

## Functions related to RDD using python

In PySpark, Resilient Distributed Datasets (RDDs) are the fundamental data structure representing distributed collections of objects. RDDs can be created from various data sources, and various transformation and action operations can be performed on them.

Some examples of functions related to RDDs using PySpark:

### Creating RDDs:

#### 1. Parallelize:

- Create an RDD from a Python list.

```
from pyspark import SparkContext
```

```
sc = SparkContext("local", "RDDExample")
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)
```

#### 2. TextFile:

- Create an RDD by reading data from a text file.

```
from pyspark import SparkContext
```

```
sc = SparkContext("local", "RDDExample")
rdd = sc.textFile("path/to/textfile.txt")
```

### Transformation Operations:

#### 1. Map:

- Apply a function to each element of the RDD.

```
def square(x):
    return x**2
```

```
rdd_squared = rdd.map(square)
```

#### 2. Filter:

- Filter elements based on a condition.

```
def is_even(x):
    return x % 2 == 0
```

```
rdd_even = rdd.filter(is_even)
```

#### 3. FlatMap:

- Apply a function that returns a sequence for each element and flatten the result.

```
def split_words(x):
    return x.split(" ")
```

```
rdd_flatmap = rdd.flatMap(split_words)
```

### Action Operations:

#### 1. Collect:

- Retrieve all elements of the RDD to the driver program.

```
result = rdd.collect()
```

#### 2. Count:

- Count the number of elements in the RDD.

```
count = rdd.count()
```

#### 3. Reduce:

- Aggregate the elements using a specified function.

from operator import add

sum\_result = rdd.reduce(add)

#### 4. Take:

- Retrieve the first n elements of the RDD.

first\_elements = rdd.take(3)

#### Transformation and Action Combined:

Word Count Example:

- A classic example using flatMap, map, reduceByKey, and collect.

```
words = rdd_flatmap.map(lambda x: (x, 1))
word_counts = words.reduceByKey(add)
result_word_counts = word_counts.collect()
```

The specific operations you choose will depend on your use case and data processing needs.

#### Apache Kafka Installation

Download the Apache kafka using the below mentioned link

[https://archive.apache.org/dist/kafka/2.8.0/kafka\\_2.13-3.6.1.tgz](https://archive.apache.org/dist/kafka/2.8.0/kafka_2.13-3.6.1.tgz)

For install installing scala use the following command

```
$ sudo curl -fL https://github.com/coursier/coursier/releases/latest/download/cs-x86_64-pc-linux.gz | gzip -d > cs && chmod +x cs && ./cs setup
```

```
hadoop@bigadmin:~$ sudo curl -fL https://github.com/coursier/coursier/releases/latest/download/cs-x86_64-
pc-linux.gz | gzip -d > cs && chmod +x cs && ./cs setup
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %         Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0
100 19.8M 100 19.8M    0     0 2370k      0  0:00:08  0:00:08 --:--:-- 4106k
Checking if a JVM is installed
https://github.com/coursier/jvm-index/raw/master/index.json
100.0% [#####] 1.4 MiB (417.8 KiB / s)
No JVM found, should we try to install one? [Y/n] Y
Should we update ~/.profile? [Y/n] Y
Some shell configuration files were updated. It is recommended to close this terminal once the setup comm
and is done, and open a new one for the changes to be taken into account.

Checking if ~/.local/share/coursier/bin is in PATH
Should we add ~/.local/share/coursier/bin to your PATH via ~/.profile? [Y/n] Y

Checking if the standard Scala applications are installed
Installed ammonite
Installed cs
Installed coursier
Installed scala
Installed scalac
Installed scala-cli
Installed sbt
Installed sbt-n
Installed scalafmt
```

```
$ sudo apt install default-jre
```

```
hadoop@servernode:/usr/local/kafka$ java -version
```

```
openjdk version "11.0.21" 2023-10-17
```

```
OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu120.04)
```

```
OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu120.04, mixed mode, sharing)
```

```
$ scala -version
```

```
scala code runner version 3.3.1 -- Copyright 2002-2023, LAMP/EPFL
```

```
$ tar -zxf kafka_2.13-3.61.tgz
$ su
# mv kafka_2.13-3.6.1 /usr/local/kafka
```

```
$ sudo nano /etc/systemd/system/zookeeper.service
```

[Unit]

Description=Apache Zookeeper server  
Documentation=http://zookeeper.apache.org  
Requires=network.target remote-fs.target  
After=network.target remote-fs.target

[Service]

