Complete Hadoop Project with Kafka-Pyspark-MongoDB

What is Kafka?

Apache Kafka® is a distributed streaming platform. Kafka is used for building real-time streaming data pipelines that reliably get data between systems or applications or building real-time streaming applications that transform or react to the streams of data. Like many publish-subscribe messaging systems, Kafka maintains feeds of messages in topics. Producers write data to topics and consumers read from topics. Since Kafka is a distributed system, topics are partitioned and replicated across multiple nodes.

- **Topics:** A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.
- **Producers:** Producers publish data to the topics of their choice. The producer is responsible for choosing which record to assign to which partition within the topic. This can be done in a round-robin fashion simply to balance load or it can be done according to some semantic partition function
- **Consumers:** Consumers label themselves with a consumer group name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group. Consumer instances can be in separate processes or on separate machines.

What makes Kafka unique?

Kafka treats each topic partition as a log (an ordered set of messages). Each message in a partition is assigned a unique offset. Kafka does not attempt to track which messages were read by each consumer and only retain unread messages; rather, Kafka retains all messages for a set amount of time, and consumers are responsible to track their location in each log. Consequently, Kafka can support a large number of consumers and retain large amounts of data with very little overhead.

Download address: https://www.apache.org/dyn/closer.cgi?path=/kafka/2.2.0/kafka_2.12-2.2.0.tgz

What is Zookeeper?

Apache Zookeeper is an open source distributed coordination service that helps you manage a large set of hosts. Management and coordination in a distributed environment are tricky. Zookeeper automates this process and allows developers to focus on building software features rather worry about the distributed nature of their application.

Zookeeper helps you to maintain configuration information, naming, group services for distributed applications. It implements different protocols on the cluster so that the application should not implement on their own. It provides a single coherent view of multiple machines.

Download address: https://www.apache.org/dyn/closer.cgi/zookeeper/

Let’s Start

Step 1:

Download the 2.2.0 release and un-tar it.

> tar -xzf kafka_2.12-2.2.0.tgz

> cd kafka_2.12-2.2.0

Step 2:

Kafka uses ZooKeeper so you need to first start a ZooKeeper server

> bin/zookeeper-server-start.sh config/zookeeper.properties

Now start the Kafka server:

> bin/kafka-server-start.sh config/server.properties

STEP 3:

Let's create a topic named "test" with a single partition

> bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test

STEP 4:

Send some messages. Run the producer and then type a few messages into the console to send to the server.

> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

This is a message

This is another message

STEP 5:

Kafka also has a command line consumer that will dump out messages to standard output

> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

This is a message

This is another message

What is Spark (Pyspark)?

Spark is a general-purpose distributed data processing engine that is suitable for use in a wide range of circumstances. On top of the Spark core data processing engine, there are libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application. Programming languages supported by Spark include: Java, Python, Scala, and R. Application developers and data scientists incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, IoT, or financial systems, and machine learning tasks.

STEP 1:

Install Python

STEP 2:

Download & Install Spark with pip

> pip install pyspark

STEP 3:

Your ~/.bash_profile file should be like that;

```
export PATH=$PATH:/usr/local/mongodb/bin
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/
export SPARK_HOME=/Users/apple/spark
export PATH="$SPARK_HOME/bin:$PATH"
export PATH=$JAVA_HOME/bin:$SBT_HOME:$SBT_HOME/lib:$SCALA_HOME/bin:$SCALA_HOME/lib:$PATH
export PATH=$JAVA_HOME/bin:$SPARK_HOME:$SPARK_HOME/bin:$SPARK_HOME/sbin:$PATH
export PATH="/usr/local/opt/python/libexec/bin:$PATH"
PATH="/Library/Frameworks/Python.framework/Versions/3.7/bin:${PATH}"
export PATH
PATH="/Library/Frameworks/Python.framework/Versions/2.7/bin:${PATH}"
export PATH
# Setting PATH for Python 2.7
# The original version is saved in .bash_profile.pysave
PATH="/Library/Frameworks/Python.framework/Versions/2.7/bin:${PATH}"
export PATH

export PYSPARK_PYTHON=python2

# Setting PATH for Python 3.7
# The original version is saved in .bash_profile.pysave
PATH="/Library/Frameworks/Python.framework/Versions/3.7/bin:${PATH}"
export PATH



test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

~
```

What is MongoDB?

