**KAFKA:**

Apache Kafka is a distributed message broker designed to handle large volumes of real-time data efficiently. Unlike traditional brokers like ActiveMQ and RabbitMQ, Kafka runs as a cluster of one or more servers which makes it highly scalable and due to this distributed nature it has inbuilt fault-tolerance while delivering higher throughput when compared to its counterparts.

Apache Kafka is a distributed event streaming platform that can be used to build high-performance data pipelines, data integration, stream analytics, and mission-critical applications. Users can utilize Kafka Streams in particular to implement end-to-end event streaming. Users can also create and read event streams, as well as import and export data from other systems, using Kafka as a data stream platform.

Apache Kafka is a distributed, highly scalable, elastic, fault-tolerant, and secure data stream platform that can be used on-premises as well as in the cloud. Users can also select between “self-managing their Kafka setups” and using “vendor-managed services” based on the requirements. Kafka is one of the five most active Apache Software Foundation projects, according to the developers, and is trusted by more than 80% of Fortune 100 organizations

**Install Kafka on Ubuntu 20.04:**

**Step 1: Install Java and Bookeeper**

Kafka is written in Java and Scala and requires jre 1.7 and above to run it.

In this step,

**$ sudo apt-get update**

**$ sudo apt-get install default-jre**

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Kafka uses Zookeeper for maintaining the heartbeats of its nodes, maintaining configuration, and most importantly to elect leaders.

**$ sudo apt-get install zookeeperd**

Step 2: Download Apache Kafka

Now, you need to download and extract Kafka binaries in your Kafka user’s home directory. You can create your directory using the following command:

I downloaded latest BINARY release from here <https://kafka.apache.org/downloads>. I used <https://dlcdn.apache.org/kafka/3.0.0/kafka_2.13-3.0.0.tgz>

**Or use below command to download**

sudo wget <https://dlcdn.apache.org/kafka/3.0.0/kafka_2.13-3.0.0.tgz>

**Unpack and move**

**Mkdir** /usr/local/kafka

tar xzf kafka\_2.13-3.0.0.tgz

mv kafka\_2.13-3.0.0 /usr/local/kafka

**Step 4: Configuring Kafka Server**

$ nano ~/kafka/config/server.properties

Add a setting that allows us to delete Kafka topics first. Add the following to the file’s bottom:

delete.topic.enable = true

**edit zookeeper unit file**

sudo vi /etc/systemd/system/zookeeper.service

add this content

[Unit]

Description=Apache Zookeeper server

Documentation=http://zookeeper.apache.org

Requires=network.target remote-fs.target

After=network.target remote-fs.target

[Service]

Type=simple

ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties

ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh

Restart=on-abnormal

[Install]

WantedBy=multi-user.target

**Edit Kafka systemd unit file**

sudo vi /etc/systemd/system/kafka.service

and add the content below. ****Note: You must change JAVA\_HOME=path to your path****

[Unit]

Description=Apache Kafka Server

Documentation=http://kafka.apache.org/documentation.html

Requires=zookeeper.service

[Service]

Type=simple

ExecStart=/usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server.properties

ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh

[Install]

WantedBy=multi-user.target

**Reload the systemd daemon to apply new changes.**

sudo systemctl daemon-reload

**Start zookeeper and kafka**

sudo systemctl start zookeeper

sudo systemctl start kafka

**check kafka status now, it should be running**

sudo systemctl status kafka

The Kafka service has been begun. But if you rebooted your server, Kafka would not restart automatically. To enable the Kafka service on server boot, run the following commands:

**$ sudo systemctl enable zookeeper**

**$ sudo systemctl enable kafka**

### **Step 6: Testing installation**

In this stage, you’ll put your Kafka setup to the test. To ensure that the Kafka server is functioning properly, you will publish and consume a “Hello World” message.

In order to publish messages in Kafka, you must first:

* A producer who allows records and data to be published to topics.
* A person who reads communications and data from different themes.

To get started, make a new topic called ****TutorialTopic****:

$ /usr/local/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic

The****kafka-console-producer.sh**** script can be used to build a producer from the command line. As arguments, it expects the hostname, port, and topic of the Kafka server.

