1. Prerequisites: 3
2. Installation of Kafka – 90 Mins 8
3. Installation Confluent Kafka (Local) – 30 Minutes 16
4. Basic Kafka Operations - CLI (Topic) – 30 Mins 17
5. Basic Kafka Operations - CLI (Producer & Consumer) – 30 Mins 21
6. Zookeeper – 120 Minutes 23
7. Kafka cluster – 90 Minutes 42
8. Securing Kafka ACL – 60 Minutes 67
9. Kafka Connector (File) - 60 Minutes 90
10. Kafka Connector (JDBC) - 60 Minutes 99
12. Errors 120
I. {test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient) 120
13. LOG verification + segment Sizing 122
14. Annexure Code: 123
II. DumplogSegment 123
III. Resources 124

Hardware:

8 GB RAM , 30 GB HDD , Centos 7 or above OS. Access to internet.

Software Inventory:

- Zookeeper Version: apache-zookeeper-3.8.0-bin.tar
- Apache kafka : 2.13-3.2.1
- JDK 11.0.16
- Eclipse for Linux. (Any Latest version for JEE Development)
- Status : Color is Verified

Last Updated: Dec 25 - 2022.

1. Prerequisites:

Option I

Start the VM using VM player and Logon to the server using telnet or directly in the VM console. Enter the root credentials to logon.

You can copy files from the host to VM using winscp.exe.

Option II.

Using docker:
Instantiate a container, kafka0.
You can copy files from the host to container using docker copy command.

Execute the following command, it will perform the following:

- Create a network : spark-net
- download the image, centos:7
- start a container with the name, kafka0 and mount host folder in /opt

#docker network create --driver bridge spark-net

#docker run --name kafka0 --hostname kafka0 -p 9092:9092 -p 8081:8081 -p 2181:2181 –p 9999:9999 -i -t --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Software:/Software centos:8 /usr/sbin/init

Optional:
#docker run --name kafka0 --hostname kafka0 -p 9092:9092 -p 9081:8081 -p 9082:8082 -p 3181:2181 -p 9990:9999 -p 9021:9021 -i -t --privileged --network spark-net -v /Volumes/Samsung_T5/software/:/Software -v /Volumes/Samsung_T5/software/install/:/opt -v /Volumes/Samsung_T5/software/data/:/data centos:7 /usr/sbin/init

You can verify the container from a separate terminal:

```
(base) Henrys-MacBook-Air:~ henrypotsangbam$ docker ps
CONTAINER ID        IMAGE                                COMMAND             CREATED             STATUS
        PORTS                                        NAMES
0945477d457b        ehenry0227/learning:spark-kafka-raw  "/bin/bash"         34 seconds ago      Up 33 secon
ds      0.0.0.0:6062->6062/tcp, 0.0.0.0:8081->8081/tcp  kafka0
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

Perform the installation after this as a normal server.

Update the server.properties.

// Don’t change the listeners address if the client also run in the host machine.

advertised.listeners=PLAINTEXT://localhost:9092

After installation to start the zookeeper and kafka broker:

Open two separate terminals and execute the following

```
# /opt/kafka/bin/zookeeper-server-start.sh /opt/kafka/config/zookeeper.properties
# /opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

Kafka 2nd Cluster: (It will be used as DC2 for Kafka Mirroring)

```
#docker run --name kafka1 --hostname kafka1  -p 7092:9092 -p 7081:8081 -p 4181:2181 -p
8999:9999 -i -t --privileged --network spark-net -v
/Volumes/Samsung_T5/software/:/Software -v
/Volumes/Samsung_T5/software/install/:/opt -v
/Volumes/Samsung_T5/software/data/:/data centos:7 /usr/sbin/init
```

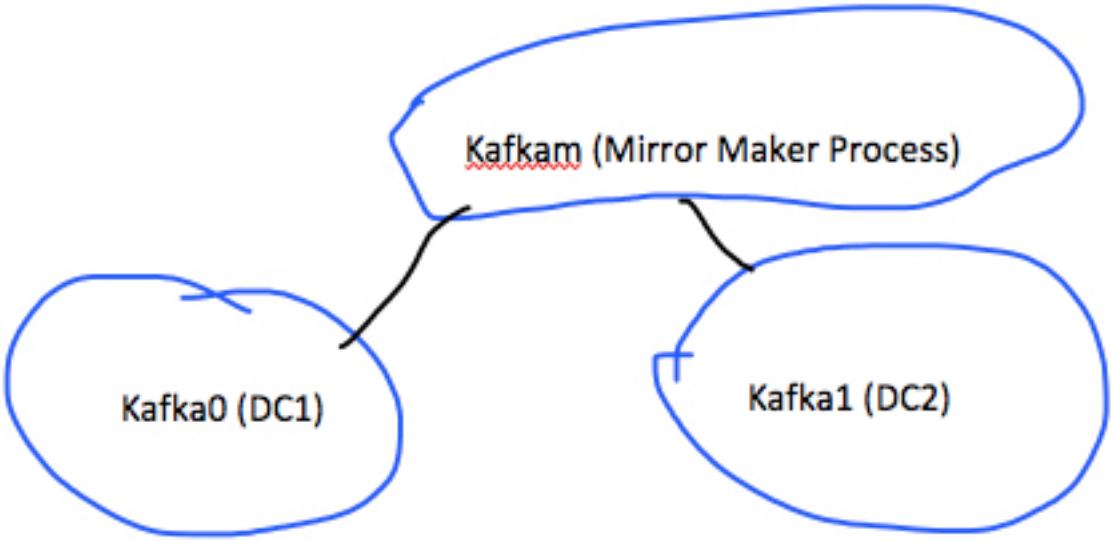
Install the kafka as done before if required or copy the kafka folder in different name to persevere the libraries.

Kafka Mirror Maker: (It will execute the Kafka Mirroring process)

```
#docker run --name kafkam --hostname kafkam  -p 6092:9092 -p 6081:8081  -i -t --
privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt
centos:7 /usr/sbin/init
```

Kafka Mirror Architecture:

12:46 PM



Note:
If you are using docker ensure to update the server.properties with the following entries for accessing the broker from the host machine.

// Changes Begin

listeners=PLAINTEXT://0.0.0.0:9092,PLAINTEXT_HOST://0.0.0.0:8081
advertised.listeners=PLAINTEXT://kafka0:9092,PLAINTEXT_HOST://localhost:8081
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT

// Changes End

use container, kafka0 to connect from Docker. However, use localhost:8081 for connecting from the Host machine.

kafka_2.13-3.2.1.tgz
jdk-11.0.16_linux-aarch64_bin.tar
apache-zookeeper-3.8.0-bin.tar.gz

2. Installation of Kafka – 90 Mins

You need to install java before installing zookeeper and Kafka.

Installation of Java
#tar -xvf jdk-11.0.16_linux* -C /opt
#cd /opt
#mv jdk* jdk

Set the above path in the PATH variable and JAVA_HOME

Update profile as follow:

#vi ~/.bashrc

export JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin

Update the shell scripts using the following command.
#bash

Installing Zookeeper

You can choose any of the options given below:

Option I (Fresh Installation): (We will use this for our lab)

The following steps install Zookeeper with a basic configuration in /opt/zookeeper.
Its configured to store data in /opt/data/zookeeper:

Extract the zookeeper archive file in /opt and rename the installation folder for brevity.

```
# tar -xvf apache-zookeeper* -C /opt
#cd /opt
#mv a*zookeeper* /opt/zookeeper
#mkdir -p /opt/data/zookeeper
```

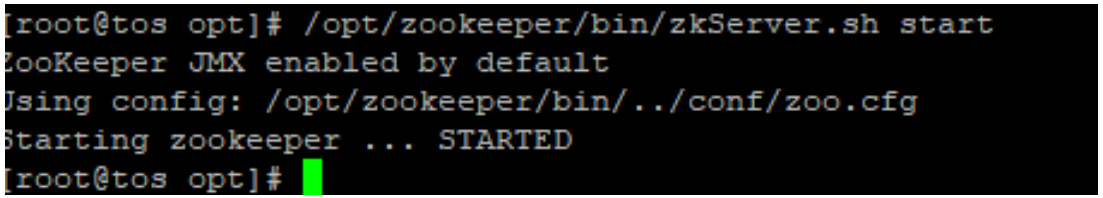
Create a zookeeper configuration file and update with the following values.

```
#vi /opt/zookeeper/conf/zoo.cfg
 tickTime=2000
 dataDir=/opt/data/zookeeper
 clientPort=2181
```

Update the zoo.cfg with the above entries. You can save the file using esc+wq!

Start the zookeeper using the following scripts.

/opt/zookeeper/bin/zkServer.sh start



Option II (Part of the Apache Kafka) – Skip.

```
#bin/zookeeper-server-start.sh config/zookeeper.properties
```

Option II Ends Here.