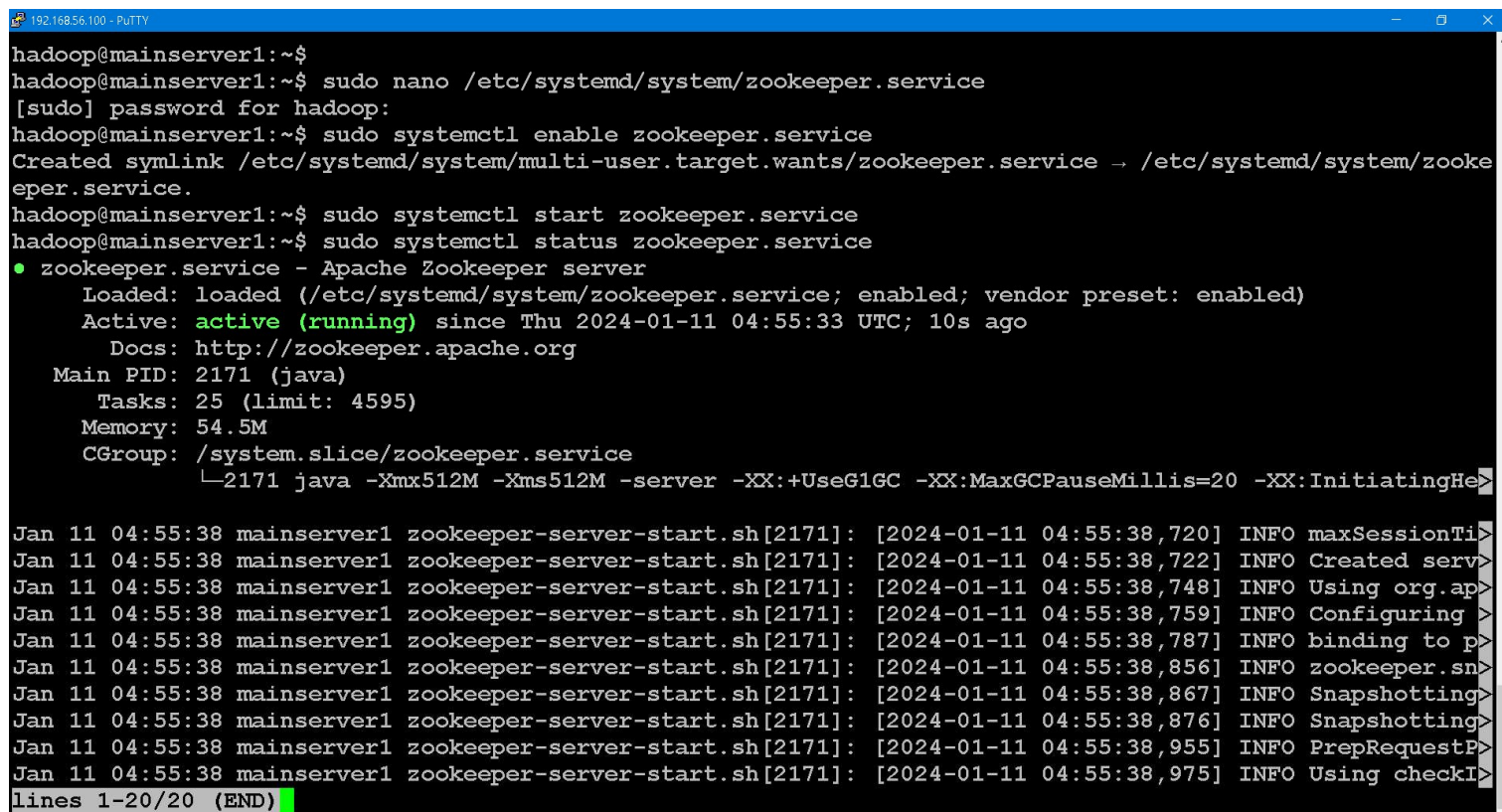
Type=simple  
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties  
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh  
Restart=on-abnormal

[Install]