The string “Hello, World” should now be published to the ****TutorialTopic****topic:

$ echo "Hello, World" | /usr/local/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic > /dev/null

Using the ****Kafka-console-consumer.sh****script, establish a Kafka consumer. As parameters, it requests the ZooKeeper server’s hostname and port, as well as a topic name.

Messages from ****TutorialTopic****are consumed by the command below. Note the usage of the —from-beginning flag, which permits messages published before the consumer was launched to be consumed:

$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TutorialTopic --from-beginning

Hello, World will appear in your terminal if there are no configuration issues:

Hello, World

The script will keep running while it waits for further messages to be published. Open a new terminal window and log into your server to try this.  
Start a producer in this new terminal to send out another message:

$ echo "Hello World from Sammy at DigitalOcean!" | ~/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic > /dev/null

This message will appear in the consumer’s output:

Hello, World

Hello World from Sammy at DigitalOcean!

To stop the consumer script, press CTRL+C once you’ve finished testing.

**Implemenataion:**

**Step 1: check scala version**

**$ spark -shell**

**Using Scala version 2.12.15**

**step 2: check pyspark version**

**spark version 3.3.0**

**--packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.3.0**

**refers as: 0-10\_ - streaming**

**scala\_version: 2.12**

**spark version: 3.3.0**

**create topic:**

**note: create new kafka topic whenver needed**

**1> /home/manishmehta/Documents/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic\_v2.1**

**or 2> /home/manishmehta/Documents/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic\_v2**

**or 3> /home/manishmehta/Documents/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic pkttest**

**producer:**

**note: producer produce message through command line.**

**1>**

**echo "Hello, World" | /home/manishmehta/Documents/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic\_v2 > /dev/null**

**2>**

**echo "Hello World from data science team" | /home/manishmehta/Documents/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic TutorialTopic\_v2 > /dev/null**

**consumer:**

**1> /home/manishmehta/Documents/spark-3.3.0-bin-hadoop3/bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.3.0 consumer\_test\_v2.py**

**note:**

**spark = SparkSession \**

**.builder \**

**.appName("SSKafka") \**

**.config("spark.jars.packages","org.apache.spark:spark- sql-kafka-0-10\_2.12:3.3.0") \**

**.getOrCreate()**

**if “.config("spark.jars.packages","org.apache.spark:spark- sql-kafka-0-10\_2.12:3.3.0") “ in creating spark session then direct run python file as**

**$ python consumer.py**

**example:**

**step 1: create kafka topic:**

**using below command:**

**/home/manishmehta/Documents/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TutorialTopic\_v2.1**

**step 2: producer**

**producer\_iris.py**

**python producer\_iris.py**

**“””**

**# -\*- coding: utf-8 -\*-**

**"""prodicer\_iris.ipynb**

**Automatically generated by Colaboratory.**

**Original file is located at**

**https://colab.research.google.com/github/Siddharth1698/Structured-Streaming-Tutorial/blob/main/prodicer\_iris.ipynb**

**"""**

**import pandas as pd**

**from kafka import KafkaProducer**

**from datetime import datetime**

**import time**

**import random**

**import numpy as np**

**# pip install kafka-python**

**#KAFKA\_TOPIC\_NAME\_CONS = "TutorialTopic\_v2.1"**

**KAFKA\_TOPIC\_NAME\_CONS = "pkttest"**

**KAFKA\_BOOTSTRAP\_SERVERS\_CONS = 'localhost:9092'**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("Kafka Producer Application Started ... ")**

