**Processing Kafka stream using spark on SBT**

1. **Things required to be downloaded on machine**

Java-1.8-u111

Sbt-1.1.6

Sbt-eclipse-4.7.0

zookeeper-3.4.9

kafka\_2.11-0.10.1.0

1. **Installation Instruction for running the application**

Java

SBT

SBT-Eclipse

SBT-Workspace set up

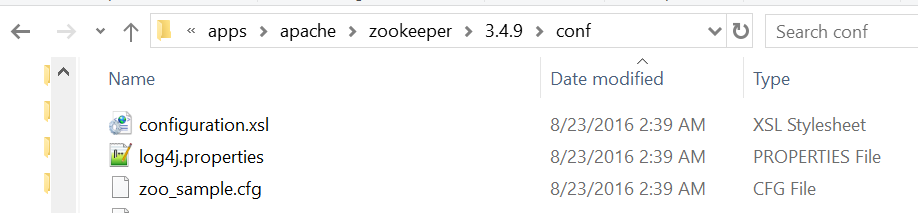
Zookeeper

Kafka

1. **Running Zookeeper Server**

Changing data location for windows.

Access below location



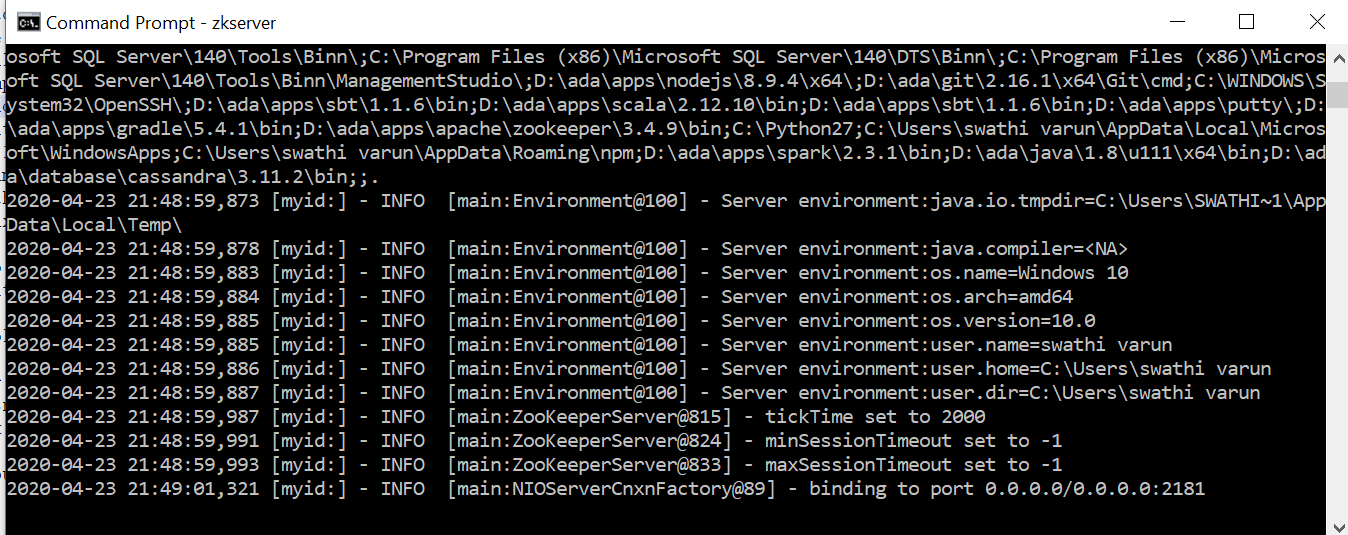
Rename zoo.sample.cfg to zoo.cfg.