MongoDB is a document-oriented NoSQL database used for high volume data storage which instead of having data in a relational type format, it stores the data in documents. MongoDB is a database which came into light around the mid-2000s. It falls under the category of a NoSQL database.

Download from https://www.mongodb.com/download-center

LET'S START CODING

Step 1:

Configure your zoo.cfg from Zookeeper/conf

Set a dir for zookeeper data with "dataDir=" line

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=C:\zookeeper\apache-zookeeper-3.5.5-bin\data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
#
http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc
_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
```

Default Zookeeper port is 2181

Step 2:

Start zookeeper with

> bin/zookeeper-server-start.sh config/zookeeper.properties

Step 3:

Configure your settings for Kafka server in kafka_2.12-2.2.0\config\server.properties

Set your Listener : “`listeners=PLAINTEXT://localhost:9092`”

And start server with bin/kafka-server-start.sh config/server.properties

Step 4:

Unzip KafkaProducer.zip and run SipringBootKafkaProducerExampleApplication.java to start Producer.

UserResource.java generates random values for Book object and sends it to the Topic.

```java
@RestController
@RequestMapping("kafka")
public class UserResource {
    private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss"); // Date format
    static String numofro; // number of rooms
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @GetMapping("/")
    public String getnum() { // Gets number of room which sent from test.java
        return numofro;
    }

    @GetMapping("/{name:.+}")
    public String post(@PathVariable("name") final String name) { // this Get request generates random values and sent it to the topic

        Timestamp timestamp = new Timestamp(System.currentTimeMillis()); //current time

        int numOfRoom = Integer.parseInt(name);
        int rate[] = {1,1,1,1,1,1,1,1,0,0}; // rate of sensors

        for(int i = 0 ; i < numOfRoom ; i ++) {
            int rnd1 = new Random().nextInt(rate.length);
            int rnd2 = new Random().nextInt(rate.length);
            double temp = 25.0;
            double max = 2.0;
            double min = -2.0;

            Random r = new Random();
            temp = temp + (min + (max - min) * r.nextDouble()); //random temperature
            String roomnum = "room" + (i+1); // room names : room1, room2 ...

            String full = roomnum + "," + rate[rnd1] + "," + rate[rnd2] + "," + sdf.format(timestamp) + "," + temp;
            //generated string. It will be send to the kafka topic

            kafkaTemplate.send(roomnum, full); //send function
        }
```

Step 5:

Unzip test.zip and run test.java

This script ask you to number of rooms to simulate sensors and sent to Producer with a HTTP GET.

This HTTP GET Request makes UserResource.java run continuously.

```java
import java.io.BufferedReader;

public class test {
    static double temp = 28.0;

    private final String USER_AGENT = "Mozilla/5.0";

    public static void main(String[] args) throws Exception {


        test http = new test();

        System.out.println("Number of rooms: "); //asks user number of rooms
        Scanner scan = new Scanner(System.in);
        String num = scan.next();

        System.out.println("Testing 1 - Send Http GET request");


        Timer t = new Timer();
        t.schedule(new TimerTask() { // Timer class to send GET request to the kafka in every 7 seconds
            @Override
            public void run() {
                try {
                    http.sendGet(num);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }, 0, 7000); // 7 secs


    }
```

```java
    private void sendGet(String num) throws Exception { // GET Request for running UserResource.java

        String url = "http://10.10.10.50:8081/kafka/" + num; // number of rooms sent

        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
e
        // optional default is GET
        con.setRequestMethod("GET");

