

南昌大学实验报告

姓名: Qing Liu

学号: 6130116184

邮箱地址: 1119637652@qq.com

专业班级: Class 164 of Computer Science and Technology

实验日期: THU, May 9th, 2019

课程名称: Cloud Computing Technology Experiments

实验项目名称

Introduction to Cloud Computing -- Kafka

实验目的

- Understanding the concept of message passing
- Trying to follow up the procedure of a message broker that handles message from many tenants
- Repeating what others have done in the past sheds the light on your future

实验基础

- Linux
- Docker
- Zookeeper
- Kafka

实验步骤

- Task 1 - Spark-kafka-docker demo

Setting up Project with sbt

- Download sbt and umcompress it into `/usr/local`, and configure the environment variable for it.

```
wget https://piccolo.link/sbt-1.2.8.zip
unzip sbt-1.2.8.zip
mv sbt /usr/local
```

Edit `/etc/profile` to configure the environment variable

```
# /etc/profile
...

export SBT_HOME=/usr/local/sbt
export PATH=$PATH:$SBT_HOME/bin
```

```
source /etc/profile
reboot
```

- Create a project directory

Here is the content of the project:

```
spark-kafka-docker-demo
|__project
|   |__plugins.sbt
|   |__build.sbt
|   |__src
|       |__main
|           |__scala
|               |__com
|                   |__example
|                       |__spark
|                           |__DirectKafkaWorkCount.scala
```

```
mkdir spark-kafka-docker-demo && cd spark-kafka-docker-demo
mkdir -p src/main/scala/com/example/spark
mkdir project
```

- Create `project/plugins.sbt` file

Hint: remember to change the version to `0.14.9`

```
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.14.9")
```

- Define build configuration in `build.sbt` file

```
name := "direct_kafka_word_count"

scalaVersion := "2.10.5"

val sparkVersion = "1.5.1"

libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion % "provided",
  "org.apache.spark" %% "spark-streaming" % sparkVersion % "provided",
  ("org.apache.spark" %% "spark-streaming-kafka" % sparkVersion) exclude ("o
)

assemblyJarName in assembly := name.value + ".jar"
```

- Edit `src/main/scala/com/example/spark/DirectKafkaWordCount.scala`

```
package com.example.spark

import kafka.serializer.StringDecoder
import org.apache.spark.{TaskContext, SparkConf}
import org.apache.spark.streaming.kafka.{OffsetRange, HasOffsetRanges, Kafka}
import org.apache.spark.streaming.{Seconds, StreamingContext}

object DirectKafkaWordCount {
  def main(args: Array[String]): Unit = {
    if (args.length < 2) {
      System.err.println(s"""
        |Usage: DirectKafkaWordCount <brokers> <topics>
        | <brokers> is a list of one or more Kafka brokers
        | <topics> is a list of one or more kafka topics to consume from
        |
        |""".stripMargin)
      System.exit(1)
    }

    val Array(brokers, topics) = args

    // Create context with 10 second batch interval
    val sparkConf = new SparkConf().setAppName("DirectKafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(10))

    // Create direct kafka stream with brokers and topics
    val topicsSet = topics.split(",").toSet
    val kafkaParams = Map[String, String]("metadata.broker.list" -> brokers)
```

```

val messages = KafkaUtils.createDirectStream[String, String, StringDecoder](
  ssc, kafkaParams, topicsSet)

// Get the lines, split them into words, count the words and print
val lines = messages.map(_._2)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1L)).reduceByKey(_ + _)
wordCounts.print()

// Start the computation
ssc.start()
ssc.awaitTermination()
}
}