Change data location of dataDir= c:\ada\apps\apache\zookeeper\3.4.9\data

Open command prompt and run **zkserver**



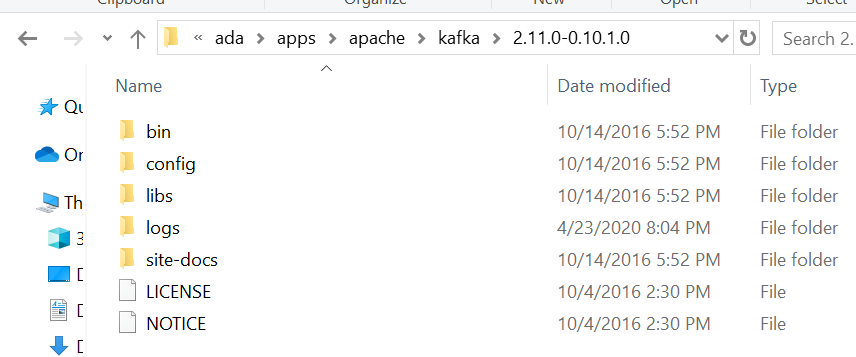
If installation run successfully you will see below logs on command prompt



**Why zookeeper is required for kafka server**

1. Electing a controller. The controller is one of the brokers and is responsible for maintaining the leader/follower relationship for all the partitions. When a node shuts down, it is the controller that tells other replicas to become partition leaders to replace the partition leaders on the node that is going away. Zookeeper is used to elect a controller, make sure there is only one and elect a new one it if it crashes.
2. Cluster membership - which brokers are alive and part of the cluster? this is also managed through ZooKeeper.
3. Topic configuration - which topics exist, how many partitions each has, where are the replicas, who is the preferred leader, what configuration overrides are set for each topic
4. (0.9.0) - Quotas - how much data is each client allowed to read and write
5. (0.9.0) - ACLs - who is allowed to read and write to which topic
6. (old high level consumer) - Which consumer groups exist, who are their members and what is the latest offset each group got from each partition.
7. **Running Kafka Server**

Changing kafka log directory



Edit server.properties file and edit log file

log.dirs=/tmp/kafka-logs” to “log.dir= log

If your ZooKeeper is running on some other machine or cluster you can edit “zookeeper.connect:2181” to your custom IP and port. For this demo, we are using the same machine so there's no need to change. Also the Kafka port and broker.id are configurable in this file. Leave other settings as is.

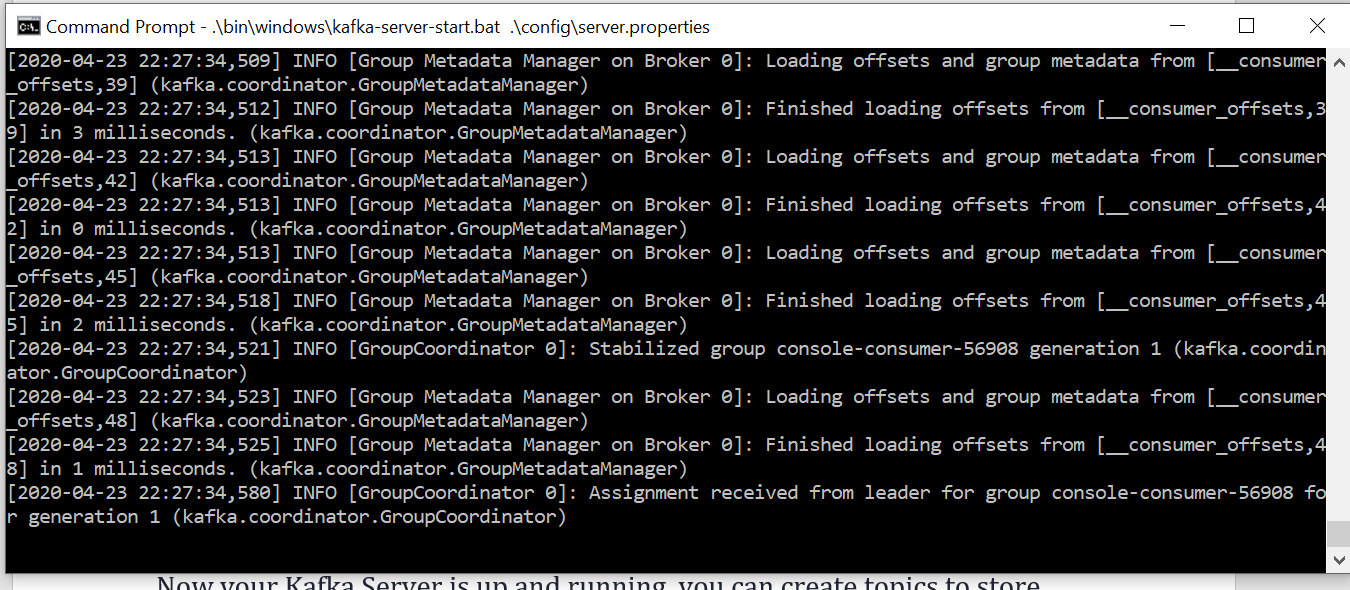
Your Kafka will run on default port 9092 and connect to ZooKeeper’s default port, 2181.(if 2181 port is not available server will provide another port number)