        //add request header
        con.setRequestProperty("User-Agent", USER_AGENT);

        int responseCode = con.getResponseCode();
        System.out.println("\nSending 'GET' request to URL : " + url);
        System.out.println("Response Code : " + responseCode);

        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();

        //print result
        System.out.println(response.toString());

    }

}
```

Step 6:

Run multconsumer.py with

> python multconsumer.py

```python
from kafka import KafkaConsumer
import os
from kafka import *
import time
import random
from datetime import datetime, date
import requests

sn = requests.get('http://10.10.10.50:8081/kafka/')     #This Line Gives Number of Sensor and Room with GET REQUEST
print("Number Of Room:" + sn.content)
cont = int(sn.content)
#print(['room{}'.format(i) for i in range(1, cont+1)])

consumer = KafkaConsumer(bootstrap_servers='10.10.10.50:9092',      #Defined IP for Consumer
                         auto_offset_reset='latest')
consumer.subscribe(['room{}'.format(i) for i in range(1, cont+1)])

# !!PRODUCER -->

mykafka = KafkaClient("10.10.10.50:9092")       #Produce Datas again to consume for DB

producer = SimpleProducer(mykafka)
```

```python
##PARSING PROCESSES
class TempData:
    def parse(self, line):
        fields = line.split(',')
        self.roomname = fields[0]
        self.doorstatus = fields[1]
        self.sensor = fields[2]
        self.timestamp = fields[3]
        self.temperature= fields[4]

        return self
    def repr(self):
        return "roomname = %s,doorstatus = %s ,sensor = %s, timestamp = %s, temperature = %s" % (self.roomname, self.doorstatus,

    def to_json(self):
        return '{"roomname":%s","doorstatus":\"%s\","sensor":\"%s\","timestamp":\"%s\","temperature":"%s}' % (self.roomname,self.



datafile = open ("output.txt", "a", 0)

#Writing to OUTPUT.TXT and CONVERTING TO JSON for data transfer

for message in consumer:
    line = str(message).split("value=")
    first_part = line[1].split("\'")
    data = first_part[1].split("\'")
    real = data[0].split("\"")
    datafile.write(real[1] + "\n")
    print(real[1])

    json_string = TempData().parse(data[0])
    json_data = json_string.to_json()
    #print(json_data)

    producer.send_messages("delta", json_data)  #FUNC => Produce the data
```

Part 7:

We have to run consumerAll.py with

> python consumerAll.py

To send incoming message to the database from kafka which produced by multconsumer.py.

```
from kafka import KafkaConsumer
from pymongo import MongoClient
from json import loads
import json

##THIS CONSUMER TAKES EVERY TYPE OF DATA NON-DIVIDED AND SEND THEM TO DB COLLECTION

consumer2 = KafkaConsumer(
    'delta',                            ##TOPIC THAT CONTAINS NON-DIVIDED AND NOT-PROCESSED DATA
    bootstrap_servers=['10.10.10.50:9092'],
    auto_offset_reset='latest',
    enable_auto_commit=True,
    group_id='my-group2',
    value_deserializer=lambda x: loads(x.decode('utf-8'))) #DECODE INCOMING JSON

client2 = MongoClient('localhost:27017')       ##CONNECTS Mongodb client
collection2 = client2.admin.allData

for message2 in consumer2:              ##TAKES MASSAGES AS A DATA AND SEND IT TO DB ONE-BY-ONE
    message2 = message2.value
    # print(message2)
    collection2.insert_one(message2)  # INSERT TO THE DATABASE
    print('{} added to {}'.format(message2, collection2))
```

Part 8:

Run Multnew_consumer.py to process incoming data from a text file and produces data about changes and averages of temperatures.

Sends this data to alpha topic to be consumed by consumerAct.py

```
from pyspark import SparkContext
import time
import os
from datetime import datetime
from kafka import *
import random
import requests
import json



mykafka = KafkaClient("10.10.10.50:9092")      #This Line Gives Number of Sensor and Room with GET REQUEST

producer = SimpleProducer(mykafka)
sn = requests.get('http://10.10.10.50:8081/kafka/') #Defined IP for Consumer

print("Number Of Room:" + sn.content)
cont = int(sn.content)