```

- Build fat jar by running `sbt assembly` from project's root

```

[info] Done updating.
[warn] There may be incompatibilities among your library dependencies; run 'evicted' to see detailed eviction warnings.
[info] Compiling 1 Scala source to /home/cleo/spark-kafka-docker-demo/target/scala-2.10/classes ...
[info] Non-compiled module 'compiler-bridge_2.10' for Scala 2.10.5. Compiling...
[info]   Compilation completed in 10.754s.
[info] Done compiling.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.protobuf.UnsafeUtil (file:/root/.sbt/boot/scala-2.12.7/org.scala-sbt/sbt/1.2.8/protobuf-java-3.3.1.jar) to field java.nio.Buffer.address
WARNING: Please consider reporting this to the maintainers of com.google.protobuf.UnsafeUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[info] Strategy 'discard' was applied to 10 files (Run the task at debug level to see details)
[info] Strategy 'rename' was applied to 5 files (Run the task at debug level to see details)
[info] Packaging /home/cleo/spark-kafka-docker-demo/target/scala-2.10/direct_kafka_word_count.jar ...
[info] Done packaging.
[succes] Total time: 617 s, completed May 11, 2019, 12:14:17 PM

```

```

root@vm-xenial0:/home/cleo/spark-kafka-docker-demo/target/scala-2.10# ls
classes direct_kafka_word_count.jar resolution-cache
root@vm-xenial0:/home/cleo/spark-kafka-docker-demo/target/scala-2.10#

```

Setup Container

I have already installed the docker and docker-compose in my virtual machines.

- Edit the `docker-compose.yml` in root directory of project

```

kafka:
  image: antlypls/kafka-legacy
  environment:
    - KAFKA=localhost:9092
    - ZOOKEEPER=localhost:2181
  expose:
    - "2181"
    - "9092"

spark:
  image: antlypls/spark:1.5.1
  command: bash
  volumes:
    - ./target/scala-2.10:/app
  links:
    - kafka

```

A lot of things are going on here, let's go through it step by step. This yml file defines two services: kafka and docker. `Kafka` service runs image based on `spotify/kafka` repository, this image provides everything we need for running kafka in one container: kafka broker and zookeeper server. Also two environment variables are added: `KAFKA` and `ZOOKEEPER`, those variables are helpful when you run kafka CLI tools inside kafka container. We also expose a kafka broker port `9092` and a port for zookeeper `2181`, so linked services can access it. Then `spark` service is defined. I've prepared an image `antlypls/spark`, which provides spark running on YARN. The image is slightly modified version of `sequenceiq/spark` repository, with `spark 1.5.1` and without a few packages that we don't need for this demo. We specify `bash` as command, since we want to have interactive shell session within the spark container. `volumes` option mounts build directory into the spark container, so we will be able to access `.jar` right in the container. At the end `kafka` service is linked to `spark`.

- Now we are ready to run everything together.

Start all containers with docker-compose: `docker-compose run --rm spark`. This starts kafka and then spark and logs us into spark container shell. The `--rm` flag makes docker-compose to delete corresponding spark container after run.

```
Starting sshd: [ OK ]
Starting namenodes on [9baa58ab0202]
9baa58ab0202: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-9baa58ab0202.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-9baa58ab0202.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-9baa58ab0202.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanager-9baa58ab0202.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-9baa58ab0202.out
bash-4.1#
```

Test for Word Count

- Create a topic in a kafka broker that we are going to read from.

To do that open a separate terminal session and run:

```
docker exec -it $(docker-compose ps -q kafka) bash
kafka-topics.sh --create --zookeeper $ZOOKEEPER --replication-factor 1 --par
```

```
root@342b8bc7ccf6:/# kafka-topics.sh --create --zookeeper $ZOOKEEPER --replicati
on-factor 1 --partitions 2 --topic word-count
Created topic "word-count".
```

- Check that new topic has been created by running commands

```
kafka-topics.sh --list --zookeeper $ZOOKEEPER
kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic word-count
```

```

root@342b8bc7ccf6:/# kafka-topics.sh --list --zookeeper $ZOOKEEPER
word-count
root@342b8bc7ccf6:/# kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic word-count
Topic:word-count      PartitionCount:2      ReplicationFactor:1   Configs:
    Topic: word-count      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
    Topic: word-count      Partition: 1      Leader: 0      Replicas: 0      Isr: 0

```

Keep this shell session open, we will use it to add messages to the topic.