WantedBy=multi-user.target

<save and exit>

```
$ sudo systemctl enable zookeeper.service
$ sudo systemctl start zookeeper.service
$ sudo systemctl status zookeeper.service
```



```
hadoop@mainserver1:~$
hadoop@mainserver1:~$ sudo nano /etc/systemd/system/zookeeper.service
[sudo] password for hadoop:
hadoop@mainserver1:~$ sudo systemctl enable zookeeper.service
Created symlink /etc/systemd/system/multi-user.target.wants/zookeeper.service → /etc/systemd/system/zookeeper.service.
hadoop@mainserver1:~$ sudo systemctl start zookeeper.service
hadoop@mainserver1:~$ sudo systemctl status zookeeper.service
● zookeeper.service - Apache Zookeeper server
   Loaded: loaded (/etc/systemd/system/zookeeper.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2024-01-11 04:55:33 UTC; 10s ago
     Docs: http://zookeeper.apache.org
    Main PID: 2171 (java)
      Tasks: 25 (limit: 4595)
     Memory: 54.5M
    CGroup: /system.slice/zookeeper.service
            └─2171 java -Xmx512M -Xms512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHe>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,720] INFO maxSessionTi>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,722] INFO Created serv>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,748] INFO Using org.ap>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,759] INFO Configuring>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,787] INFO binding to p>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,856] INFO zookeeper.sn>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,867] INFO Snapshotting>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,876] INFO Snapshotting>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,955] INFO PrepRequestP>
Jan 11 04:55:38 mainserver1 zookeeper-server-start.sh[2171]: [2024-01-11 04:55:38,975] INFO Using checkI>
lines 1-20/20 (END)
```

```
$ sudo nano /etc/systemd/system/kafka.service
```

[Unit]

Description=Apache Kafka Server  
Documentation=http://kafka.apache.org/documentation.html  
Requires=zookeeper.service

[Service]

Type=simple  
Environment="JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64"  
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

<save and exit>

**\$ sudo systemctl enable kafka.service**

**\$ sudo systemctl start kafka.service**

**\$ sudo systemctl status kafka.service**

```
hadoop@mainserver1:~$ sudo nano /etc/systemd/system/kafka.service
hadoop@mainserver1:~$ ^C
hadoop@mainserver1:~$ sudo systemctl enable kafka.service
Created symlink /etc/systemd/system/multi-user.target.wants/kafka.service → /etc/systemd/system/kafka.service.
hadoop@mainserver1:~$ sudo systemctl start kafka.service
hadoop@mainserver1:~$ sudo systemctl status kafka.service
● kafka.service - Apache Kafka Server
   Loaded: loaded (/etc/systemd/system/kafka.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2024-01-11 05:01:42 UTC; 6s ago
     Docs: http://kafka.apache.org/documentation.html
    Main PID: 2598 (java)
      Tasks: 16 (limit: 4595)
     Memory: 32.3M
    CGroup: /system.slice/kafka.service
            └─2598 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:M
Jan 11 05:01:42 mainserver1 systemd[1]: Started Apache Kafka Server.
hadoop@mainserver1:~$
```

**\$ ss -ant**

```
hadoop@mainserver1:~$ ss -ant
State      Recv-Q    Send-Q    Local Address:Port      Peer Address:Port      Process
LISTEN     0          70        127.0.0.1:33060          0.0.0.0:*
LISTEN     0          151        127.0.0.1:3306          0.0.0.0:*
LISTEN     0         4096      127.0.0.53%lo:53        0.0.0.0:*
LISTEN     0          128        0.0.0.0:22             0.0.0.0:*
ESTAB      0          0        192.168.56.100:22      192.168.56.1:12309
ESTAB      0          0        192.168.56.100:22      192.168.56.1:13893
LISTEN     0          50          *:9092                  *:
LISTEN     0          50          *:37861                  *:
LISTEN     0          50          *:2181                   *:
LISTEN     0          50          *:37301                  *:
LISTEN     0         128        [::]:22                 [::]:*
ESTAB      0          0        [::ffff:127.0.0.1]:2181 [::ffff:127.0.0.1]:53066
ESTAB      0          0        [::ffff:127.0.0.1]:9092 [::ffff:127.0.0.1]:48822
ESTAB      0          0        [::ffff:127.0.0.1]:48822 [::ffff:127.0.0.1]:9092
ESTAB      0          0        [::ffff:127.0.0.1]:53066 [::ffff:127.0.0.1]:2181
hadoop@mainserver1:~$
```

- When all the components are installed, we need to create a topic and try to send a message.

**\$ bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic sampleTopic**

```
hadoop@servernode:/usr/local/kafka$ bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic sampleTopic
Created topic sampleTopic.
hadoop@servernode:/usr/local/kafka$
```

**hadoop@servernode:/tmp\$ cd /usr/local/kafka/**

**hadoop@servernode:/usr/local/kafka\$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092**

**sampleTopic**

**hadoop@servernode:/usr/local/kafka\$**

```
hadoop@servernode:/tmp$ cd /usr/local/kafka/
hadoop@servernode:/usr/local/kafka$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
sampleTopic
hadoop@servernode:/usr/local/kafka$ ^C
hadoop@servernode:/usr/local/kafka$
```

## Send and Receive a Message in Kafka

In Kafka, a 'producer' is an application that writes data into topics across different partitions. Applications integrate a Kafka client library to write a message to Apache Kafka. Kafka client libraries are diverse and exist for a myriad of programming languages including Java, Python among others.