##PARSING PROCESSES

class TempData:
    def parse(self, line):
        fields = line.split(',')
        self.roomname = fields[0]
        self.doorstatus = fields[1]
        self.sensor = fields[2]
        self.timestamp = fields[3]
        self.temperature= fields[4]
        return self

    def __repr__(self):
        return 'roomname = %s,doorstatus = %s ,sensor = %s, timestamp = %d, temperature = %s' % (self.roomname,self.doorstatus,se

    def to_json(sel roomname: str
        return '{ "roomname":"%s","doorstatus":"%s" ,"sensor":"%s", "timestamp":"%d"," "temperature":"%s"}'%(self.roomname,self.d
```

```
count = 0
sc = SparkContext(appName="spark_temperature_processor")   #Defined Spark Context

##STARTING OUR RULES
while 1==1:

    stationData = sc.textFile("output.txt") ##PULLING DATA FROM OUTPUT.TXT


    data = stationData.map(lambda x: x.split('\n')) # EXMP: [u'1,2016-05-12 19:28:33.875937,36']
    split_data = data.map(lambda x: x[0].split(',')) #EXMP :[u'1', u'2016-05-12 19:28:33.875937', u'36']

    joint_data_class = split_data.map(lambda x: str(x[0]) + "," + str(x[1]) + "," + str(x[2]) + "," + str(x[3] + "," + str(x[4]))

    ##PARSING DATAS FROM OUTPUT.TXT LINE BY LINE ACCORDING TO ROOM NUMBER

    i = 0
    while i < cont:

        lastTime = (data.map(lambda x: x[0].split(',')).map(lambda x: str(x[3]))).collect()[-1]

        all_sensor = data.map(lambda x: x[0].split(',')).map(lambda x: float(x[2]))
        all_temp = data.map(lambda x: x[0].split(',')).map(lambda x: float(x[4]))
        i += 1


    ##TAKING SUM OF TEMPS FROM PARSED DATA

    i = -cont
    sumOftemps = 0
    while i < 0:
        sumOftemps = sumOftemps + float((data.map(lambda x: x[0].split(',')).map(lambda x: str(x[4]))).collect()[i])
        i += 1
    number_of_entries = data.count()
    avgSample = sumOftemps / cont
```

```
    sum_temp = float(all_temp.sum())

    avg_All = sum_temp / number_of_entries
    change = (avg_All - avgSample) / 100

    print("Avarage = " + str(avgSample))
    print("Change = %" + str(change))
    i = -cont
    x = 1

    while i < 0:
        temp_temp = float((data.map(lambda x: x[0].split(',')).map(lambda x: str(x[4]))).collect()[i])
        print("Room" + str(x)+ " = " + str(temp_temp))
        i += 1
        x += 1

    avg_temp = sum_temp / number_of_entries
    timestamp2 = lastTime

    ##CREATES JSON DATA AND SEND IT TO ANOTHER CONSUMER TO SEND DB

    data_alpha = "{\"timestamp\":\"" + str(timestamp2) + "\"," + "\"Number Of Room\":\"" + str(cont) + "\"," + "\"Temp-Change\":\
    producer.send_messages("alpha",data_alpha)
    time.sleep(2)
```

PART 9:

We have to run consumerAct.py with

> python consumerAct.py

To send incoming message to the database from kafka which produced by multnew_consumer.py

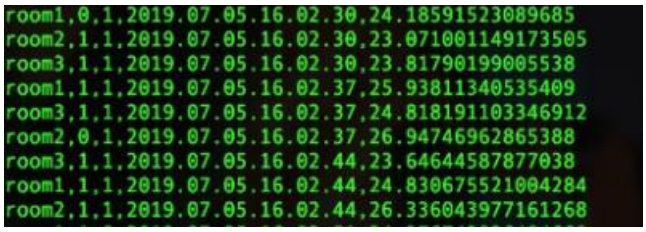
```
from kafka import KafkaConsumer
from pymongo import MongoClient
from json import loads
import json