Accessing kafka installation directory and run below command

.\bin\windows\kafka-server-start.bat .\config\server.properties



If everything went fine, your command prompt will look like this:



**Creating Topics**

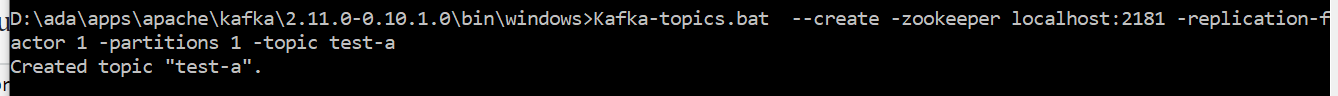
1) Now your Kafka Server is up and running, you can create topics to store messages. Also, we can produce or consume data from Java or Scala code or directly from the command prompt.

2) Open command prompt in the location D:\ada\apps\apache\kafka\2.11.0-0.10.1.0\bin\windows

3) Now lets create a ‘test’ topic in kafka server

Kafka-topics.bat --create –zookeeper localhost:2128 –replication-factor 1 –partitions 1 –topic test-a

Below is the output you would find



**Creating a Producer and Consumer to test server**

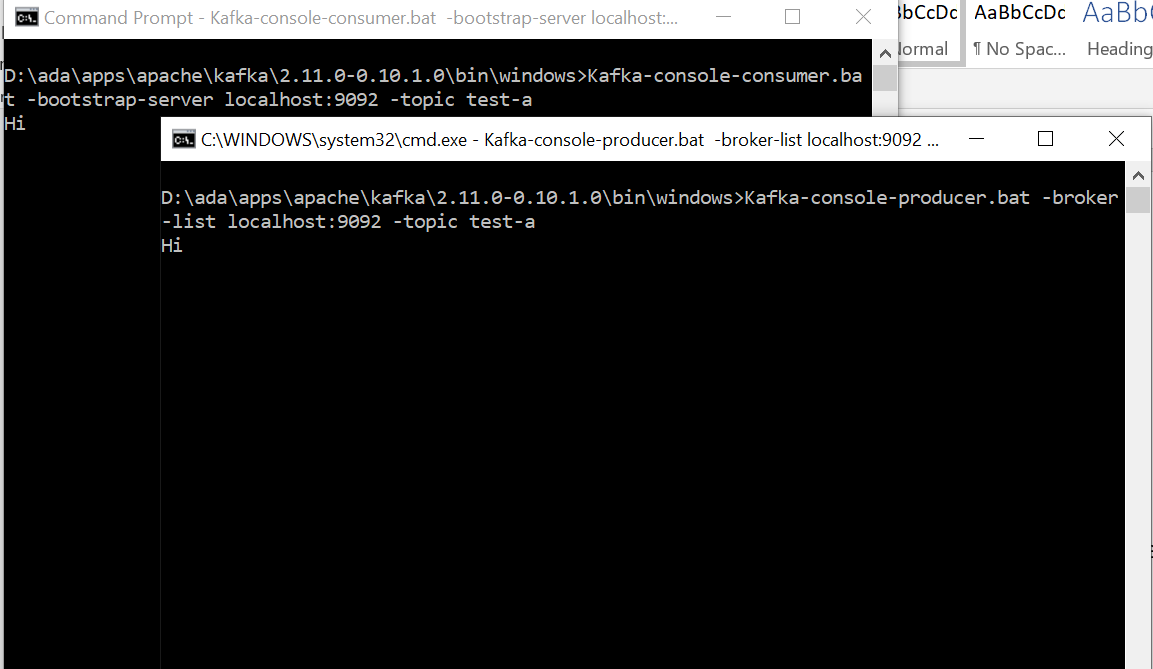
1. Open a new command prompt in the location D:\ada\apps\apache\kafka\2.11.0-0.10.1.0\bin\windows
2. To start a producer type the following command:

Kafka-console-producer.bat –broker-list localhost:9092 –topic test-a

1. Again open a new command prompt in the location D:\ada\apps\apache\kafka\2.11.0-0.10.1.0\bin\windows

Kafka-console-consumer.bat –bootstrap-server localhost:9092 –topic test-a

1. Now you will have two commands prompts as such



**Some Other Useful Commands**

1. List Topics: kafka-topics.bat --list --zookeeper localhost:2181
2. Describe Topic: kafka-topics.bat --describe --zookeeper localhost:2181 --topic [Topic Name]
3. Read messages from the beginning
   1. Before version < 2.0: kafka-console-consumer.bat --zookeeper localhost:2181 --topic [Topic Name] --from-beginning
   2. After version > 2.0:  kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic [Topic Name] --from-beginn
4. Delete Topic: kafka-run-class.bat kafka.admin.TopicCommand --delete --topic [topic\_to\_delete] --zookeeper localhost:2181
5. **Kafka as a Source and Spark Structured streaming as sink**

Maven dependency pom.xml

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<scala-tool-version>2.11</scala-tool-version>

<spark.version>2.3.0</spark.version>

</properties>

<!-- Spark Configuration -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_${scala-tool-version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_${scala-tool-version}</artifactId>

<version>${spark.version}</version>

</dependency>

<!-- Spark Configuration -->

<!-- Kafka Configuration -->

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-10 -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka-0-10\_2.11</artifactId>

<version>2.3.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka -->

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka\_2.11</artifactId>

<version>0.10.1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka-clients</artifactId>

<version>0.10.1.0</version>

</dependency>

<!-- Kafka Configuration -->

Scala Object for Kafka Producer

**package** org.hadoopexam.spark.streaming.kafka

**import** java.util.Properties

**import** org.apache.kafka.clients.producer.KafkaProducer

**import** org.apache.kafka.clients.producer.ProducerRecord

**object** KafkaProducer {

**def** main(args : Array[*String*]):Unit = {

**val** props = **new** Properties()

props.put("bootstrap.servers", "localhost:9092")

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

**val** producer = **new** KafkaProducer[*String*,*String*](props)

**for**(count <- 0 to 10) {

println("Sending Message to Producer")

producer.send(**new** ProducerRecord[*String*, *String*]("test-a", "title "+count.toString +" data from topic"))

Thread.sleep(10000)

}

println("Message sent successfully")

producer.close()

}

}

**Scala Object for Kafka Consumer for basic test**

For Documentation : <https://spark.apache.org/docs/2.3.0/streaming-kafka-0-10-integration.html>

**package** org.hadoopexam.spark.streaming.kafka

**import** \_root\_.kafka.serializer.DefaultDecoder

**import** \_root\_.kafka.serializer.StringDecoder

//import org.apache.spark.streaming.kafka.KafkaUtils

**import** org.apache.spark.storage.StorageLevel

**import** org.apache.spark.streaming.\_

**import** org.apache.spark.SparkConf

**import** org.apache.spark.SparkContext

**import** org.apache.spark.streaming.kafka010.KafkaUtils

**import** org.apache.kafka.common.serialization.StringDeserializer

**import** org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent

**import** org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

**object** BasicKafkaTestTopic {

**def** main(args: Array[*String*]): Unit = {

**val** sparkConf = **new** SparkConf().setAppName("BasicKafkaTestTopic").setMaster("local[2]")

**val** sc = **new** SparkContext(sparkConf)

// prevent INFO logging from pollution output

sc.setLogLevel("ERROR")

// creating the StreamingContext with 5 seconds interval

**val** ssc = **new** StreamingContext(sc, Seconds(5))

**val** kafkaParams = Map[*String*, Object](

"bootstrap.servers" -> "localhost:9092",

"key.deserializer" -> classOf[StringDeserializer],

"value.deserializer" -> classOf[StringDeserializer],

"group.id" -> "org.hadoopexam.spark.streaming.kafka",

"auto.offset.reset" -> "latest",

"enable.auto.commit" -> (**false**: java.lang.Boolean))

**val** topics = Array("test", "test-a")

**val** stream = KafkaUtils.createDirectStream[*String*, *String*](

ssc,

PreferConsistent,

Subscribe[*String*, *String*](topics, kafkaParams))

stream.map(record=>(record.value().toString)).print

ssc.start()

ssc.awaitTermination()

}

}