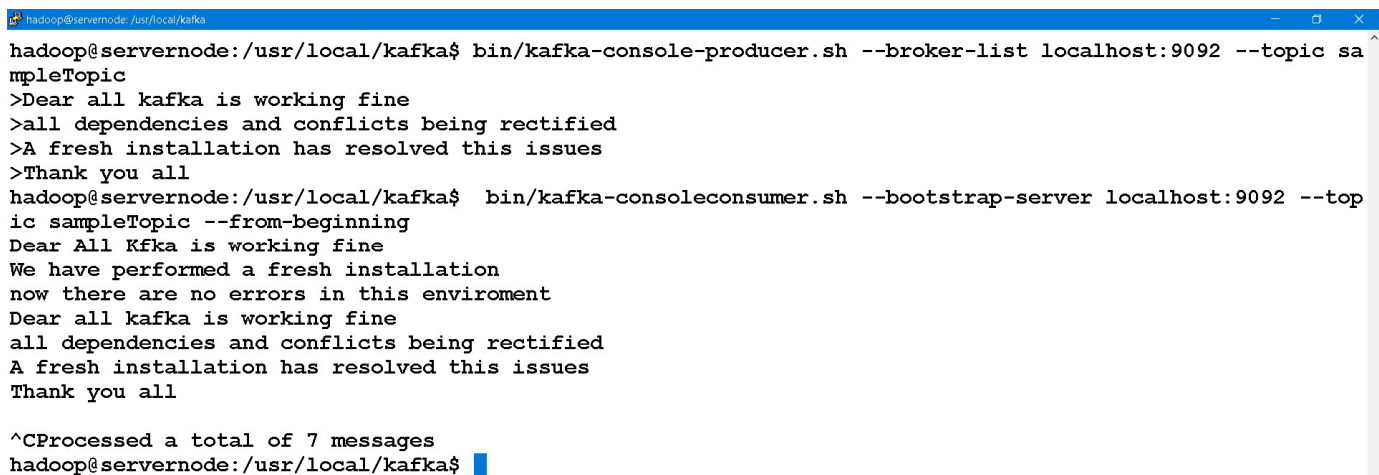
We shall now run the producer and generate a few messages on the console.

**\$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic sampleTopic**

```
hadoop@servernode:/usr/local/kafka$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic sampleTopic
```

```
>Dear all kafka is working fine
>all dependencies and conflicts being rectified
>A fresh installation has resolved this issues
>Thank you all
```

**hadoop@servernode:/usr/local/kafka\$ bin/kafka-consoleconsumer.sh --bootstrap-server localhost:9092 --topic sampleTopic --from-beginning**



```
hadoop@servernode:/usr/local/kafka$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic sampleTopic
>Dear all kafka is working fine
>all dependencies and conflicts being rectified
>A fresh installation has resolved this issues
>Thank you all
hadoop@servernode:/usr/local/kafka$ bin/kafka-consoleconsumer.sh --bootstrap-server localhost:9092 --topic sampleTopic --from-beginning
Dear All Kfka is working fine
We have performed a fresh installation
now there are no errors in this enviroment
Dear all kafka is working fine
all dependencies and conflicts being rectified
A fresh installation has resolved this issues
Thank you all

^CProcessed a total of 7 messages
hadoop@servernode:/usr/local/kafka$
```

## A sample example scripts related to Kafka

Kafka is a distributed streaming platform that is commonly used for building real-time data pipelines and streaming applications. Demonstrating basic producer and consumer functionalities using popular programming languages like Python and Java.

```
$ cd
$ mkdir kafka
$ cd kafka
```

### JAVA Kafka Producer Example:

```
$ vim kafka_producer.java
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;

public class kafka_producer {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

```
// Create Kafka producer
Producer<String, String> producer = new KafkaProducer<>(properties);

// Produce messages to a topic
String topic = "CdacTopic";
producer.send(new ProducerRecord<>(topic, "1", "Hello All"));
producer.send(new ProducerRecord<>(topic, "2", "Welcome to the world of free wares"));
producer.send(new ProducerRecord<>(topic, "3", "Enjoy larning kafka"));

producer.close();
}
}
```

<save and exit>

```
$ javac -cp /usr/local/kafka/libs/kafka-clients-3.6.1.jar kafka_producer.java
```

### **JAVA Kafka Consumer Example:**

