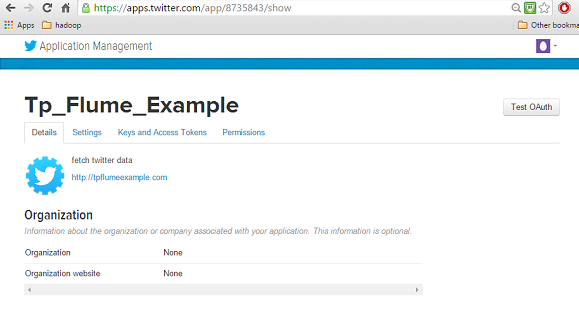
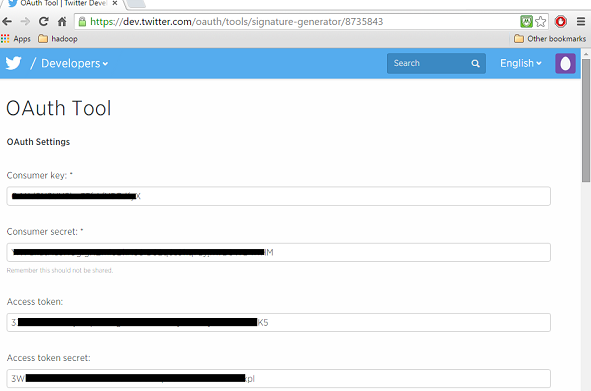
**Task 1**

****

****

Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r

e3496499ecb8d220fba99dc5ed4c99c8f9e33bb1

Compiled with protoc 2.5.0

From source with checksum 18e43357c8f927c0695f1e9522859d6a

This command was run using /home/Hadoop/hadoop/share/hadoop/common/hadoop-common-2.6.0.jar

Due to so many problem I have decided to use Hadoop 2.6.0 from research it is more stable and seems to be ok.

Every time I got problem within .

Browse through the **sbin** directory of Hadoop and start yarn and Hadoop dfs (distributed file system) as shown below.

cd /$Hadoop\_Home/sbin/

$ start-dfs.sh

localhost: starting namenode, logging to

/home/Hadoop/hadoop/logs/hadoop-Hadoop-namenode-localhost.localdomain.out

localhost: starting datanode, logging to

/home/Hadoop/hadoop/logs/hadoop-Hadoop-datanode-localhost.localdomain.out

Starting secondary namenodes [0.0.0.0]

Starting secondarynamenode, logging to

/home/Hadoop/hadoop/logs/hadoop-Hadoop-secondarynamenode-localhost.localdomain.out

$ start-white\_check\_mark.sh

Starting resourcemanager, logging to

/home/Hadoop/hadoop/logs/ white\_check\_mark-Hadoop-resourcemanager-localhost.localdomain.out

localhost: starting nodemanager, logging to

/home/Hadoop/hadoop/logs/eyes-Hadoop-nodemanager

We have to configure the source, the channel, and the sink using the configuration file in the **conf** folder.

# Naming the components on the current agent.

TwitterAgent.sources = Twitter

TwitterAgent.channels = MemChannel

TwitterAgent.sinks = HDFS

# Describing/Configuring the source

TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource

TwitterAgent.sources.Twitter.consumerKey = OAuth consumer key

TwitterAgent.sources.Twitter.consumerSecret = OAuth consumer secret

TwitterAgent.sources.Twitter.accessToken = OAuth consumer key access token

TwitterAgent.sources.Twitter.accessTokenSecret = OAuth consumer key access token secret

TwitterAgent.sources.Twitter.keywords = point,java, bigdata, mapreduce, mahout, hbase, nosql