You can now validate that Zookeeper is running correctly in standalone mode by connecting to the client port and sending the four-letter command srvr:
#yum install telnet

#telnet localhost 2181

- srvr

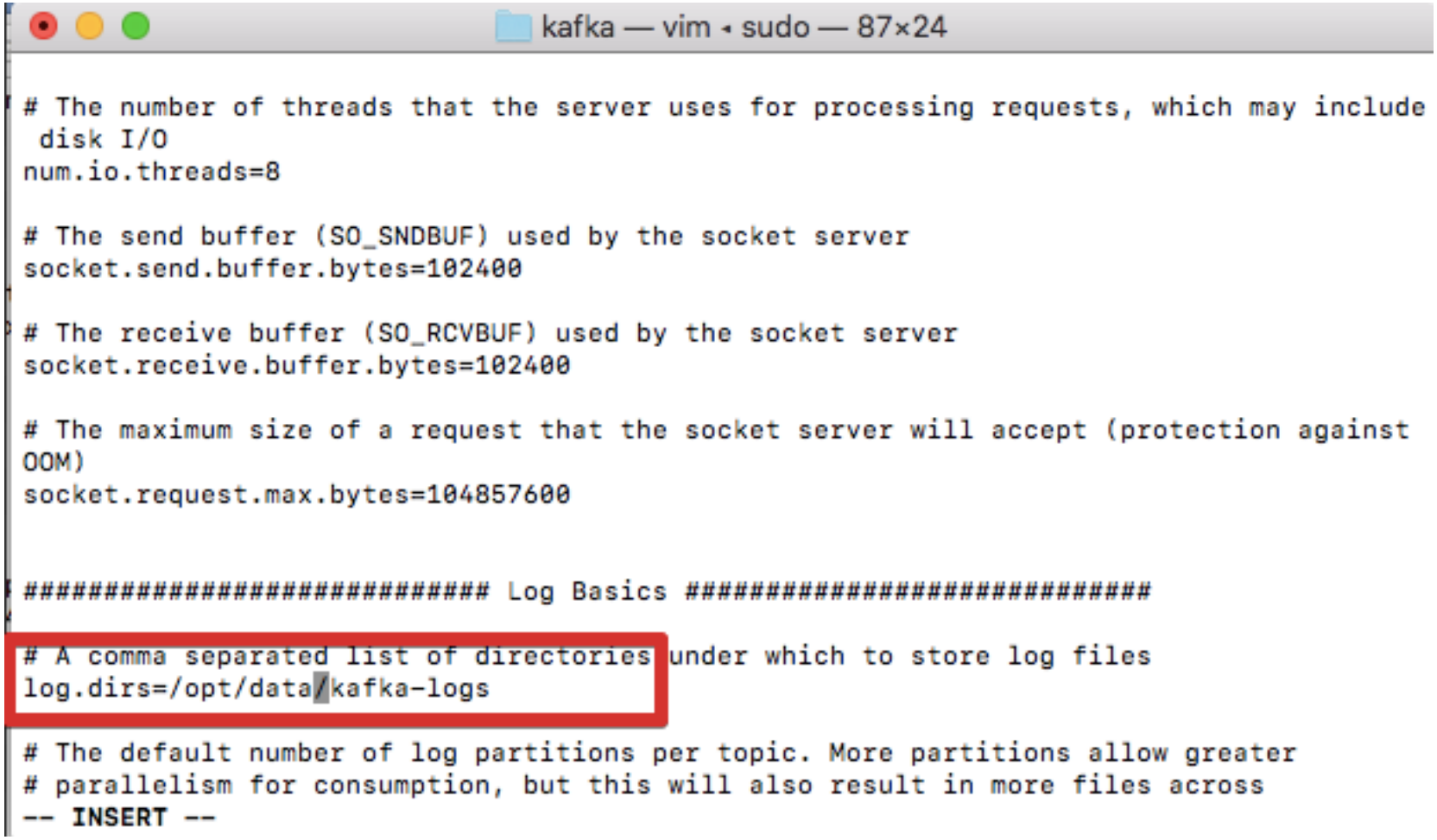
After zookeeper installation, let us install the Kafka Broker.

Installation of Kafka Broker

The following example installs Kafka in /opt/kafka, configured to use the Zookeeper server started previously and to store the message log segments stored in /tmp/kafka-logs:

tar -xvf kafka_2*–C /opt
#cd /opt
mv kafka_2* /opt/kafka
mkdir /opt/data/kafka-logs

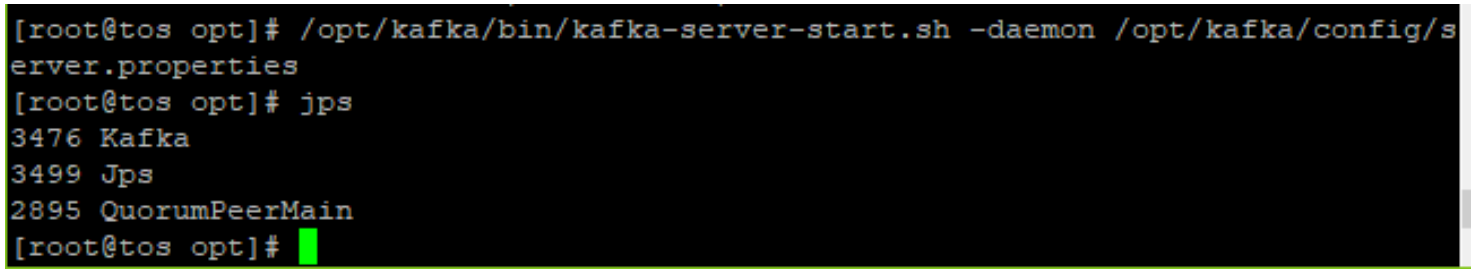
Update the /opt/kafka/config/server.properties to store the Kafka Log in the above mention folder.



If you are using docker, kindly refer the prerequisite section for setting specific to docker.

Start the broker with the following command

/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties



#mkdir /opt/scripts

All the common execution scripts will be stored in the above folder.
The following scripts will start a zookeeper along with a broker. Create the following files and update with the following scripts. It will start the zookeeper and kafka broker using the mention script.

#cd /opt/scripts
#vi startABroker.sh
############################## Scripts Begin ######################
#!/usr/bin/env bash

```
# Start Zookeeper.
/opt/zookeeper/bin/zkServer.sh start

#Start Kafka Server
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties


echo "Started Successfully"

//########################################## Scripts End
####################
To shutdown the Broker, find the process and kill.
ps -eaf  | grep java
or
create a scripts : /opt/scripts/stopBroker.sh with the following commands in it.
#!/usr/bin/env bash
/opt/kafka/bin/kafka-server-stop.sh
```

To Stop Zookeeper create the following script. Don't include the ---- line.

```
#vi /opt/scripts/stopZookeeper.sh
Update the following commands in the above script and save it.
-----------------------------------------------------------------
#!/usr/bin/env bash
# Stop Zookeeper.
```

/opt/zookeeper/bin/zkServer.sh stop
echo "Stop zookeeper Successfully"

To reinitialize the Cluster.
#vi reinitializeCluster.sh

```
#!/usr/bin/env bash

rm -fr /opt/kafka/data/kafka-logs/*
rm -fr /opt/kafka/data/zookeeper/*

cp /opt/kafka/config/server.properties_plain /opt/kafka/config/server.properties

echo "Reinitialize Successfully"
```

Lab Installation completes End here.

3. Installation Confluent Kafka (Local) – 30 Minutes

The purpose of this lab is to demonstrates the basic and most powerful capabilities of Confluent Platform – Schema Registry.

Install Confluent kafka with the following step.

Inflate the confluent kafka compress file as shown below:

#tar -xvf confluent-7.2.2.tar -C /opt

Rename the folder.

#cd /opt

mv confluent-7.2.2 confluent

Set the environment variable for the Confluent Platform directory.

```
export CONFLUENT_HOME=/opt/confluent
```

Set your PATH variable:

vi ~/.bashrc

```
export PATH=/opt/confluent/bin:${PATH};
```

type the following to activate the script.

#bash

---------- Lab Ends Here ----------

4. Basic Kafka Operations - CLI (Topic) – 30 Mins

In this lab you will be able to create a topic and perform some operations to understand the information about topic like partition and replication.
You need to start the broker using startABroker.sh. The script should be in /opt/scripts folder
#sh startABroker.sh
#jps

```
[root@tos scripts]# sh startABroker.sh
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
Started Successfully
[root@tos scripts]# jps
11665 Jps
11646 Kafka
11375 QuorumPeerMain
[root@tos scripts]#
```

Once the Kafka broker is started, we can verify that it is working by performing some simple operations against the broker; creating a test topic etc.

Create and verify details about topic:

/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic test

```
# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 12 --topic IBM

# /opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka0:9092  --describe --topic test

#/opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka0:9092 --describe --topic IBM
```

Verify the no of partition in the output.

list and describe topic.

```
#/opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka0:9092  --list
```

List and describe Topics

What does the tool do?
This tool lists the information for a given list of topics. If no topics are provided in the command line, the tool queries zookeeper to get all the topics and lists the information for them. The fields that the tool displays are - topic name, partition, leader, replicas, isr.

How to use the tool?

List only single topic named "test" (prints only topic name)

#/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092 --topic test

List all topics (prints only topic names)
#/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092

Describe only single topic named "test" (prints details about the topic)

#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092 --topic test

Describe all topics (prints details about the topics)
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092

We will understand the output in details later.

Create Topics
What does the tool do?
By default, Kafka auto creates topic if "auto.create.topics.enable" is set to true on the server. This creates a topic with a default number of partitions, replication factor and uses Kafka's default scheme to do replica assignment. Sometimes, it may be required that we would like

to customize a topic while creating it. This tool helps to create a topic and also specify the number of partitions, replication factor and replica assignment list for the topic.

How to use the tool?

create topic with default settings
#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --topic topic1 --partitions 2 --replication-factor 1

Create a topic with replication 2.