```
$ vim kafka_consumer.java
```

```
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class kafka_consumer {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("group.id", "example_group");
        properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

        // Create Kafka consumer
        Consumer<String, String> consumer = new KafkaConsumer<>(properties);
        String topic = "CdacTopic";

        // Subscribe to the topic
        consumer.subscribe(Collections.singletonList(topic));

        // Consume messages from the topic
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            records.forEach(record -> {
                System.out.println("Key: " + record.key() + ", Value: " + record.value());
            });
        }
    }
}
```

<save and exit>

```
hadoop@servernode:~/kafka$ javac -cp /usr/local/kafka/libs/kafka-clients-3.6.1.jar kafka_consumer.java
```

```
hadoop@servernode:~/kafka$ ls -l
total 16
-rw-rw-r-- 1 hadoop hadoop 2576 Jan 12 05:29 kafka_consumer.class
-rw-rw-r-- 1 hadoop hadoop 1331 Jan 12 05:28 kafka_consumer.java
-rw-rw-r-- 1 hadoop hadoop 1172 Jan 12 04:53 kafka_producer.class
-rw-rw-r-- 1 hadoop hadoop 1053 Jan 12 04:51 kafka_producer.java
hadoop@servernode:~/kafka$
```

```
$ sudo apt install python3-pip
```

```
$ sudo pip install kafka-python
```

### **Python - Kafka Producer Example:**

```
$ vim KafkaProducer.py
```

```
from kafka import KafkaProducer
import json

# Set up Kafka producer
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

# Produce messages to a topic
topic = 'CdacTopic'
messages = [
    {'key': '1', 'value': 'Python and Kafka usage'},
    {'key': '2', 'value': 'This should work file'},
    {'key': '3', 'value': 'Executing this very easier'}
]

for message in messages:
    key_bytes = message['key'].encode('utf-8') if message['key'] is not None else None
    producer.send(topic, key=key_bytes, value=message['value'])

producer.close()
```

**<save and exit>**

```
$ python3 KafkaProducer.py
```

```
$ /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic CdacTopic --from-beginning
{ Note: you should be able to view the messages you have edited in the file }
```

### **Python - Kafka Consumer Example:**

```
$ vim KafkaConsumer.py
```

```
from kafka import KafkaConsumer
import json

# Set up Kafka consumer
consumer = KafkaConsumer(
    'CdacTopic',
    group_id=None,
    bootstrap_servers='localhost:9092',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

# Consume messages from the topic
for message in consumer:
    print(f"Key: {message.key}, Value: {message.value}")

consumer.close()
```

**<save and exit>**

```
$ python3 KafkaConsumer.py
```

**Your should be able to view the messages and use <ctl+c> to exit or stop processing**

**Sample output**



```

hadoop@servernode:~/kafka/python$ python3 KafkaConsumer.py
Key: None, Value: b'Welcome to the last of session on bigdata'
Key: None, Value: b'i hope you all enjoyed learning new concpets'
Key: None, Value: b'for kafka i have done an fresh installation'
Key: b'1', Value: b'"Python and Kafka usage"'
Key: b'2', Value: b'"This should work file"'
Key: b'3', Value: b'"Executing this very easier"'
^CTraceback (most recent call last):
  File "KafkaConsumer.py", line 15, in <module>
    for message in consumer:
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/consumer/group.py", line 1193, in __next__
    return self.next_v2()
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/consumer/group.py", line 1201, in next_v2
    return next(self._iterator)
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/consumer/group.py", line 1116, in _message_generator_v2
    record_map = self.poll(timeout_ms=timeout_ms, update_offsets=False)
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/consumer/group.py", line 655, in poll
    records = self._poll_once(remaining, max_records, update_offsets=update_offsets)
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/consumer/group.py", line 702, in _poll_once
    self._client.poll(timeout_ms=timeout_ms)
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/client_async.py", line 602, in poll
    self._poll(timeout / 1000)
  File "/home/hadoop/.local/lib/python3.8/site-packages/kafka/client_async.py", line 634, in _poll
    ready = self._selector.select(timeout)
  File "/usr/lib/python3.8/selectors.py", line 468, in select
    fd_event_list = self._selector.poll(timeout, max_ev)
KeyboardInterrupt

```

<<-- We shall user this later

### Java - Kafka Producer Example:

```

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;

public class KafkaProducerExample {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        // Create Kafka producer
        Producer<String, String> producer = new KafkaProducer<>(properties);

        // Produce messages to a topic
        String topic = "example_topic";
        producer.send(new ProducerRecord<>(topic, "1", "Message 1"));
        producer.send(new ProducerRecord<>(topic, "2", "Message 2"));
        producer.send(new ProducerRecord<>(topic, "3", "Message 3"));

        producer.close();
    }
}

```

<save and exit>

### Java - Kafka Consumer Example:

```

import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

```



```

public class KafkaConsumerExample {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("group.id", "example_group");
        properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

        // Create Kafka consumer
        Consumer<String, String> consumer = new KafkaConsumer<>(properties);
        String topic = "example_topic";

        // Subscribe to the topic
        consumer.subscribe(Collections.singletonList(topic));

        // Consume messages from the topic
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            records.forEach(record -> {
                System.out.println("Key: " + record.key() + ", Value: " + record.value());
            });
        }
    }
}

```

<save and exit>

**We shall user this later -->>**

- **SPARK MLlib ( Spark Machine learning library)**

Apache Spark MLlib (Machine Learning Library) is the machine learning component of the Apache Spark ecosystem.

MLlib provides a scalable and distributed machine learning library that integrates seamlessly with Spark.

It offers a wide range of machine learning algorithms and utilities for various tasks such as classification, regression, clustering, collaborative filtering, dimensionality reduction, and more.

overview of some key components and capabilities of Spark MLlib:

### Components of Spark MLlib:

1. Data Representation:
  - Spark MLlib uses Spark DataFrames as the primary data structure for machine learning tasks. This allows for seamless integration with Spark's distributed data processing capabilities.
2. Transformers and Estimators:
  - Transformers are components that transform one DataFrame into another, e.g., feature transformers.
  - Estimators are algorithms for building models, and they fit on a DataFrame to produce a model (a Transformer).
3. Pipelines:
  - A Pipeline is a sequence of stages (transformers and estimators) that can be combined to form a complete machine learning workflow. Pipelines make it easy to manage complex workflows and ensure consistency.
4. Algorithms:
  - MLlib provides a variety of algorithms for different machine learning tasks:
    - Classification: Logistic Regression, Decision Trees, Random Forests, Naive Bayes, etc.
    - Regression: Linear Regression, Generalized Linear Regression, etc.
    - Clustering: K-Means, Gaussian Mixture Model (GMM), etc.
    - Recommendation: Alternating Least Squares (ALS) for collaborative filtering.
    - Dimensionality Reduction: Principal Component Analysis (PCA).
5. Model Evaluation:
  - MLlib includes tools for evaluating the performance of machine learning models. It provides metrics such as accuracy, precision, recall, F1 score, etc.

Example: Linear Regression with Spark MLlib

Example for Spark MLlib for linear regression:

**# vim spark\_Mllib.py**

```

from pyspark.sql import SparkSession
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

```

```
# Create a Spark session
spark = SparkSession.builder.appName("LinearRegressionExample").getOrCreate()

# Load your data into a DataFrame
data = spark.read.csv("your_data.csv", header=True, inferSchema=True)

# Prepare features and label
feature_cols = data.columns[:-1]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
assembled_data = assembler.transform(data).select("features", "label")

# Split the data into training and testing sets
train_data, test_data = assembled_data.randomSplit([0.8, 0.2], seed=123)

# Create a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="label")

# Fit the model to the training data
model = lr.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)

# Show the results
predictions.select("label", "prediction").show()

# Stop the Spark session
spark.stop()
```

- **Predictive Analysis**

**Predictive Analysis with pyspark and deep learning with Spark involves using the PySpark library for distributed data processing and leveraging Spark's Mllib for machine learning tasks. Additionally, deep learning capabilities can be integrated using spark's deep learning pipelines or other deep learning libraries compatible with spark.**

#### **Predictive Analysis with PySpark:**

**\$ pip instal pyspark**

**The csv data used for this example**

**\$ predective.csv**

```
feature,label
1.2,2.8
2.5,3.4
3.1,4.0
4.0,4.8
4.5,5.2
5.2,5.9
6.0,6.7
6.8,7.2
7.5,8.1
8.2,8.7
```

**<save and exit>**

**\$ vim predictive\_spark.py**

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
spark = SparkSession.builder.appName("PredictiveAnalysis").getOrCreate()
# Assuming you have a DataFrame with features and labels
data = spark.read.csv("/home/hadoop/spark/pyspark/predective.csv", header=True, inferSchema=True)

# Feature Engineering
feature_cols = data.columns[:-1]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
assembled_data = assembler.transform(data).select("features", "label")
```

```
# Linear Regressing
lr = LinearRegression(featuresCol="features", labelCol="label")
model = lr.fit(assembled_data)
```

```
# make predictions
predictions = model.transform(assembled_data)
predictions.show()
```