# Describing/Configuring the sink

TwitterAgent.sinks.HDFS.type = hdfs

TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/Hadoop/twitter\_data/

TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream

TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text

TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000

TwitterAgent.sinks.HDFS.hdfs.rollSize = 0

TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

# Describing/Configuring the channel

TwitterAgent.channels.MemChannel.type = memory

TwitterAgent.channels.MemChannel.capacity = 10000

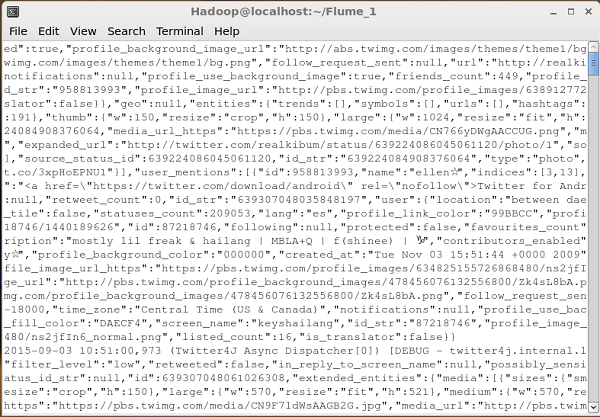
TwitterAgent.channels.MemChannel.transactionCapacity = 100

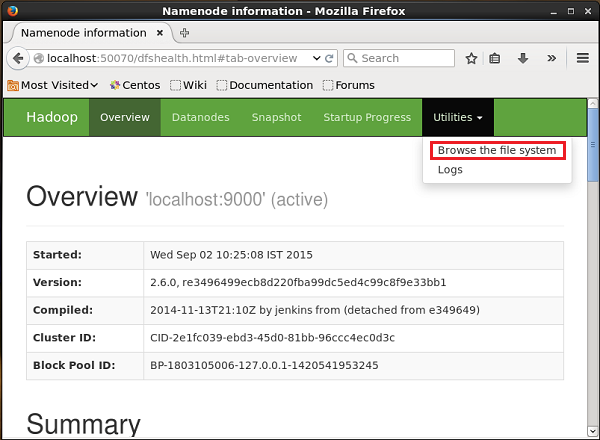
Browse through the Flume home directory and execute the application as shown below.

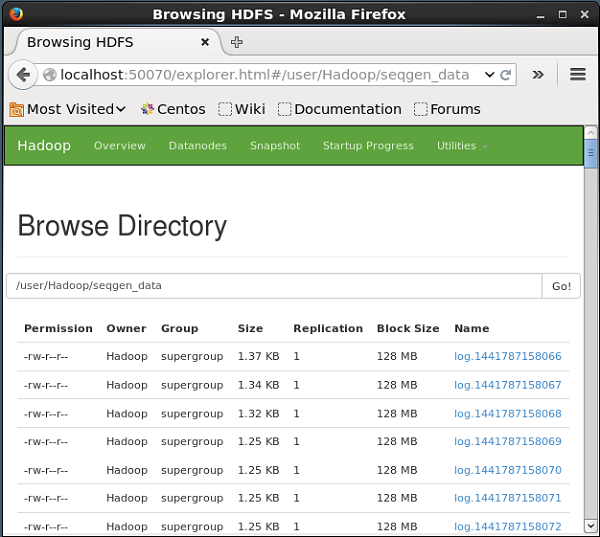
$ cd $FLUME\_HOME

$ bin/flume-ng agent --conf ./conf/ -f conf/twitter.conf

Dflume.root.logger=DEBUG,console -n TwitterAgent







**Task 2**

The following is the syntax for the export command.

$ sqoop export (generic-args) (export-args)

$ sqoop-export (generic-args) (export-args)

The employee data is available in **emp\_data** file in ‘emp/’ directory in HDFS. The **emp\_data** is as follows.

1201, Chris, manager, 50000, TP

1202, John, preader, 50000, TP

1203, Kriti, php dev, 30000, AC

1204, Paul, php dev, 30000, AC

1205, Christina, admin, 20000, TP

1206, Sarah p, grp des, 20000, GR

It is mandatory that the table to be exported is created manually and is present in the database from where it has to be exported.

The following query is used to create the table ‘employee’ in mysql command line.

$ mysql

mysql> USE db;

mysql> CREATE TABLE employee (

id INT NOT NULL PRIMARY KEY,

name VARCHAR(20),

deg VARCHAR(20),

salary INT,

dept VARCHAR(10));

The following command is used to export the table data (which is in **emp\_data** file on HDFS) to the employee table in db database of Mysql database server.