#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --topic topic2 --partitions 2 --replication-factor 2

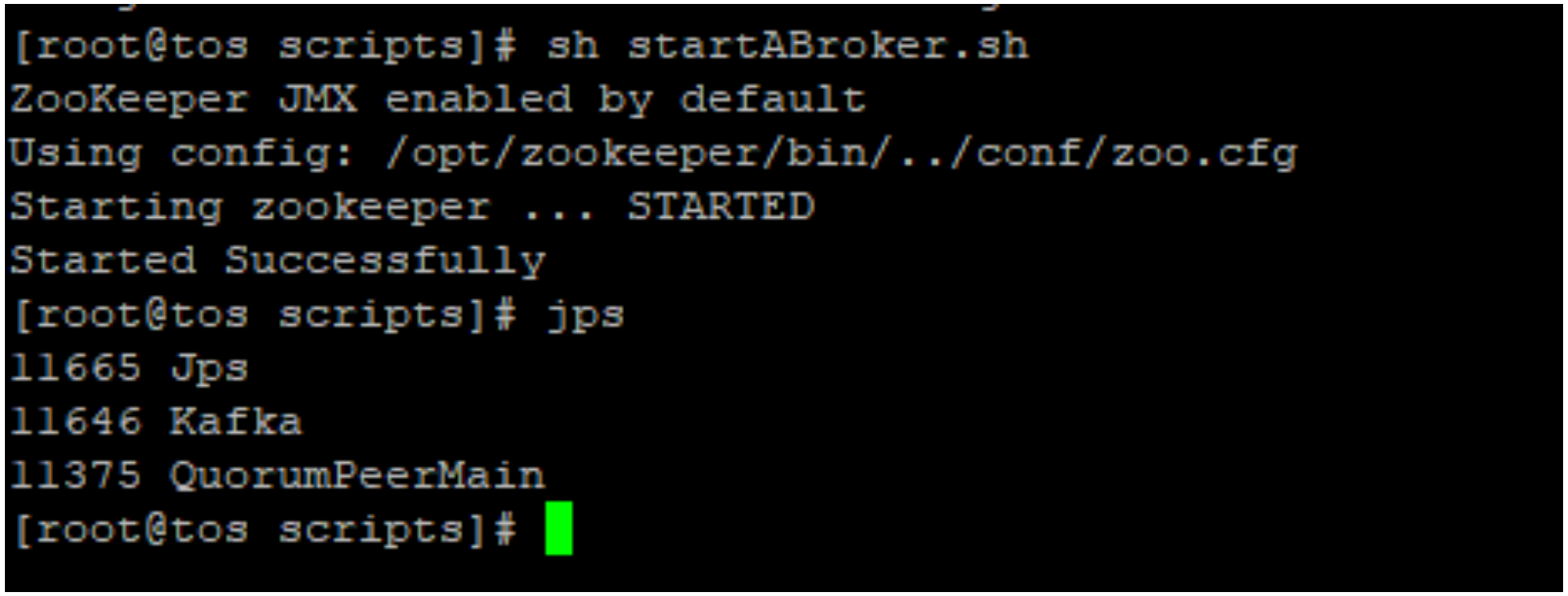
As shown above, it generates an error. Since there is only a single broker. It will fix later.

Lab CLI completes End here.

5. Basic Kafka Operations - CLI (Producer & Consumer) – 30 Mins

In this lab we will send message to the broker and consumer message using the kafka inbuilt commands.
You need to complete the previous lab before proceeding ahead.
You need to start the broker using startABroker.sh if not done earlier. The script should be in /opt/scripts folder

#sh startABroker.sh
#jps



Sent message to **test** topic: Open a console to send message to the topic, test. Enter some text as shown below.

```
# /opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic test
Test Message 1
Test Message 2
^D
#
```

Consume messages from a test topic: As soon as you enter the following script in a separate terminal, you should be able to consume the messages that we have type in the producer console.

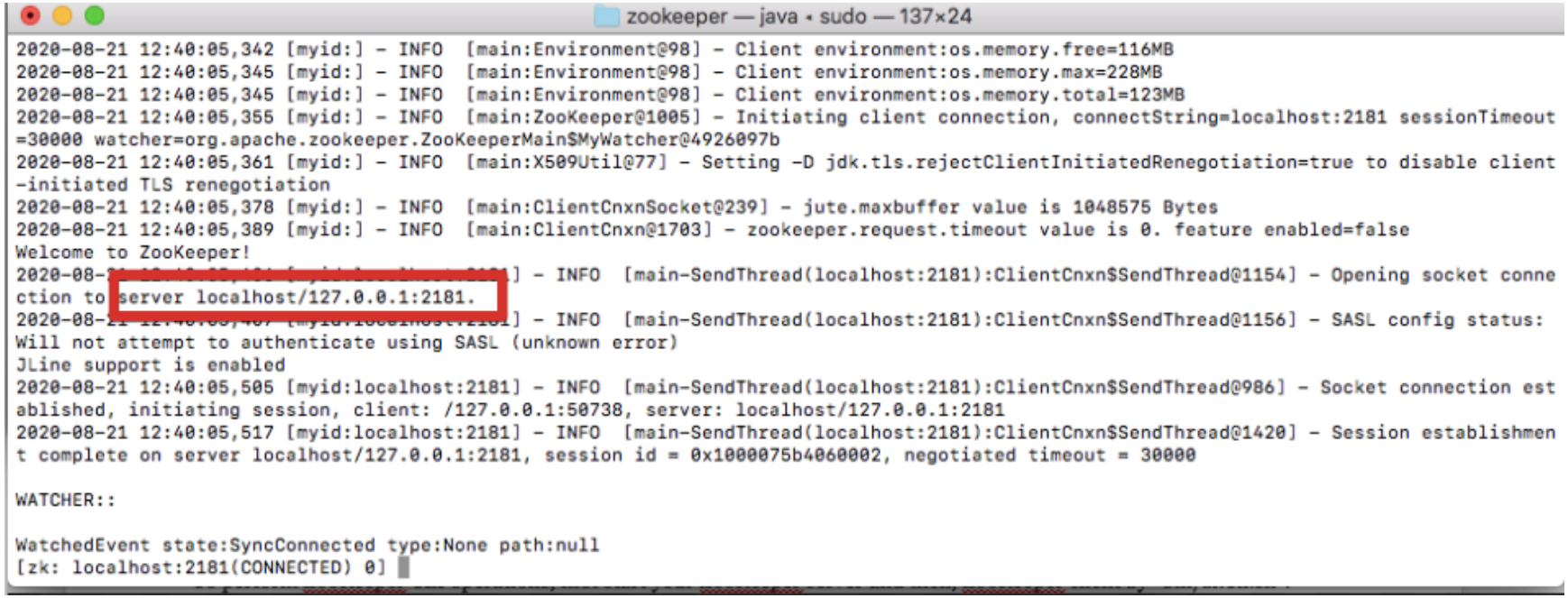
```
# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic test --from-beginning
```

Lab CLI ends here.

6. Zookeeper – 120 Minutes

In this lab, we will perform familiarization of ZK cli features and configure zookeeper ensemble:

To perform ZooKeeper CLI operations, first start your ZooKeeper server and then, ZooKeeper client by executing “bin/zkCli.sh”, which is in the Broker installation folder.
/opt/zookeeper/bin
#bin/zkCli.sh



or New CLI

```
[root@kafka0 kafka]# bin/zookeeper-shell.sh localhost:2181
Connecting to localhost:2181
Welcome to ZooKeeper!
JLine support is disabled

WATCHER::

WatchedEvent state:SyncConnected type:None path:null


ls /
[admin, brokers, cluster, config, consumers, controller, controller_epoch, feature, isr_change_notification, l
atest_producer_id_block, log_dir_event_notification, zookeeper]
ls -R /
/
/admin
/brokers
/cluster
```

As highlighted above, the cli is connected to the zookeeper running on localhost listening at port 2181.

Let us get all znode:

#ls /

```
[zk: localhost:2181(CONNECTED) 1] ls /
[admin, brokers, cluster, config, consumers, controller, controller_epoch, isr_change_notification, latest_producer_id_block, log_dir_event_notification, zookeeper]
[zk: localhost:2181(CONNECTED) 2]
```

#ls -R /
#ls -R /brokers
#get /brokers/ids/0

```
[zk: localhost:2181(CONNECTED) 22] get  /brokers/ids/0
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://192.168.0.111:9092"],"jmx_port":-1,"host":"192.168.0.111","timestamp":"1597992632471","port":9092,"version":4}
[zk: localhost:2181(CONNECTED) 23]
```

Information about the broker which is register in the znode.

Here, the Node 0 is own by the broker IP – 192.168.0.111 with the port 9092.
This way you can determine the Node details whenever there are any issues as shown below:

```
[2020-08-21 12:52:15,093] WARN [Consumer clientId=consumer-console-consumer-26076-1, groupId=console-consumer-26076] Connection to node -1 (localhost/127.0.0.1:9020) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2020-08-21 12:52:15,094] WARN [Consumer clientId=consumer-console-consumer-26076-1, groupId=console-consumer-26076] Bootstrap broker localhost:9020 (id: -1 rack: null) disconnected (org.apache.kafka.clients.NetworkClient)
^CProcessed a total of 0 messages
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$
```

Using the Kafka tool zookeeper-shell.sh (Broker Installation folder) we can connect to a ZooKeeper host in our cluster and look at how data is stored.

