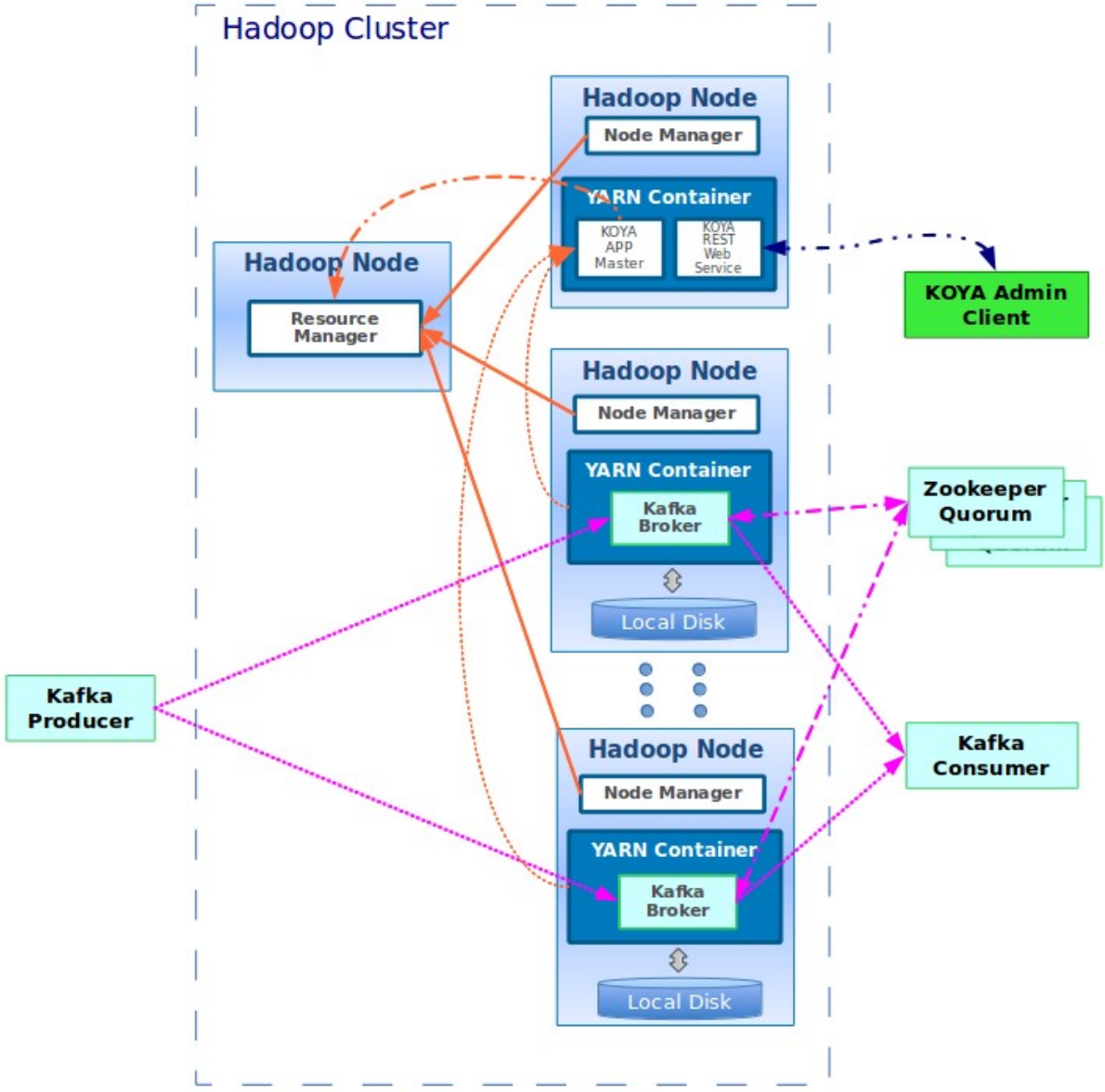
Stockage des messages : Les messages sont stockés de manière durable sur les brokers Kafka, répartis sur différentes partitions.

Consommation des données : Les consommateurs récupèrent les messages à partir des partitions et les traitent selon les besoins de l'application.

Traitement des données en temps réel : Kafka permet le traitement des données en temps réel, ce qui signifie que les données peuvent être analysées et transformées au fur et à mesure de leur arrivée.

Scalabilité et tolérance aux pannes : Kafka offre une architecture distribuée qui permet de scaler facilement en ajoutant de nouveaux brokers au cluster. De plus, grâce à la réplication des données, Kafka est capable de tolérer les pannes matérielles et de garantir la disponibilité des données.

Kafka s’insère comme cela dans le Hadoop System File :



Petit point sur Zookeeper : Il s'agit d'un service de coordination distribué qui est utilisé pour la gestion des clusters Kafka. Voici comment ZooKeeper s'intègre dans le fonctionnement de Kafka :

ZooKeeper dans l'écosystème Kafka :

Coordination des brokers : ZooKeeper est utilisé pour la coordination des brokers Kafka. Chaque broker enregistre ses métadonnées (comme les topics, les partitions, les consommateurs, etc.) dans ZooKeeper. Cela permet aux brokers de découvrir les autres brokers du cluster et de s'inscrire auprès d'eux.