$ sqoop export \

--connect jdbc:mysql://localhost/db \

--username root \

--table employee \

--export-dir /emp/emp\_data

The following command is used to verify the table in mysql command line.

mysql>select \* from employee;

If the given data is stored successfully, then you can find the following table of given employee data.

+------+--------------+-------------+-------------------+--------+

| Id | Name | Designation | Salary | Dept |

+------+--------------+-------------+-------------------+--------+

| 1201 | Chris | manager | 50000 | TP |

| 1202 | John | preader | 50000 | TP |

| 1203 | Kriti | php dev | 30000 | AC |

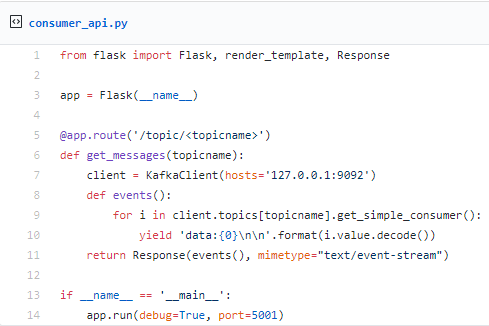
| 1204 | Paul | php dev | 30000 | AC |

| 1205 | Christina | admin | 20000 | TP

| 1206 | Sarah p | grp des | 20000 | GR |

+------+--------------+-------------+-------------------+--------+

**Task 3**

****

|  |
| --- |
| def fetch\_raw(recipe\_url): |
|  | html = None |
|  | print('Processing..{}'.format(recipe\_url)) |
|  | try: |
|  | r = requests.get(recipe\_url, headers=headers) |
|  | if r.status\_code == 200: |
|  | html = r.text |
|  | except Exception as ex: |
|  | print('Exception while accessing raw html') |
|  | print(str(ex)) |
|  | finally: |
|  | return html.strip() |
|  |  |
|  |  |
|  | def get\_recipes(): |
|  | Consumer |
|  |  |
|  | try: |
|  | r = requests.get(url, headers=headers) |
|  | if r.status\_code == 200: |
|  | html = r.text |
|  | consumer = Beautiful (html, 'lxml') |
|  | links = consumer.select('.fixed-consumer-card\_\_h3 a') |
|  | idx = 0 |
|  | for link in links: |
|  |  |
|  | sleep(2) |
|  | consumer = fetch\_raw(link['href']) |
|  | .append(consumer) |
|  | idx += 1 |
|  | if idx > 2: |
|  | break |
|  | except Exception as ex: |
|  | print('Exception in consumer') |
|  | print(str(ex)) |
|  |  |

The \_\_main\_\_ will look like below:

if \_\_name\_\_ == '\_\_main\_\_':  
 headers = {  
 'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36',  
 'Pragma': 'no-cache'  
 }all\_consumers = get\_information()  
 if len(all\_consumers) > 0:  
 kafka\_producer = connect\_kafka\_producer()  
 for consumer in all\_information:  
 publish\_message(kafka\_producer, 'raw', consumer.strip())  
 if kafka\_producer is not None:  
 kafka\_producer.close()

/anaconda3/anaconda/bin/python /Development/DataScience/Kafka/kafka-consumer-alert/producer-raw-recipies.py  
Accessing list  
Message published successfully.  
Message published successfully.  
Message published successfully.Process finished with exit code 0

consumer = KafkaConsumer(  
'numtest',  
bootstrap\_servers=['localhost:9092'],  
auto\_offset\_reset='earliest',  
enable\_auto\_commit=True,  
group\_id='my-group',  
value\_deserializer=lambda x: loads(x.decode('utf-8')))

client = MongoClient('localhost:27017')  
collection = client.numtest.numtest

for message in consumer:  
message = message.value  
collection.insert\_one(message)  
print('{} added to {}'.format(message, collection))

Kafka is a scalable, fault-tolerant, publish-subscribe messaging system that enables the user to build distributed applications. Due to its high performance and efficiency, It is getting popular among companies that are producing loads of data from various external sources and want to provide real-time findings from it.

The end-

Wagle