- Go back to spark container shell and run

```

spark-submit \
--master yarn-client \
--class com.example.spark.DirectKafkaWordCount \
app/direct_kafka_word_count.jar kafka:9092 word-count

```

```

bash-4.1# spark-submit \
> --master yarn-client \
> --class com.example.spark.DirectKafkaWordCount \
> app/direct_kafka_word_count.jar kafka:9092 word-count
19/05/11 03:32:10 INFO SparkContext: Running Spark version 1.5.1
19/05/11 03:32:10 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
19/05/11 03:32:10 INFO spark.SecurityManager: Changing view acls to: root
19/05/11 03:32:10 INFO spark.SecurityManager: Changing modify acls to: root
19/05/11 03:32:10 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); users with modify permissions: Set(root)
19/05/11 03:32:11 INFO slf4j.Slf4jLogger: Slf4jLogger started
19/05/11 03:32:11 INFO Remoting: Starting remoting
19/05/11 03:32:11 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@172.17.0.3:35237] Replicas: 0
19/05/11 03:32:11 INFO util.Utils: Successfully started service 'sparkDriver' on port 35237.
19/05/11 03:32:11 INFO spark.SparkEnv: Registering MapOutputTracker
19/05/11 03:32:11 INFO spark.SparkEnv: Registering BlockManagerMaster
19/05/11 03:32:11 INFO storage.DiskBlockManager: Created local directory at /tmp/blockmgr-251045f6-9538-4a04-9de1-29018175d4b3
19/05/11 03:32:11 INFO storage.MemoryStore: MemoryStore started with capacity 530.3 MB
19/05/11 03:32:12 INFO spark.HttpFileServer: HTTP File server directory is /tmp/spark-6a0534e7-582b-4d16-8143-c5baa181235f/httpd-e6ea6afc-8621-428e-a438-0911e7c0a172
19/05/11 03:32:12 INFO spark.HttpServer: Starting HTTP Server
19/05/11 03:32:12 INFO server.Server: Jetty-8.y.z-SNAPSHOT
19/05/11 03:32:12 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0:40693
19/05/11 03:32:12 INFO util.Utils: Successfully started service 'HTTP file server' on port 40693.
19/05/11 03:32:12 INFO spark.SparkEnv: Registering OutputCommitCoordinator
19/05/11 03:32:12 INFO server.Server: Jetty-8.y.z-SNAPSHOT
19/05/11 03:32:12 INFO server.AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
19/05/11 03:32:12 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
19/05/11 03:32:12 INFO ui.SparkUI: Started SparkUI at http://172.17.0.3:4040
19/05/11 03:32:12 INFO spark.SparkContext: Added JAR file: app/direct_kafka_word_count.jar at http://172.17.0.3:40693/jars/direct_kafka_word_count.jar with timestamp 1557559932493

```

Here we launch our application in `yarn-client` mode because we want to see output from the driver.

You might see a lot of logs written to output, that is useful for debugging, but it might be hard to see actual app output. You could use following settings for log4j if you wanted to disable those debugging logs:

```

log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.threshold=ERROR
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{

```

Replace `$SPARK_HOME/conf/log4j.properties` file with one provided above. Or simply put it somewhere in container, e.g. in shared app directory and run app like

```
spark-submit \
--master yarn-client \
--driver-java-options "-Dlog4j.configuration=file:///app/log4j.properties" \
--class com.example.spark.DirectKafkaWordCount \
app/direct_kafka_word_count.jar kafka:9092 word-count
```

```
bash-4.1# cat app/log4j.properties
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.threshold=ERROR
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1};
%m%n
```

```
bash-4.1# spark-submit \
> --master yarn-client \
> --driver-java-options "-Dlog4j.configuration=file:///app/log4j.properties" \
> --class com.example.spark.DirectKafkaWordCount \
> app/direct_kafka_word_count.jar kafka:9092 word-count
```

Note, that `docker-compose` updates `hosts` file in running containers, so linked services can be accessed using service name as a hostname. That's why in our demo we simply use `kafka:9090` as a broker address.