Élection de leader : Dans un cluster Kafka, chaque partition est répliquée sur plusieurs brokers pour assurer la redondance et la disponibilité des données. ZooKeeper est utilisé pour coordonner

l'élection d'un leader pour chaque partition. Le leader est responsable de la gestion des écritures et des lectures pour cette partition.

Détection des pannes : ZooKeeper surveille l'état des brokers Kafka et des partitions. En cas de panne d'un broker ou d'une partition, ZooKeeper peut déclencher des actions telles que la réélection d'un leader ou le rééquilibrage des partitions.

Configuration dynamique : ZooKeeper permet également la gestion de la configuration dynamique de Kafka. Les paramètres de configuration, tels que les quotas, les ACL (listes de contrôle d'accès) et les stratégies de réplication, peuvent être stockés et mis à jour dynamiquement dans ZooKeeper.

P.S : Zookeeper n'est plus trop utilisé (remplacé par KRaft entre autres ou d'autres solutions payantes).

En se basant sur la description de Kafka, et la ressource suivante : https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.linkedin.com%2Fpulse%2Fhow-deploy-kafka-zookeeper-cluster-linux-based-operating-tiwari&psig=AOvVaw1UtFhomTg_5lxrYo_2T_Cy&ust=1714680951358000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCJCz07ii7YUDFQAAAAAdAAAAABAE

Refaites un schéma sur Figma ou autre illustrant l'utilisaiton de Kafka et Zookeeper dans un HDFS.

III. Lancement de Kafka

Pour lancer Kafka sur le docker main (on devrait normalement créer le consumer et producer en dehors des docker malheureusement la manip pour accéder à votre VM en dehors de Docker est compliqué à mettre en place donc on va rester sur docker pour éviter les soucis techniques), utiliser la commande suivante : ./kafka-topics.sh --create --topic data_stream --bootstrap-server localhost:9092

Elle va créer un tropic nommé data_stream sur le port 9092 local.

Installer ensuite la bibliothèque python : confluent_kafka à l'aide de pip install confluent_kafka.

Doc dispo ici : https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html

On va ensuite envoyer des données à l’aide du script produce.py (à créer à la racine de votre docker main :

```
from confluent_kafka import Producer

def delivery_callback(err, msg):
    if err:
        print(f"Failed to deliver message: {err}")
    else:
        print(f"Message delivered: {msg}")


p = Producer({'bootstrap.servers': 'localhost:9092'})

p.produce('data_stream', 'Insérer ici votre premier message', callback=delivery_callback)

p.produce('data_stream', 'Deuxieme', callback=delivery_callback)

// Ainsi de suite faire une 10 aine de message différent

p.flush()
```

Maintenant on va lire ces messages à l'aide de consume.py

```
from confluent_kafka import Consumer, KafkaError
c = Consumer({
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'my_group',
    'auto.offset.reset': 'earliest'
})

c.subscribe(['data_stream'])

while True:
    msg = c.poll(timeout=1.0)
    if msg is None:
        continue
    if msg.error():
        if msg.error().code() == KafkaError._PARTITION_EOF:
            continue
        else:
            print(msg.error())
            break
    print('Received message: {}'.format(msg.value().decode('utf-8')))

c.close()
```

Insérer ici un screenshot des messages bien reçus par le consumer.

IV. Stockage sur le HDFS

On va maintenant stocker les données envoyées sur un fichier txt sur le HDFS.

Executer ce code dans un script consumeinhdfs.py (installer hdfs avec pip install hdfs si besoin) :

```
from confluent_kafka import Consumer
from hdfs import InsecureClient

client = InsecureClient('http://localhost:9870')

# Créer un répertoire dans HDFS pour stocker les données
client.makedirs('/root/data')

c = Consumer({
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'my_group',
    'auto.offset.reset': 'earliest'
})

c.subscribe(['data_stream'])

while True:
    msg = c.poll(timeout=1.0)
    if msg is None:
        continue
    if msg.error():
        if msg.error().code() == KafkaError._PARTITION_EOF:
            continue
        else:
            print(msg.error())
            break
```

```
    # Écrire les données dans un fichier sur HDFS
    with client.write('/root/data/data_stream.txt', overwrite=True) as writer:
        writer.write(msg.value().decode('utf-8') + '\n')

c.close()
```

Relancer le code qui envoie des données sur kafka depuis un nouveau terminal. Regarder si le fichier data_stream.txt est bien crée dans le HDFS. Insérer un screenshoot de son contenu ici.

V. Kafka + Spark

Pour finir on va exécuter un comptage de mots simple avec Spark sur le flux de données à l'aide du script : countmessagespark.py

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Analyse de données") \
    .getOrCreate()

# Charger les données depuis HDFS en tant que DataFrame
df = spark.read.text("hdfs://localhost:9870/root/data/data_stream.txt")

# Effectuer une analyse simple
word_count = df.count()

print("Nombre total de messages:", word_count)

spark.stop()
```

Si ça sort un erreur de taille, il faudra éditer le fichier : $HADOOP_HOME/etc/hadoop/hdfs_site.xml

<configuration>

<property>

<name>dfs.blocksize</name>

<value>VALEUR PAR DEFAUT</value> ← la changer par une valeur plus grande.

</property>

</configuration>

Insérer le résultat du comptage de mots ici, tenter ensuite un vrai word count comme vu TP2 ou TP1 à l'aide d'un MapReduce ou de Spark sur le live données.