#zookeeper-shell.sh localhost:2181
#ls /brokers/topics

If we look at the path /brokers/topics you should see a list of the topics that you have created.

```
(base) Henrys-MacBook-Air:bin henrypotsangbam$ sh zookeeper-shell.sh localhost:2181
Connecting to localhost:2181
Welcome to ZooKeeper!
JLine support is disabled

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

ls
ls [-s] [-w] [-R] path
ls /brokers/topics
[__consumer_offsets, quickstart-events, test, topic1]
```

You should also be able to see the topic __consumer_offsets. That topic is one that you did not create, it is in fact a private topic used internal by Kafka itself. This topic stored the committed offsets for each topic and partition per group id.

The path /controller exists in zookeeper and we are running one command to look at that current value

zookeeper-shell.sh localhost:2181
get /controller

Now in the next section, let us configure cluster of zookeeper – 3 nodes.
Ensemble

Setting up a ZooKeeper Ensemble

Create 3 zookeepers' configuration to create zookeeper ensemble of 3 nodes:

The `<ZOOKEEPER_HOME>/conf/zoo1.cfg` file should have the content:

```
dataDir=/opt/data/zookeeper/1
clientPort=2181
initLimit=5
syncLimit=2
server.1=localhost:2888:3888
server.2=localhost:2889:3889
server.3=localhost:2890:3890
```

The <ZOOKEEPER_HOME>/conf/zoo2.cfg file should have the content:

```
dataDir=/opt/data/zookeeper/2
clientPort=2182
initLimit=5
syncLimit=2
server.1=localhost:2888:3888
server.2=localhost:2889:3889
server.3=localhost:2890:3890
```

The <ZOOKEEPER_HOME>/conf/zoo3.cfg file should have the content:

```
dataDir=/opt/data/zookeeper/3
clientPort=2183
initLimit=5
syncLimit=2
server.1=localhost:2888:3888
server.2=localhost:2889:3889
server.3=localhost:2890:3890
```

To complete your multi-node configuration, you will specify a node ID on each of the servers. To do this, you will create a myid file on each node. Each file will contain a number that correlates to the server number assigned in the configuration file.

#cd /opt/data/zookeeper/
#mkdir 1 2 3
#vi 1/myid → 1
2/myid → 2
3/myid → 3

```
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo vi 1/myid
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo vi 2/myid
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo vi 3/myid
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ more 1/myid
1
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ pwd
/opt/data/zookeeper
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$
```

To start the servers, you can simply explicitly reference the configuration files:

```
cd <ZOOKEEPER_HOME>
bin/zkServer.sh start conf/zoo1.cfg
bin/zkServer.sh start conf/zoo2.cfg
bin/zkServer.sh start conf/zoo3.cfg
```

```
4078 QuorumPeerMain
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo bin/zkServer.sh start conf/zoo2.cfg
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: conf/zoo2.cfg
Starting zookeeper ... STARTED
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo bin/zkServer.sh start conf/zoo3.cfg
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: conf/zoo3.cfg
Starting zookeeper ... STARTED
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ jps
4136 Jps
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo jps
4131 QuorumPeerMain
4139 Jps
4107 QuorumPeerMain
4078 QuorumPeerMain
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$
```

You will now start a ZooKeeper command line client and connect to ZooKeeper on **node 1**:

```
#bin/zkCli.sh -server localhost:2181
```

List the newly created znode:

```
#ls /
```

You can verify the leader using command:

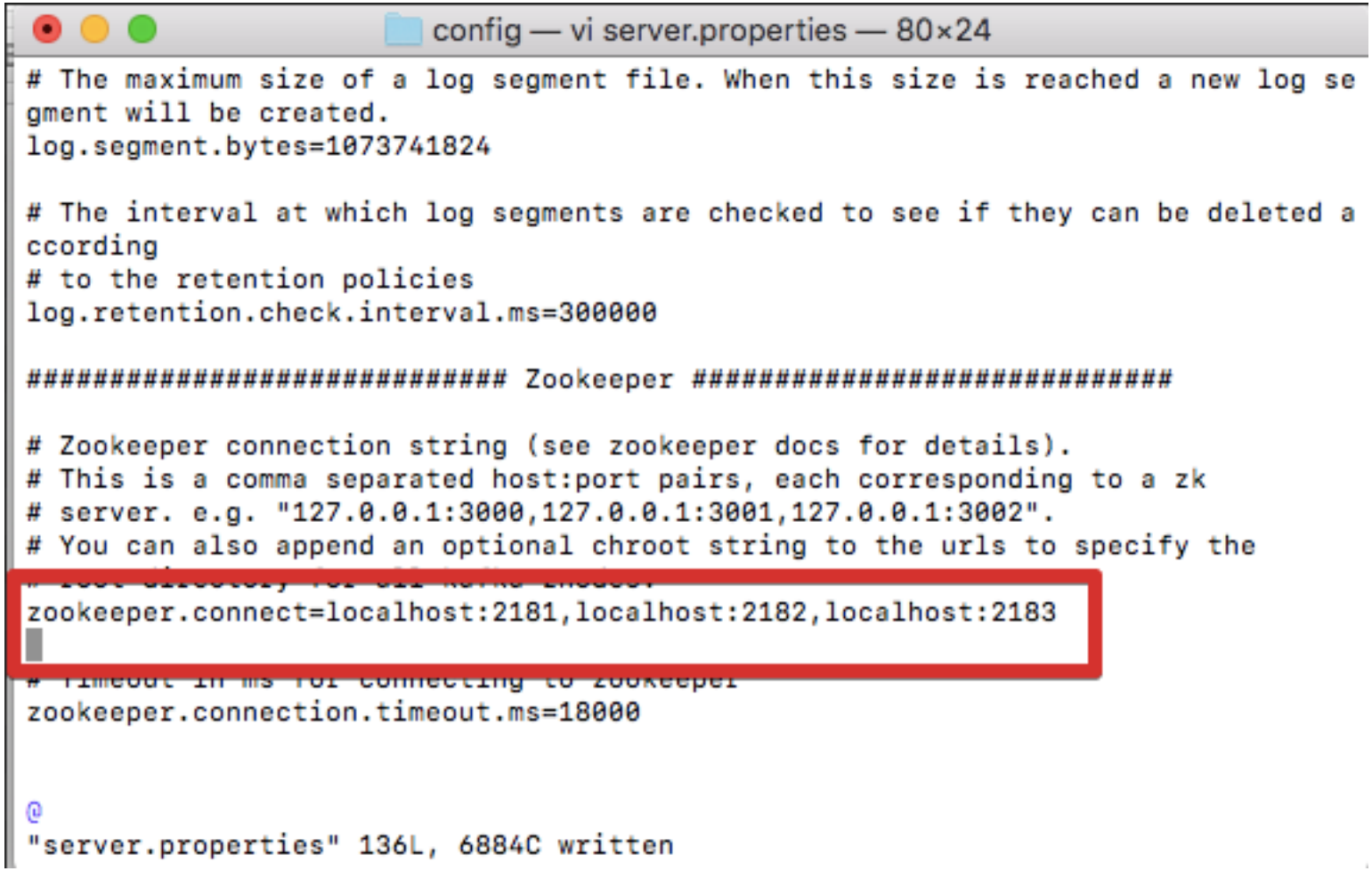
```
#cd /opt/zookeeper
#bin/zkServer.sh status conf/zoo1.cfg
# bin/zkServer.sh status conf/zoo2.cfg
# bin/zkServer.sh status conf/zoo3.cfg
```

```
[root@kafka0 zookeeper]# bin/zkServer.sh status conf/zoo2.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo2.cfg
Client port found: 2182. Client address: localhost.
Mode: leader
[root@kafka0 zookeeper]# bin/zkServer.sh status conf/zoo1.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo1.cfg
Client port found: 2181. Client address: localhost.
Mode: follower
[root@kafka0 zookeeper]# bin/zkServer.sh status conf/zoo3.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo3.cfg
Client port found: 2183. Client address: localhost.
Mode: follower
[root@kafka0 zookeeper]# pwd
/opt/zookeeper
```

Here, the leader is running in 2182 port.

Finally, let us configure the broker to point to the zookeeper ensemble.

The configuration will be in the Kafka conf folder.
Copy conf/server.properties into conf/server1.properties and modify the following two properties in it. Use cp command.



log.dirs=/opt/data/kafka-logs1

Updated the Zookeeper.connect property to connect to all zookeeper nodes.