- Now let's add some data into the topic, just run following in the `kafka` container

```
kafka-console-producer.sh --broker-list $KAFKA --topic word-count
```

And for input like this:

```
Hello World
!!!
```

You should see output from spark app like:

```
-----
Time: 1234567890000 ms
-----
(Hello,1)
(World,1)
(!!!,1)
```



```

cleo@vm-xenial0: ~/spark-kafka-docker-demo
Created topic "word-count".
root@342b8bc7ccf6:/# kafka-topics.sh --list --zookeeper $ZOOKEEPERword-count
root@342b8bc7ccf6:/# kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic w
ord-count
Topic:word-count      PartitionCount:2      ReplicationFactor:1   Configs:
Topic: word-count      Partition: 0          Leader: 0              Replicas: 0          I
sr: 0
Topic: word-count      Partition: 1          Leader: 0              Replicas: 0          I
sr: 0
root@342b8bc7ccf6:/# kafka-topics.sh --list --zookeeper $ZOOKEEPER
word-count
root@342b8bc7ccf6:/# kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic w
ord-count
Topic:word-count      PartitionCount:2      ReplicationFactor:1   Configs:
Topic: word-count      Partition: 0          Leader: 0              Replicas: 0
Isr: 0
Time: 1557560790000 ms
Topic: word-count      Partition: 1          Leader: 0              Replicas: 0
Isr: 0
Time: 1557560810000 ms
root@342b8bc7ccf6:/# kafka-console-producer.sh --broker-list $KAFKA --topic word
-count
Hello World!
!!!
Time: 1557560820000 ms
(Hello,1)
(!!!,1)
(World!,1)
Time: 1557560830000 ms

```

`docker-compose` doesn't stop/delete linked containers when `run` command exits. To stop linked containers run `docker-compose stop`, and `docker-compose rm` to delete them.

```

cleo@vm-xenial0:~/spark-kafka-docker-demo$ docker-compose stop kafka
Stopping sparkkafkadockerdemo_kafka_1 ... done
cleo@vm-xenial0:~/spark-kafka-docker-demo$ docker-compose rm kafka
Going to remove sparkkafkadockerdemo_kafka_1
Are you sure? [yN] y
Removing sparkkafkadockerdemo_kafka_1 ... done

```

- Task 2 - Stream Word Count demo

Prerequisites

- Install docker

I have finished it for a long time. Check [here](#).

- Map the hostname `zookeeper` and `broker` to docker host ip in host file

Add 2 maps in `/etc/hosts` :

```

192.168.206.131    broker
192.168.206.131    zookeeper

```

- Start kafka and zookeeper services

Pull the images from Docker Hub

```
docker pull zookeeper
docker pull wurstmeister/kafka
```

Edit the `docker-compose.yml` :

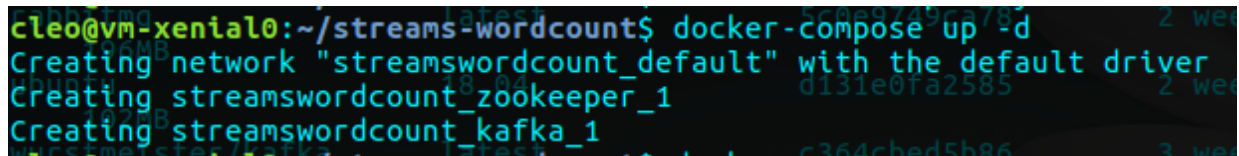
```
version: '2'

services:
  zookeeper:
    image: zookeeper
    volumes:
      - /etc/localtime:/etc/localtime
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka
    volumes:
      - /etc/localtime:/etc/localtime
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: broker
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

Run containers with `docker-compose` :

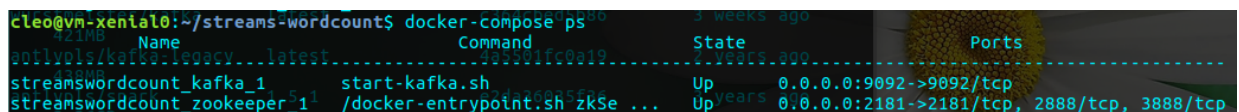
```
docker-compose up -d
```