<save and exit>

# spark-submit predictive\_spark.py

### Sample output

```
cutor driver) (1/1)
24/01/12 09:45:48 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
24/01/12 09:45:48 INFO DAGScheduler: ResultStage 4 (showString at NativeMethodAccessorImpl.java:0) finished in 0.352 s
24/01/12 09:45:48 INFO DAGScheduler: Job 4 is finished. Cancelling potential speculative or zombie tasks for this job
24/01/12 09:45:48 INFO TaskSchedulerImpl: Killing all running tasks in stage 4: Stage finished
24/01/12 09:45:48 INFO DAGScheduler: Job 4 finished: showString at NativeMethodAccessorImpl.java:0, took 0.377029 s
24/01/12 09:45:48 INFO CodeGenerator: Code generated in 69.334801 ms
+-----+-----+-----+
|features|label|prediction|
+-----+-----+-----+
| [1.2] | 2.8 | 2.4402985720467347 |
| [2.5] | 3.4 | 3.5785720467330178 |
| [3.1] | 4.0 | 4.103929035049764 |
| [4.0] | 4.8 | 4.891964517524882 |
| [4.5] | 5.2 | 5.329762007788838 |
| [5.2] | 5.9 | 5.942678494158375 |
| [6.0] | 6.7 | 6.643154478580702 |
| [6.8] | 7.2 | 7.34363046300303 |
| [7.5] | 8.1 | 7.956546949372567 |
| [8.2] | 8.7 | 8.569463435742104 |
+-----+-----+-----+

24/01/12 09:45:48 INFO SparkContext: Invoking stop() from shutdown hook
24/01/12 09:45:48 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/01/12 09:45:48 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
24/01/12 09:45:48 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

### Connecting DB's with Spark

\$ vim MySQL\_PySpark.py

```
from pyspark.sql import SparkSession
```

```
# Step 1: Create a Spark Session
spark = SparkSession.builder.appName("MySQLSparkExample").getOrCreate()
```

```
# Step 2: Define MySQL Connection Properties
mysql_props = {
    "url": "jdbc:mysql://localhost:3306/harshadb",
    "driver": "com.mysql.cj.jdbc.Driver",
    "user": "airflow",
    "password": "airflow"
}
```