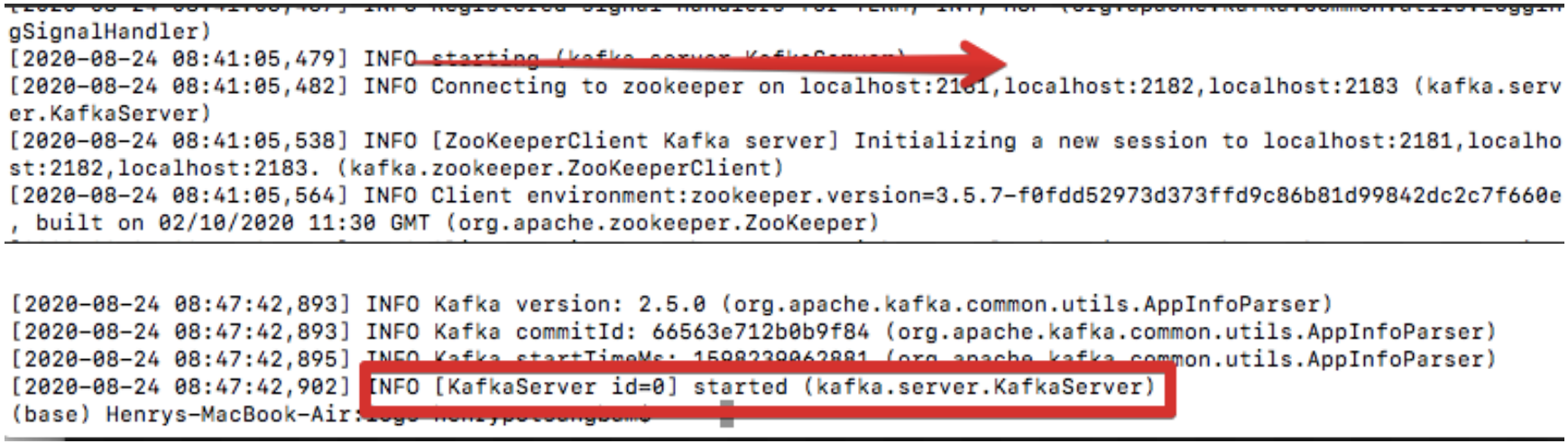
Start all the zookeepers if not done.
#cd /opt/zookeeper

```
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo bin/zkServer.sh start
conf/zoo1.cfg
Password:
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: conf/zoo1.cfg
Starting zookeeper ... STARTED
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo bin/zkServer.sh start
conf/zoo2.cfg
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: conf/zoo2.cfg
Starting zookeeper ... STARTED
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$ sudo bin/zkServer.sh start
conf/zoo3.cfg
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: conf/zoo3.cfg
Starting zookeeper ... STARTED
(base) Henrys-MacBook-Air:zookeeper henrypotsangbam$
```

Start the broker using the modified configuration.

#/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server1.properties

You can verify from the server.log that the broker is referring to the new property as shown below: /opt/kafka/logs/server.log



Let us perform some activities using the new set up.

/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 2 --topic testz
/opt/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic testz

/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testz
Test Message 1
Test Message 2
To exit : ^D

Consume the above messages:

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

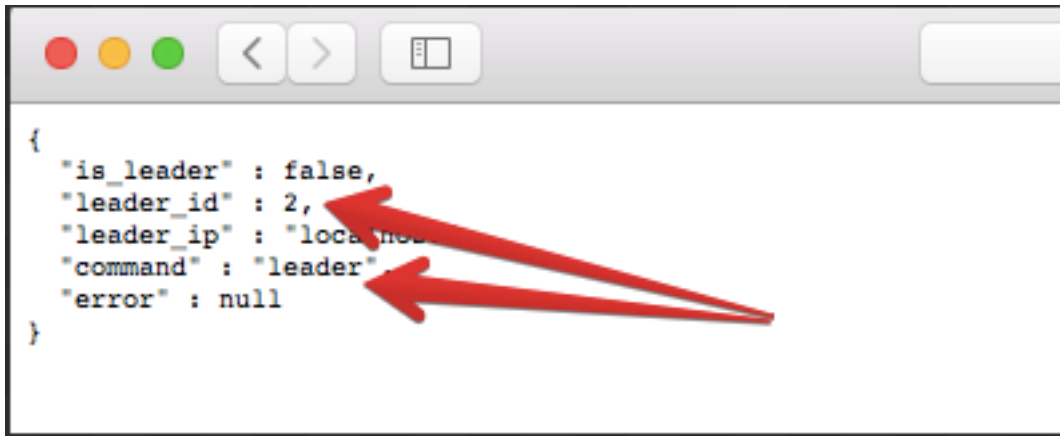
```
(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-producer.sh --broker-list localh
ost:9092 --topic testz
>Test Message 1
Test Message 2
>>(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server
ocalhost:9092 --topic testz --from-beginning
Test Message 1
Test Message 2
```

Let us verify the Leader of the zookeeper and Kill the leader process.

http://localhost:8080/commands/leader

or CLI :

#/opt/zookeeper/bin/zkServer.sh status /opt/zookeeper/conf/zoo2.cfg




```
[root@kafka0 /]# /opt/zookeeper/bin/zkServer.sh status /opt/zookeeper/conf/zoo2.cfg
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/conf/zoo2.cfg
Client port found: 2182. Client address: localhost.
Mode: leader
[root@kafka0 /]#
```

You can verify the other configuration if 2 is not the leader.

Since 2 is the leader. Let us kill the process Id of the Zookeper id 2.

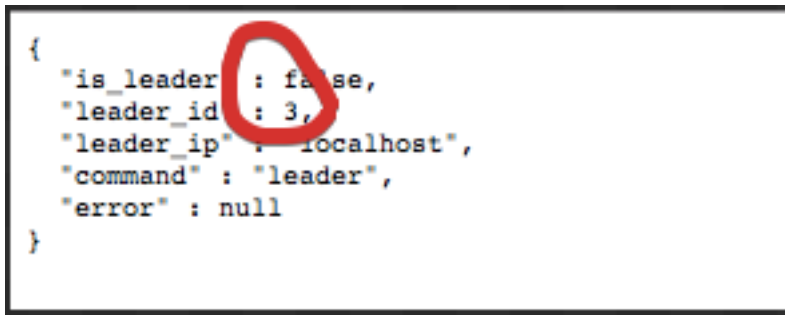
```
#ps -eaf | grep zoo2.cfg
```

```
(base) Henrys-MacBook-Air:config henrypotsangbam$ sudo ps -eaf | grep zoo2.cfg
  501  3725   537   0  9:05AM ttys000    0:00.00 grep zoo2.cfg
    0   684     1   0  8:39AM ttys001    0:05.91 /usr/bin/java -Dzookeeper.log.dir=/opt/zookeeper/bin/../logs -Dzookeeper.log.fil
e=zookeeper-root-server-Henrys-MacBook-Air.local.log -Dzookeeper.root.logger=INFO,CONSOLE -XX:+HeapDumpOnOutOfMemoryError -XX:OnO
utOfMemoryError=kill -9 %p -cp /opt/zookeeper/bin/../zookeeper-server/target/classes:/opt/zookeeper/bin/../build/classes:/opt/zoo
keeper/bin/../zookeeper-server/target/lib/*.jar:/opt/zookeeper/bin/../build/lib/*.jar:/opt/zookeeper/bin/../lib/zookeeper-prometh
eus-metrics-3.6.1.jar:/opt/zookeeper/bin/../lib/zookeeper-jute-3.6.1.jar:/opt/zookeeper/bin/../lib/zookeeper-3.6.1.jar:/opt/zooke
eper/bin/../lib/snappy-java-1.1.7.jar:/opt/zookeeper/bin/../lib/slf4j-log4j12-1.7.25.jar:/opt/zookeeper/bin/../lib/slf4j-api-1.7.
25.jar:/opt/zookeeper/bin/../lib/simpleclient_servlet-0.6.0.jar:/opt/zookeeper/bin/../lib/simpleclient_hotspot-0.6.0.jar:/opt/zoo
keeper/bin/../lib/simpleclient_common-0.6.0.jar:/opt/zookeeper/bin/../lib/simpleclient-0.6.0.jar:/opt/zookeeper/bin/../lib/netty-
transport-native-unix-common-4.1.48.Final.jar:/opt/zookeeper/bin/../lib/netty-transport-native-epoll-4.1.48.Final.jar:/opt/zookee
per/bin/../lib/netty-transport-4.1.48.Final.jar:/opt/zookeeper/bin/../lib/netty-resolver-4.1.48.Final.jar:/opt/zookeeper/bin/../l
ib/netty-handler-4.1.48.Final.jar:/opt/zookeeper/bin/../lib/netty-common-4.1.48.Final.jar:/opt/zookeeper/bin/../lib/netty-codec-4
.1.48.Final.jar:/opt/zookeeper/bin/../lib/netty-buffer-4.1.48.Final.jar:/opt/zookeeper/bin/../lib/metrics-core-3.2.5.jar:/opt/zoo
keeper/bin/../lib/log4j-1.2.17.jar:/opt/zookeeper/bin/../lib/json-simple-1.1.1.jar:/opt/zookeeper/bin/../lib/jline-2.11.jar:/opt/
zookeeper/bin/../lib/jetty-util-9.4.24.v20191120.jar:/opt/zookeeper/bin/../lib/jetty-servlet-9.4.24.v20191120.jar:/opt/zookeeper/
bin/../lib/jetty-server-9.4.24.v20191120.jar:/opt/zookeeper/bin/../lib/jetty-security-9.4.24.v20191120.jar:/opt/zookeeper/bin/../
lib/jetty-io-9.4.24.v20191120.jar:/opt/zookeeper/bin/../lib/jetty-http-9.4.24.v20191120.jar:/opt/zookeeper/bin/../lib/javax.servl
et-api-3.1.0.jar:/opt/zookeeper/bin/../lib/jackson-databind-2.10.3.jar:/opt/zookeeper/bin/../lib/jackson-core-2.10.3.jar:/opt/zoo
keeper/bin/../lib/jackson-annotations-2.10.3.jar:/opt/zookeeper/bin/../lib/commons-lang-2.6.jar:/opt/zookeeper/bin/../lib/commons
-cli-1.2.jar:/opt/zookeeper/bin/../lib/audience-annotations-0.5.0.jar:/opt/zookeeper/bin/../zookeeper-*.jar:/opt/zookeeper/bin/..
/zookeeper-server/src/main/resources/lib/*.jar:/opt/zookeeper/bin/../conf: -Xmx1000m -Dcom.sun.management.jmxremote -Dcom.sun.man
agement.jmxremote.local.only=false org.apache.zookeeper.server.quorum.QuorumPeerMain conf/zoo2.cfg
(base) Henrys-MacBook-Air:config henrypotsangbam$
```

In this case, the process id is 684.
kill -9 684

```
(base) Henrys-MacBook-Air:config henrypotsangbam$ sudo kill -9 684
(base) Henrys-MacBook-Air:config henrypotsangbam$ sudo jps
3745 Jps
659 QuorumPeerMain
3382 ConsoleConsumer
710 QuorumPeerMain
2026 Kafka
(base) Henrys-MacBook-Air:config henrypotsangbam$
```

You can refresh the browser to determine the new leader.



or using cli.

```
[root@kafka0 /]# /opt/zookeeper/bin/zkServer.sh status /opt/zookeeper/conf/zoo2.cfg
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/conf/zoo2.cfg
Client port found: 2182. Client address: localhost.
Error contacting service. It is probably not running.
[root@kafka0 /]# /opt/zookeeper/bin/zkServer.sh status /opt/zookeeper/conf/zoo3.cfg
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/conf/zoo3.cfg
Client port found: 2183. Client address: localhost.
Mode: leader
```

So which node is the leader in your case? Here its 3

Try sending some messages again:

```
(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testz
Password:
>Another Message
>
^CProcessed a total of 0 messages
(base) Henrys-MacBook-Air:config henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic testz --from-beginning
Test Message 1
Test Message 2
Another Message
```

You should be able to consume messages without any issue.