```
cleo@vm-xenia10:~/streams-wordcount$ docker-compose up -d
Creating network "streamswordcount_default" with the default driver
Creating streamswordcount_zookeeper_1
Creating streamswordcount_kafka_1
```

Check the running containers

```
docker-compose ps
```



```
cleo@vm-xenia10:~/streams-wordcount$ docker-compose ps
Name                                Command                                State      Ports
streamswordcount_kafka_1            start-kafka.sh                        Up         0.0.0.0:9092->9092/tcp
streamswordcount_zookeeper_1        /docker-entrypoint.sh zkSe ...       Up         0.0.0.0:2181->2181/tcp, 2888/tcp, 3888/tcp
```

- Create topics

Create the `input` topic

```
docker exec -it $(docker-compose ps -q kafka) bash
kafka-topics.sh --create --zookeeper zookeeper:2181 \
--replication-factor 1 \
--partitions 1 \
--topic streams-plaintext-input
```

```
bash-4.4# kafka-topics.sh --create --zookeeper zookeeper:2181 \
> --replication-factor 1 \
> --partitions 1 \
> --topic streams-plaintext-input
Created topic streams-plaintext-input.
```

Create the `output` topic

```
kafka-topics.sh --create --zookeeper zookeeper:2181 \
--replication-factor 1 \
--partitions 1 \
--topic streams-wordcount-output \
--config cleanup.policy=compact
```

```
bash-4.4# kafka-topics.sh --create --zookeeper zookeeper:2181 \
> --replication-factor 1 \
> --partitions 1 \
> --topic streams-wordcount-output \
> --config cleanup.policy=compact
Created topic streams-wordcount-output.
```

- Describe them

```
kafka-topics.sh --zookeeper zookeeper:2181 --describe
```

```
bash-4.4# kafka-topics.sh --zookeeper zookeeper:2181 --describe
Topic: streams-plaintext-input PartitionCount:1 ReplicationFactor:1 Configs:
Topic: streams-plaintext-input Partition: 0 Leader: 1001 Replicas: 1001 Isr: 1001
Topic: streams-wordcount-output PartitionCount:1 ReplicationFactor:1 Configs:cleanup.policy=compact
Topic: streams-wordcount-output Partition: 0 Leader: 1001 Replicas: 1001 Isr: 1001
```

- Code for java `WordCountDemo` :

The website of the code in the given blog response 404. So I found it in

```
package com.gerardnico.kafka.demo;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
```

```
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.KeyValueMapper;
import org.apache.kafka.streams.kstream.Produced;
import org.apache.kafka.streams.kstream.ValueMapper;

import java.util.Arrays;
import java.util.Locale;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

public class WordCountDemo {

    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.StringSerde.class);
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.StringSerde.class);

        // setting offset reset to earliest so that we can re-run the demo c
        // Note: To re-run the demo, you need to use the offset reset tool:
        // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+A
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        StreamsBuilder builder = new StreamsBuilder();

        KStream<String, String> source = builder.stream("streams-plaintext-i

        KTable<String, Long> counts = source
            .flatMapValues(new ValueMapper<String, Iterable<String>>() {
                @Override
                public Iterable<String> apply(String value) {
                    return Arrays.asList(value.toLowerCase(Locale.getDefault())
                }
            })
            .groupBy(new KeyValueMapper<String, String, String>() {
                @Override
                public String apply(String key, String value) {
                    return value;
                }
            })
            .count();