```
table_name = "animals"
```

```
# Step 3: Read Data from MySQL into PySpark DataFrame
mysql_df = spark.read.jdbc(url=mysql_props["url"],table=table_name, properties=mysql_props)
```

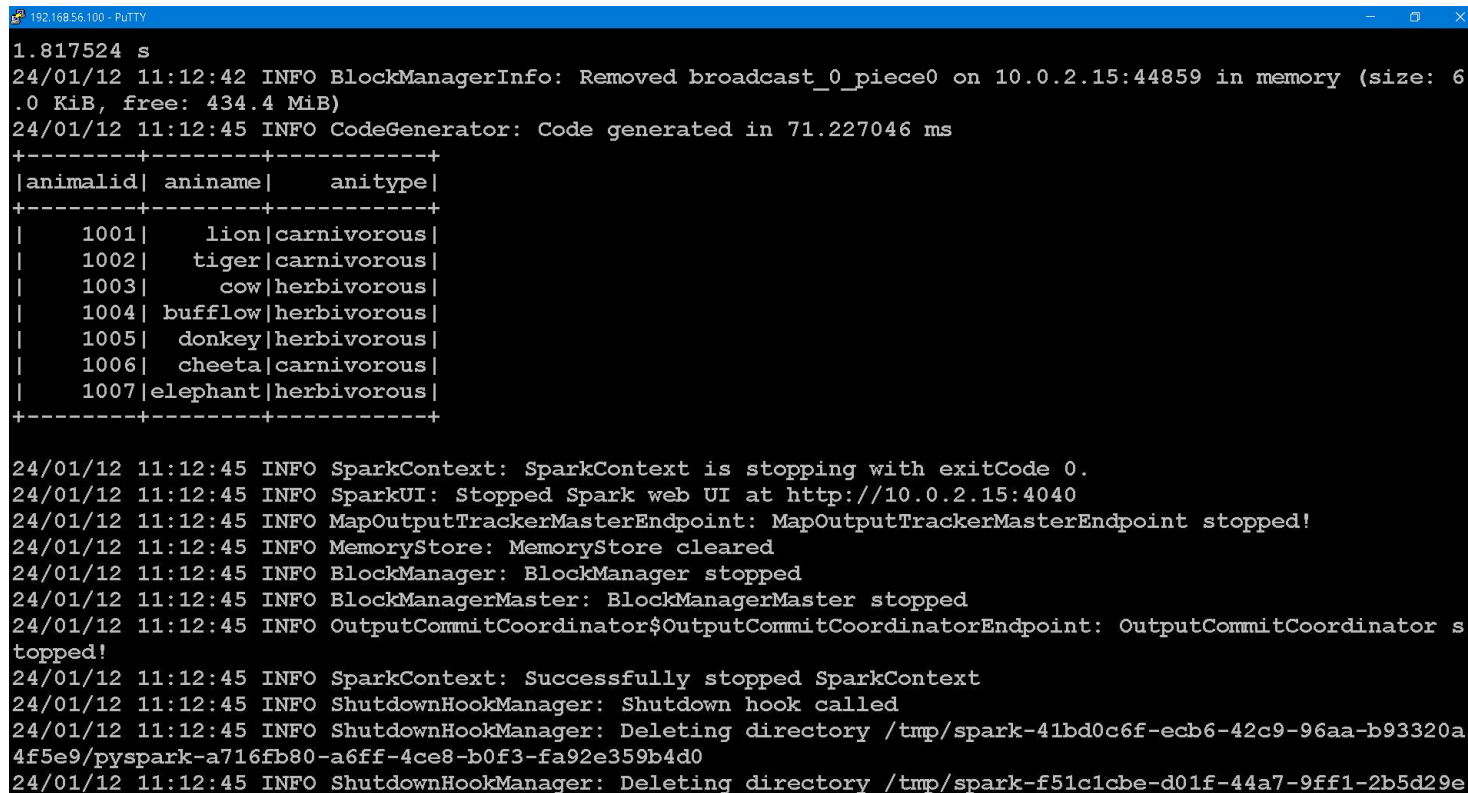
```
# Step 4: Perform Operations on the DataFrame
# For example, display the first few rows
mysql_df.show()
```

```
# Step 5: Write Data from PySpark DataFrame to MySQL (Optional)
# Assuming you have another DataFrame called 'result_df'
# result_df.write.jdbc(url=mysql_props["url"], table="new_table", mode="overwrite", properties=mysql_props)
```

```
# Step 6: Stop the Spark Session
spark.stop()
```

<save and exit>

```
$ spark-submit --jars /usr/share/java/mysql-connector-java-8.1.0.jar MySQL_PySpark.py
```



The screenshot shows a terminal window with the following content:

```
1.817524 s
24/01/12 11:12:42 INFO BlockManagerInfo: Removed broadcast_0_piece0 on 10.0.2.15:44859 in memory (size: 6
.0 KiB, free: 434.4 MiB)
24/01/12 11:12:45 INFO CodeGenerator: Code generated in 71.227046 ms
+-----+-----+-----+
|animalid| aniname|  anitype|
+-----+-----+-----+
|    1001|   lion|carnivorous|
|    1002|  tiger|carnivorous|
|    1003|   cow|herbivorous|
|    1004|bufflow|herbivorous|
|    1005| donkey|herbivorous|
|    1006| cheeta|carnivorous|
|    1007|elephant|herbivorous|
+-----+-----+-----+

24/01/12 11:12:45 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/01/12 11:12:45 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
24/01/12 11:12:45 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/01/12 11:12:45 INFO MemoryStore: MemoryStore cleared
24/01/12 11:12:45 INFO BlockManager: BlockManager stopped
24/01/12 11:12:45 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/12 11:12:45 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator s
topped!
24/01/12 11:12:45 INFO SparkContext: Successfully stopped SparkContext
24/01/12 11:12:45 INFO ShutdownHookManager: Shutdown hook called
24/01/12 11:12:45 INFO ShutdownHookManager: Deleting directory /tmp/spark-41bd0c6f-ecb6-42c9-96aa-b93320a
4f5e9/pyspark-a716fb80-a6ff-4ce8-b0f3-fa92e359b4d0
24/01/12 11:12:45 INFO ShutdownHookManager: Deleting directory /tmp/spark-f51c1cbe-d01f-44a7-9ff1-2b5d29e
```

\*\*\*\*\* End Of Big Data Concept Sessions \*\*\*\*\*