Get some config details from the zookeeper:

```
get /controller
get /brokers/topics/testz
get /brokers/ids/0
get /zookeeper/config
```

```
[zk: localhost:2181(CONNECTED) 19] get /controller
{"version":1,"brokerid":0,"timestamp":"1598239062599"}
[zk: localhost:2181(CONNECTED) 20] get /brokers/topics/testz
{"version":2,"partitions":{"1":[0],"0":[0]},"adding_replicas":{},"removing_replicas":{}}
[zk: localhost:2181(CONNECTED) 21] get /brokers/ids/0
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://quickstart.cloudera:9092"],"jmx_port":-1,"host":"quickstart.cloudera","timestamp":"1598239062331","port":9092,"version":4}
[zk: localhost:2181(CONNECTED) 22] get /zookeeper/config
server.1=localhost:2888:3888:participant
server.2=localhost:2889:3889:participant
server.3=localhost:2890:3890:participant
version=0
[zk: localhost:2181(CONNECTED) 23]
```

---------- lab Ends Here ----------
Log:

---

server.1=localhost:2888:3888 (2181)
server.2=localhost:2889:3889. (2182) - First Leader
server.3=localhost:2890:3890 -(2183) Second Leader

Scenarios : 2 was leader , after 2 shutdown -> 3 Became a leader

---

7. Kafka cluster – 90 Minutes

Understanding Kafka Failover in a cluster environment.
In this tutorial, we are going to run many Kafka Nodes on our development laptop so, you will need at least 8 GB of RAM for local dev machine. You can run just two servers if you have less memory than 8 GB.

We are going to create a replicated topic and then demonstrate consumer along with broker failover. We also demonstrate load balancing of Kafka consumers.
We show how, with many groups, Kafka acts like a Publish/Subscribe. But, when we put all of our consumers in the same group, Kafka will load share the messages to the consumers in the same group (more like a queue than a topic in a traditional MOM sense).

If not already running, start ZooKeeper.
Also, shut down Kafka from the first tutorial.

Next, you need to copy server properties for three brokers (detailed instructions to follow). Then we will modify these Kafka server properties to add unique Kafka ports, Kafka log locations, and unique Broker ids. Then we will create three scripts to start these servers up using these properties, and then start the servers.
Lastly, we create replicated topic and use it to demonstrate Kafka consumer failover, and Kafka broker failover.

Create three new Kafka server-n.properties files

In this section, we will copy the existing Kafka server.properties to server-0.properties, server-1.properties, and server-2.properties.
Then change server-0.properties to set log.dirs to "/opt/data/kafka-logs/kafka-0.
Then we modify server-1.properties to set port to 9093, broker id to 1,
and log.dirs to "/opt/data/kafka-logs/kafka-1".
Lastly modify server-2.properties to use port 9094, broker id 2,
and log.dirs "/opt/data/kafka-logs/kafka-2".

Copy server properties file as follows: We will store all server's configuration in a single folder config.

```
$ cd /opt
$ mkdir -p kafka-config/config
$ cp kafka/config/server.properties kafka-config/config/server-0.properties
$ cp kafka/config/server.properties kafka-config/config/server-1.properties
$ cp kafka/config/server.properties kafka-config/config/server-2.properties
```

```
[root@tos opt]# mkdir -p kafka-config/config
[root@tos opt]# pwd
/opt
[root@tos opt]# ls
couchbase  data  jdk1.8.0_45  kafka  kafka-config  zookeeper  zookeeper.out
[root@tos opt]# cp kafka/config/server.properties kafka-config/server-0.properti
es
[root@tos opt]# ls
couchbase  data  jdk1.8.0_45  kafka  kafka-config  zookeeper  zookeeper.out
[root@tos opt]# cp kafka/config/server.properties kafka-config/config/server-1.p
roperties
[root@tos opt]# cp kafka/config/server.properties kafka-config/config/server-2.p
roperties
[root@tos opt]#
```

With your favourite text editor update server-0.properties so that log.dirs is set to ./logs/kafka-0. Leave the rest of the file the same. Make sure log.dirs is only defined once.

#vi /opt/kafka-config/config/server-0.properties
broker.id=0
listeners=PLAINTEXT://kafka0:9092
advertised.listeners=PLAINTEXT://kafka0:9092
log.dirs=/opt/data/kafka-logs/kafka-0

With your favorite text editor change log.dirs, broker.id and and log.dirs of server-1.properties as follows.

#vi /opt/kafka-config/config/server-1.properties

```
broker.id=1
listeners=PLAINTEXT://kafka0:9093
advertised.listeners=PLAINTEXT://kafka0:9093
log.dirs=/opt/data/kafka-logs/kafka-1
```

With your favorite text editor change log.dirs, broker.id and and log.dirs of server-2.properties as follows.

#vi /opt/kafka-config/config/server-2.properties

```
broker.id=2
listeners=PLAINTEXT://kafka0:9094
advertised.listeners=PLAINTEXT://kafka0:9094
log.dirs=/opt/data/kafka-logs/kafka-2
```

Create Startup scripts for these three Kafka servers

The startup scripts will just run kafka-server-start.sh with the corresponding properties file.

#vi /opt/kafka-config/start-1st-server.sh

```
#!/usr/bin/env bash
## Run Kafka
/opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-0.properties"
```

#vi /opt/kafka-config/start-2nd-server.sh

```
#!/usr/bin/env bash
## Run Kafka
/opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-1.properties"
```

#vi /opt/kafka-config/start-3rd-server.sh

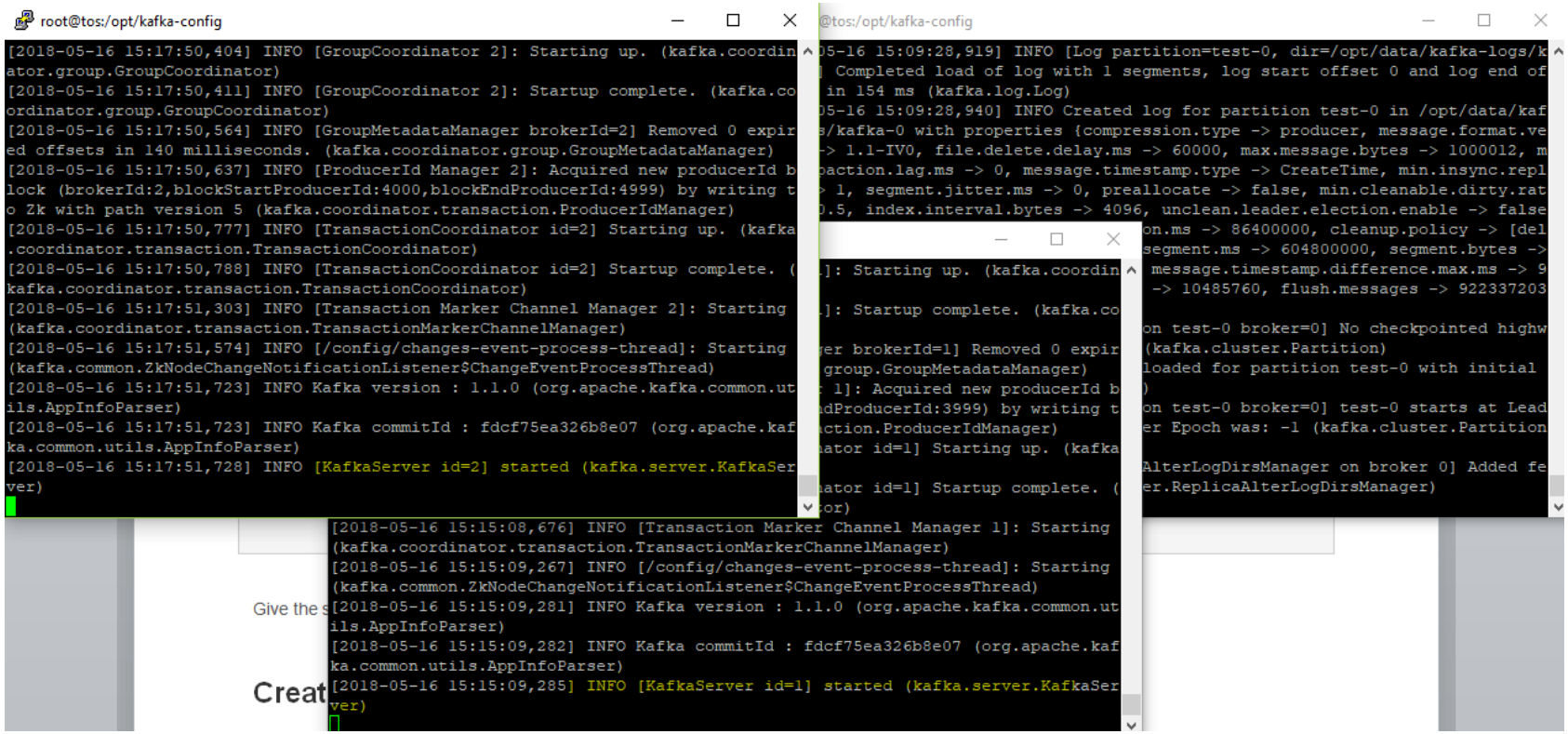
```
#!/usr/bin/env bash
## Run Kafka
/opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-2.properties"
```

Notice we are passing the Kafka server properties files that we created in the last step.
Now run all three in separate terminals/shells.