##THIS CONSUMER TAKES EVERY TYPE OF DATA PROCESSED AND DIVIDED AND SEND THEM TO DB COLLECTION

consumer2 = KafkaConsumer(
    'alpha',            ##TOPIC THAT CONTAINS DIVIDED AND PROCCESED DATA
    bootstrap_servers=['10.10.10.50:9092'],
    auto_offset_reset='latest',
    enable_auto_commit=True,
    group_id='my-group',
    value_deserializer=lambda x: loads(x.decode('utf-8')))


client2 = MongoClient('localhost:27017')            ##CONNECTS Mongodb client
collection2 = client2.admin.Activities


for message2 in consumer2:              ##TAKES MASSAGES AS A DATA AND SEND IT TO DB ONE-BY-ONE
    message2 = message2.value
    #print(message2)
    collection2.insert_one(message2)
    print('{} added to {}'.format(message2, collection2))
```

Conclusion:

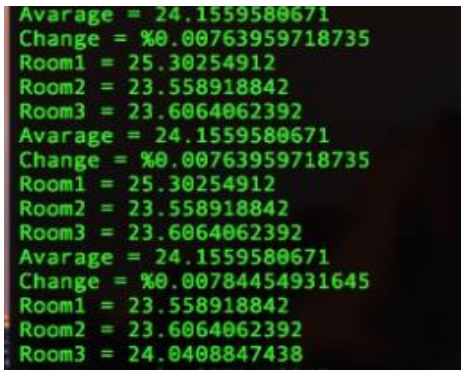
As you can see ,we can analyze and scale real-time streaming with Spark. Also we use Apache Kafka to send this data or to encrypt topics , to create and edit message queues .

Our output examples are down below ;

multconsumer.py Screenshot:



multnew_consumer.py Screenshot:



Screenshot of acquired data from mongoDB which consumed by consumerAct.py:

Activities

	_id ObjectId	Temp-Change String	timestamp String	Number Of Room String	Avg_Temp String
21	5d1f078fb4b5596612c36318	"0.00448590089695"	"2019.07.05.11.17.13"	"4"	"24.4611715756"
22	5d1f0791b4b5596612c36319	"-0.00194009304602"	"2019.07.05.11.17.20"	"4"	"25.1044589462"
23	5d1f0794b4b5596612c3631a	"-0.00194009304602"	"2019.07.05.11.17.20"	"4"	"25.1044589462"
24	5d1f0796b4b5596612c3631b	"-0.00194009304602"	"2019.07.05.11.17.20"	"4"	"25.1044589462"
25	5d1f0799b4b5596612c3631c	"-0.00409100859971"	"2019.07.05.11.17.27"	"4"	"25.3209960876"
26	5d1f079bb4b5596612c3631d	"-0.00409100859971"	"2019.07.05.11.17.27"	"4"	"25.3209960876"
27	5d1f079eb4b5596612c3631e	"-0.00409100859971"	"2019.07.05.11.17.27"	"4"	"25.3209960876"
28	5d1f07a0b4b5596612c3631f	"0.000260995619673"	"2019.07.05.11.17.34"	"4"	"24.8857037658"
29	5d1f07a3b4b5596612c36320	"0.000260995619673"	"2019.07.05.11.17.34"	"4"	"24.8857037658"
30	5d1f07a5b4b5596612c36321	"0.000260995619673"	"2019.07.05.11.17.34"	"4"	"24.8857037658"
31	5d1f07a8b4b5596612c36322	"0.00475632109456"	"2019.07.05.11.17.41"	"4"	"24.4345023338"
32	5d1f07aab4b5596612c36323	"0.00475632109456"	"2019.07.05.11.17.41"	"4"	"24.4345023338"