        // need to override value serde to Long type
        counts.toStream().to("streams-wordcount-output", Produced.with(Serdes.LongSerde.class));

        final KafkaStreams streams = new KafkaStreams(builder.build(), props);
        final CountDownLatch latch = new CountDownLatch(1);

        // attach shutdown handler to catch control-c
        Runtime.getRuntime().addShutdownHook(new Thread("streams-wordcount-s
```

```

        @Override
        public void run() {
            streams.close();
            latch.countDown();
        }
    });

    try {
        streams.start();
        latch.await();
    } catch (Throwable e) {
        System.exit(1);
    }
    System.exit(0);
}
}

```

- Run the wordcount application

```
kafka-run-class.sh org.apache.kafka.streams.examples.wordcount.WordCountDemo
```

```

bash-4.4# kafka-run-class.sh org.apache.kafka.streams.examples.wordcount.WordCountDemo
[2019-05-12 04:18:33,103] WARN The configuration 'admin.retries' was supplied but isn't a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
[2019-05-12 04:18:33,104] WARN The configuration 'admin.retry.backoff.ms' was supplied but isn't a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)

```

- Test for WordCountDemo

Now we can start the console producer in a separate terminal to write some input data to this topic:

```
kafka-console-producer.sh --broker-list localhost:9092 --topic streams-plain
```

Inspect the output of the WordCount demo application by reading from its output topic with the console consumer in a separate terminal:

```

kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic streams-wordcount-output \
--from-beginning \
--formatter kafka.tools.DefaultMessageFormatter \
--property print.key=true \
--property print.value=true \
--property key.deserializer=org.apache.kafka.common.serialization.StringDese
--property value.deserializer=org.apache.kafka.common.serialization.LongDese

```

```

cleo@vm-xenial0: ~/streams-wordcount
cleo@vm-xenial0:~$ cd streams-wordcount/
cleo@vm-xenial0:~/streams-wordcount$ docker-compose exec kafka bash
bash-4.4# exit
cleo@vm-xenial0:~/streams-wordcount$ kafka-console-consumer.sh --bootstrap-server 192.168.206.131:9092 \
> ^C
cleo@vm-xenial0:~/streams-wordcount$ kafka-console-producer.sh --broker-list 192.168.206.131:9092 --topic streams-plaintext-input
kafka-console-producer.sh: command not found
cleo@vm-xenial0:~/streams-wordcount$ docker-compose exec kafka bash
bash-4.4# kafka-console-producer.sh --broker-list 192.168.206.131:9092 --topic streams-plaintext-input
>hello
>world
>hello
>hello stream
>
bash-4.4# kafka-console-consumer.sh --bootstrap-server 192.168.206.131:9092 \
> --topic streams-wordcount-output \
> --from-beginning \
> --formatter kafka.tools.DefaultMessageFormatter \
> --property print.key=true \
> --property print.value=true \
> --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
> --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
hello      1
world      1
hello      2
hello      3
stream     1

```

实验数据或结果

Spark-kafka-docker demo