Run Kafka servers each in own terminal from /opt/kafka-config

```
#cd /opt/kafka-config
// to make the scripts executable.
#chmod 755 start-1st-server.sh  start-2nd-server.sh start-3rd-server.sh
$ ./start-1st-server.sh

$ ./start-2nd-server.sh

$ ./start-3rd-server.sh
```

Give these servers a couple of minutes to startup and connect to ZooKeeper.

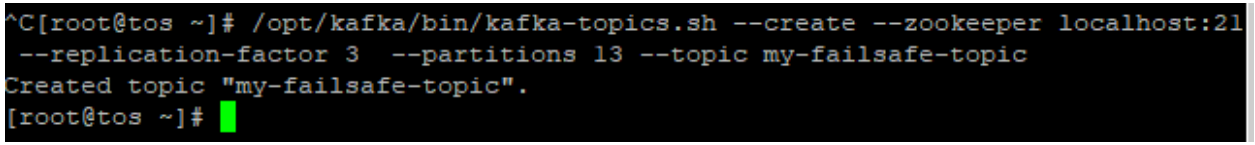


Create a replicated topic my-failsafe-topic

Now we will create a replicated topic that the console producers and console consumers can use.

Open a separate terminal and execute the following;

```
#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-fact
or 3  --partitions 13 --topic my-failsafe-topic
```



Notice that the replication factor gets set to 3, and the topic name is my-failsafe-topic, and like before it has 13 partitions.

Start Kafka Consumer that uses Replicated Topic

Start the consumer with the script in a separate terminal;

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9094,kafka0:9092
--topic my-failsafe-topic  --from-beginning
```

Notice that a list of Kafka servers is passed to --bootstrap-server parameter. Only, two of the three servers get passed that we ran earlier. Even though only one broker is needed, the

consumer client will learn about the other broker from just one server. Usually, you list multiple brokers in case there is an outage so that the client can connect.

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localh
ost:9094,localhost:9092 --topic my-failsafe-topic
```

Start Kafka Producer that uses Replicated Topic

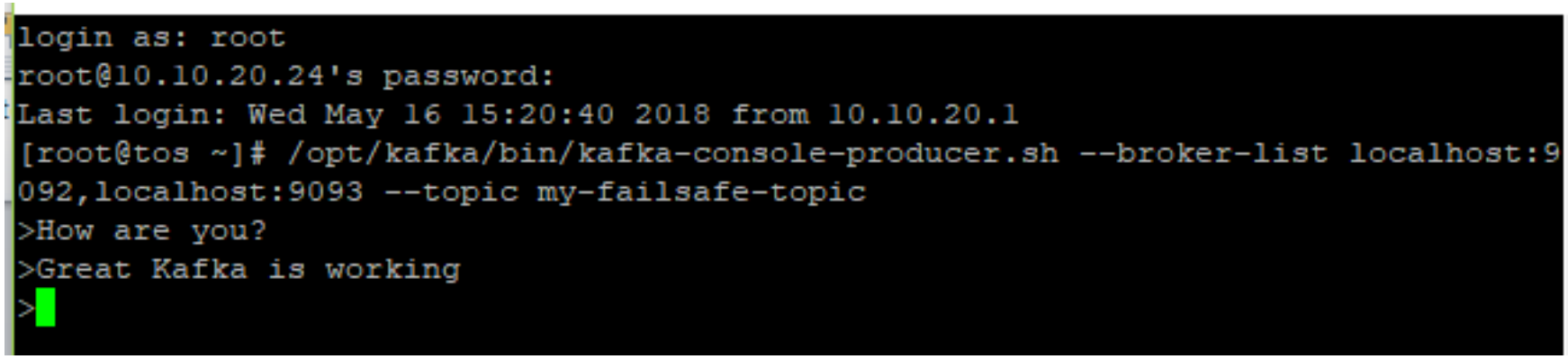
Next, we create a script that starts the producer. Then launch the producer with the script you create.

```
#/opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092,kafka0:9093 --topic
my-failsafe-topic
```

Notice we start Kafka producer and pass it a list of Kafka Brokers to use via the parameter --broker-list.
Now use the start producer script to launch the producer as follows.

Now send messages
Now send some message from the producer to Kafka and see those messages consumed by the consumer.

Producer Console.



Consumer Console

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localh
ost:9094,localhost:9092 --topic my-failsafe-topic
How are you?
Great Kafka is working
```

Now Start two more consumers and send more messages.

Now Start two more consumers in their own terminal window and send more messages from the producer. (Replace the hostname of your server aka localhost)

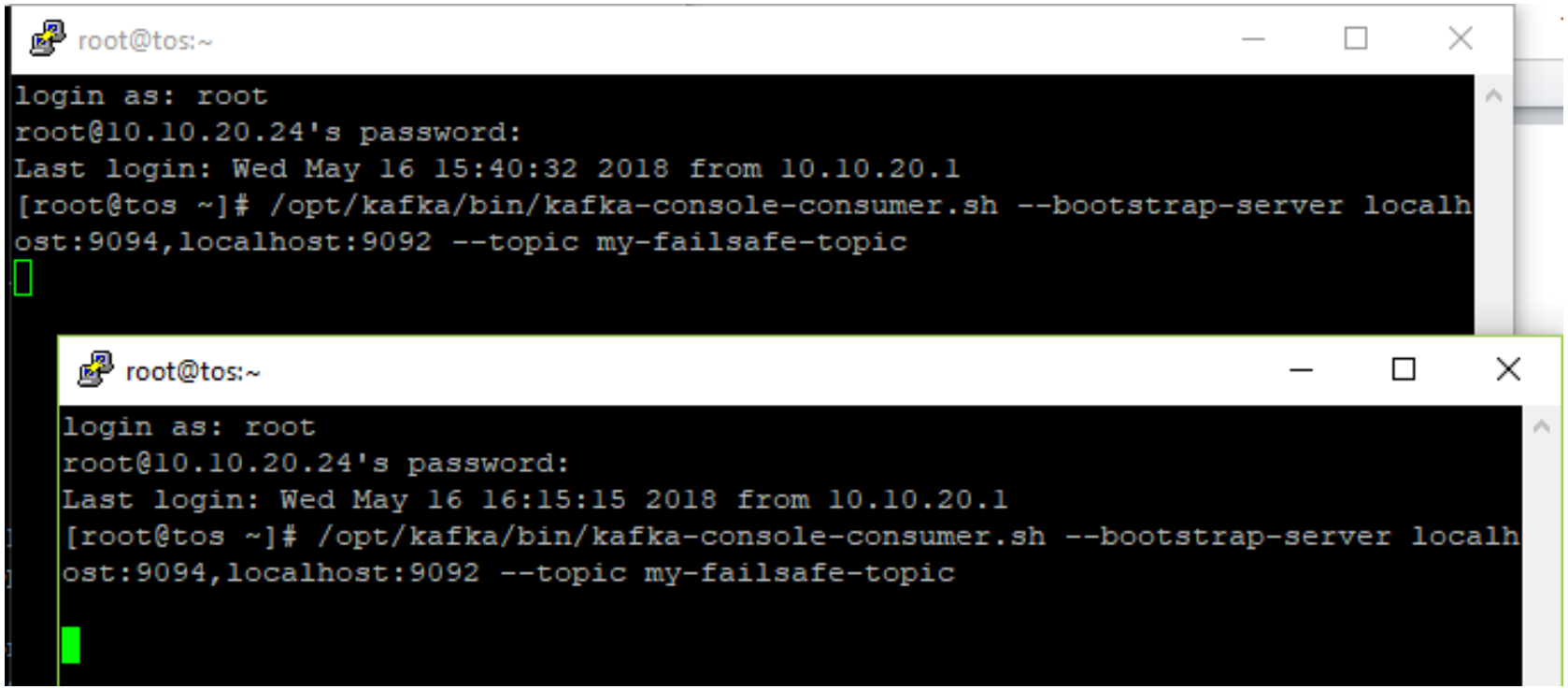
Consumer 1.

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9094,kafka0:9092
--topic my-failsafe-topic     --from-beginning
```

Consumer 2.