**kafka\_producer\_obj = KafkaProducer(bootstrap\_servers=KAFKA\_BOOTSTRAP\_SERVERS\_CONS,**

**value\_serializer=lambda x: x.encode('utf-8'))**

**filepath = "IRIS.csv"**

**flower\_df = pd.read\_csv(filepath)**

**flower\_df['order\_id'] = np.arange(len(flower\_df))**

**flower\_list = flower\_df.to\_dict(orient="records")**

**message\_list = []**

**message = None**

**for message in flower\_list:**

**message\_fields\_value\_list = []**

**message\_fields\_value\_list.append(message["order\_id"])**

**message\_fields\_value\_list.append(message["sepal\_length"])**

**message\_fields\_value\_list.append(message["sepal\_width"])**

**message\_fields\_value\_list.append(message["petal\_length"])**

**message\_fields\_value\_list.append(message["petal\_width"])**

**message\_fields\_value\_list.append(message["species"])**

**message = ','.join(str(v) for v in message\_fields\_value\_list)**

**print("Message Type: ", type(message))**

**print("Message: ", message)**

**kafka\_producer\_obj.send(KAFKA\_TOPIC\_NAME\_CONS, message)**

**time.sleep(1)**

**print("Kafka Producer Application Completed. ")**

**"""**

**step 3: consumer:**

**python consumer\_irispy**

**using below commands:**

**$ /home/manishmehta/Documents/spark-3.3.0-bin-hadoop3/bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.3.0 consumer\_iris.py**

**“””from pyspark.sql import SparkSession**

**from pyspark.sql.functions import \***

**import time**

**kafka\_topic\_name = "pkttest"**

**kafka\_bootstrap\_servers = "localhost:9092"**

**spark = SparkSession \**

**.builder \**

**.appName("Structured Streaming Pkt\_") \**

**.master("local[\*]") \**

**.getOrCreate()**

**spark.sparkContext.setLogLevel("ERROR")**

**# Construct a streaming DataFrame that reads from topic**

**pkt\_df = spark \**

**.readStream \**

**.format("kafka") \**

**.option("kafka.bootstrap.servers", kafka\_bootstrap\_servers) \**

**.option("subscribe", kafka\_topic\_name) \**

**.load()**

**pkt\_df1 = pkt\_df.selectExpr("CAST(value AS STRING)", "timestamp")**

**pkt\_schema\_string = "order\_id INT,sepal\_length DOUBLE,sepal\_length DOUBLE,sepal\_length DOUBLE,sepal\_length DOUBLE,species STRING"**

**pkt\_df2 = pkt\_df1 \**

**.select(from\_csv(col("value"), pkt\_schema\_string) \**

**.alias("flower"), "timestamp")**

**pkt\_df3 = pkt\_df2.select("flower.\*", "timestamp")**

**#pkt\_df3 = pkt\_df2.select("pkt.\*")**

**#summary = pkt\_df3 \**

**# .groupBy("proto") \**

**# .count()**

**#query = summary \**

**query = pkt\_df3 \**

**.writeStream \**

**.trigger(processingTime='15 seconds') \**

**.outputMode("update") \**

**.format("console") \**

**.start()**

**### If you want to try storing the data frames in files.**

**#query = pkt\_df3 \**

**# .writeStream \**

**# .trigger(processingTime='5 seconds') \**

**# .outputMode("append") \**

**# .format("csv") \**

**# .option("checkpointLocation", "/home/raja/kafka/pkt-example") \**

**# .option("path", "/home/raja/kafka/pkt-example") \**

**# .start()**

**query.awaitTermination()**

**“””**

**References:**

[**https://hevodata.com/blog/how-to-install-kafka-on-ubuntu/**](https://hevodata.com/blog/how-to-install-kafka-on-ubuntu/)

[**https://stackoverflow.com/questions/67705738/problem-with-kafka-failed-with-result-exit-code-status-1-failure**](https://stackoverflow.com/questions/67705738/problem-with-kafka-failed-with-result-exit-code-status-1-failure)

[**https://kafka.apache.org/downloads**](https://kafka.apache.org/downloads)

[**https://medium.com/codex/streaming-live-network-packets-data-in-to-spark-streaming-using-kafka-25c8b5f58181**](https://medium.com/codex/streaming-live-network-packets-data-in-to-spark-streaming-using-kafka-25c8b5f58181)

[**https://github.com/raja-surya/Network-Traffic-Kafka-Spark**](https://github.com/raja-surya/Network-Traffic-Kafka-Spark)

[**https://www.analyticsvidhya.com/blog/2021/06/setting-up-real-time-structured-streaming-with-spark-and-kafka-on-windows-os/**](https://www.analyticsvidhya.com/blog/2021/06/setting-up-real-time-structured-streaming-with-spark-and-kafka-on-windows-os/)

[**https://github.com/Siddharth1698/Structured-Streaming-Tutorial**](https://github.com/Siddharth1698/Structured-Streaming-Tutorial)