```

cleo@vm-xenial0: ~/spark-kafka-docker-demo
Created topic "word-count".
root@342b8bc7ccf6:/# kafka-topics.sh --list --zookeeper $ZOOKEEPER
word-count
root@342b8bc7ccf6:/# kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic word-count
Topic:word-count      PartitionCount:2      ReplicationFactor:1   Configs:
sr: 0
  Topic: word-count    Partition: 0          Leader: 0              Replicas: 0          I
  Topic: word-count    Partition: 1          Leader: 0              Replicas: 0          I
sr: 0
root@342b8bc7ccf6:/# kafka-topics.sh --list --zookeeper $ZOOKEEPER
word-count
root@342b8bc7ccf6:/# kafka-topics.sh --describe --zookeeper $ZOOKEEPER --topic word-count
Topic:word-count      PartitionCount:2      ReplicationFactor:1   Configs:
Isr: 0
  Topic: word-count    Partition: 0          Leader: 0              Replicas: 0          I
  Topic: word-count    Partition: 1          Leader: 0              Replicas: 0          I
Isr: 0
root@342b8bc7ccf6:/# kafka-console-producer.sh --broker-list $KAFKA --topic word-count
Hello World!
!!!
Time: 1557560810000 ms
Time: 1557560820000 ms
(Hello,1)
(!!!,1)
(World!,1)
Time: 1557560830000 ms

```


Stream Word Count demo in java

```

cleo@vm-xenial0: ~/streams-wordcount
cleo@vm-xenial0:~$ cd streams-wordcount/
cleo@vm-xenial0:~/streams-wordcount$ docker-compose exec kafka bash
bash-4.4# exit
cleo@vm-xenial0:~/streams-wordcount$ kafka-console-consumer.sh --bootstrap-server 192.168.206.131:9092 \
> ^C
cleo@vm-xenial0:~/streams-wordcount$ kafka-console-producer.sh --broker-list 192.168.206.131:9092 --topic streams-plaintext-input
kafka-console-producer.sh: command not found
cleo@vm-xenial0:~/streams-wordcount$ docker-compose exec kafka bash
bash-4.4# kafka-console-producer.sh --broker-list 192.168.206.131:9092 --topic streams-plaintext-input
>hello
>world
>hello
>hello stream
^C
bash-4.4# kafka-console-consumer.sh --bootstrap-server 192.168.206.131:9092 \
> --topic streams-wordcount-output \
> --from-beginning \
> --formatter kafka.tools.DefaultMessageFormatter \
> --property print.key=true \
> --property print.value=true \
> --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
> --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
hello 1
world 1
hello 2
hello 3
stream 1
^C
Processed a total of 5 messages
bash-4.4#

```

实验思考

Kafka

Apache Kafka is a distributed streaming platform. A streaming platform has three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

Kafka Streams

Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in Kafka clusters. It combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the benefits of Kafka's server-side cluster technology.

Kafka provides two APIs for defining flow processing topologies:

1. **Kafka Streams DSL API:** Here we provide some commonly used methods, we can use them directly, without deep operations, such as map, filter, join and so on.
2. **Low-Level Processor API:** If the above-mentioned methods can not meet our business needs, then we use this API to develop our own processing logic, in this API according to our business needs to customize processing logic. This method is customized, so we need to operate some of the underlying processing, which is somewhat complicated to implement.

KStream: It's a kind of flow. As mentioned above, flow exists in the form of key-value pairs. So data is recorded in the form of key-value pairs in KStream. But its characteristic is that data is continuously appended to it, similar to Insert operation in database. Just add data to it. Every data is independent and has nothing to do with other data. Looking at the name also helps us understand that stream is to interpret it as a continuous flow of data that does not cover or update the data. We call this a record stream.

KTable: It is also a stream that records data in the form of key-value pairs. But the main difference between KStream and KStream is that the records are updated continuously according to the key, similar to the data update based on the primary key in the database. Key keeps only the latest record. Looking at the table in the name, you can understand that it is similar to a stateful table, recording only the latest data. We call this update log flow. Well, the data is constantly updated.

参考资料

- <https://data-flair.training/blogs/kafka-docker/>
- <https://kafka.apache.org/10/documentation/streams/quickstart>
- https://gerardnico.com/dit/kafka/stream_wordcount#about
- <http://lanxinglan.cn/2017/10/18/%E5%9C%A8Docker%E7%8E%AF%E5%A2%83%E4%B8%8B%E9%83%A8%E7%BD%B2Kafka/>
- <https://kafka.apache.org/>