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9094,kafka0:9092
--topic my-failsafe-topic     --from-beginning
```

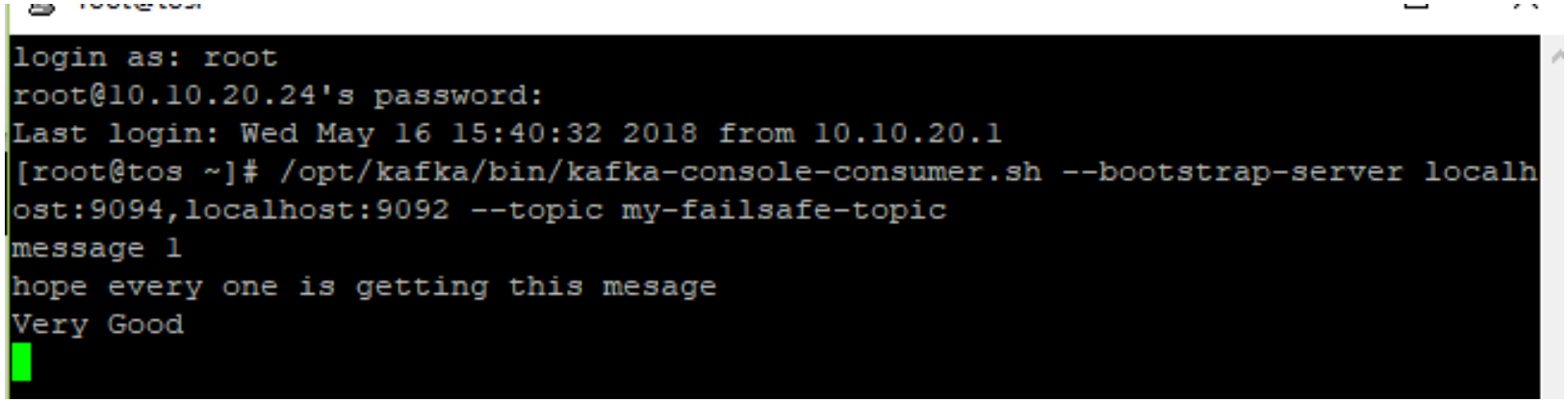
The two consumer consoles will be as shown below



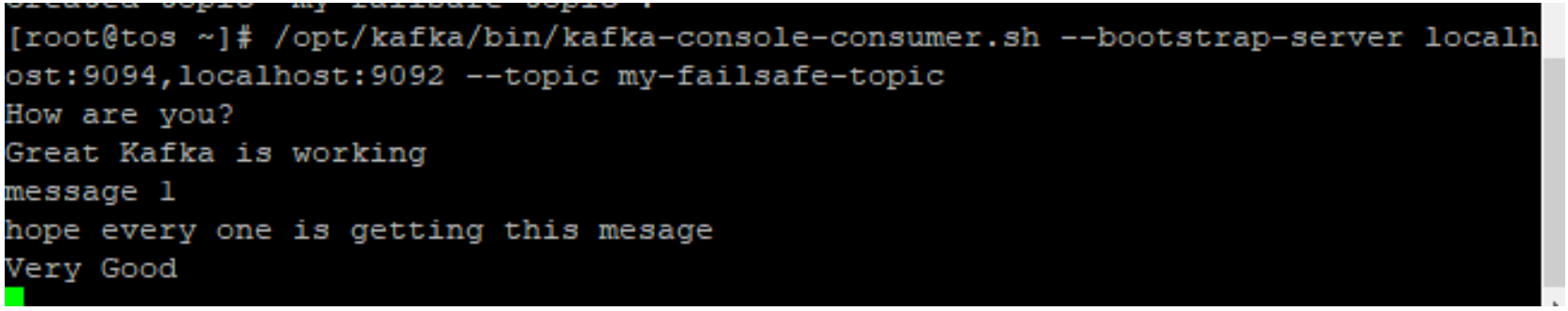
Producer Console will be as shown below:

```
login as: root
root@10.10.20.24's password:
Last login: Wed May 16 15:20:40 2018 from 10.10.20.1
[root@tos ~]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhos
092,localhost:9093 --topic my-failsafe-topic
>How are you?
>Great Kafka is working
>message 1
>hope every one is getting this mesage
>Very Good
>
```

Consumer Console 1st, you should be able to view the messages whatever you type on the producer console after the new consumer console was started.



Consumer Console 2nd in new Terminal, similarly all the messages that were type on the producer console should be also display in the second console after it was started.
Consumer Console 2nd in new Terminal



Notice that the messages are sent to all of the consumers because each consumer is in a different consumer group.

Change consumer to be in their own consumer group.

Stop the producers and the consumers before, but leave Kafka and ZooKeeper running. You can use ctrl + c.
We want to put all of the consumers in same *consumer group*. This way the consumers will share the messages as each consumer in the *consumer group* will get its share of partitions.

Run the following scripts three times – from different console.

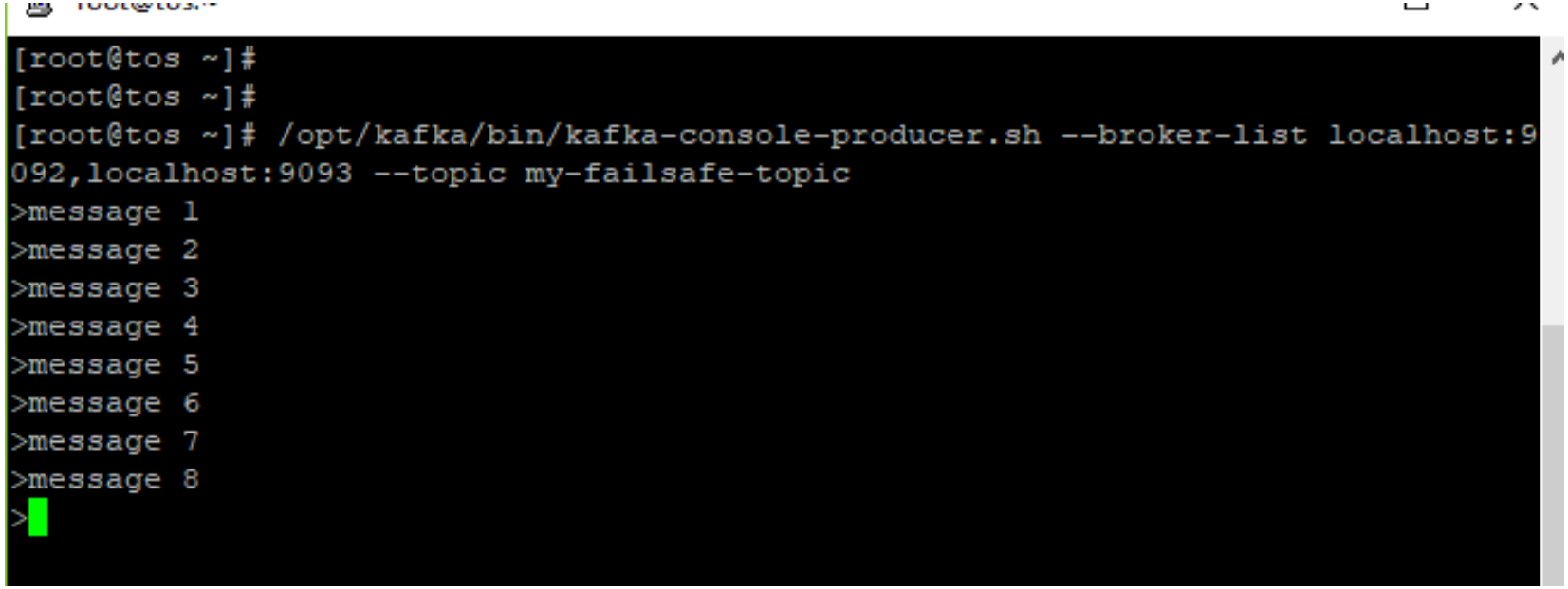
```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9094,kafka0:9092
--topic my-failsafe-topic   --consumer-property group.id=mygroup
```

Notice that the script is the same as before except we added `--consumer-property group.id=mygroup` which will put every consumer that runs with this script into the `mygroup` consumer group.
Now we just run the producer and three consumers.

Run Producer Console

```
/opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092,kafka0:9093 --topic
my-failsafe-topic
```

Now send eight messages from the Kafka producer console.
Producer Console



Notice that the messages are spread evenly among the consumers.

1st Kafka Consumer gets m3, m5

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 5
message 6
message 7
```

Notice the first consumer gets messages m3 and m5.

2nd Kafka Consumer gets m2, m6

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 2
message 4
message 8
```

Notice the second consumer gets messages m2 and m6.

3rd Kafka Consumer gets m1, m4, m7

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 1
message 3
```

Notice the third consumer gets messages m1, m4 and m7.

Notice that each consumer in the group got a share of the messages.

Kafka Consumer Failover

Next, let's demonstrate consumer failover by killing one of the consumers and sending seven more messages. Kafka should divide up the work to the consumers that are running. First, kill the third consumer (CTRL-C in the consumer terminal does the trick).

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 1
message 3
^CProcessed a total of 2 messages
[root@tos ~]#
```

Now send seven more messages from the Kafka console-producer.

Producer Console - send seven more messages m8 through m14

```
[root@tos ~]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9
092,localhost:9093 --topic my-failsafe-topic
>message 1
>message 2
>message 3
>message 4
>message 5
>message 6
>message 7
>message 8
>
>message 9
>m 10
>m11
>m 12
>m 13
>m 14
>m 15
>
```

Notice that the messages are spread evenly among the remaining consumers.

1st Kafka Consumer gets m8, m9, m11, m14

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 2
message 4
message 8

message 9
m 12
m 13
m 14
```

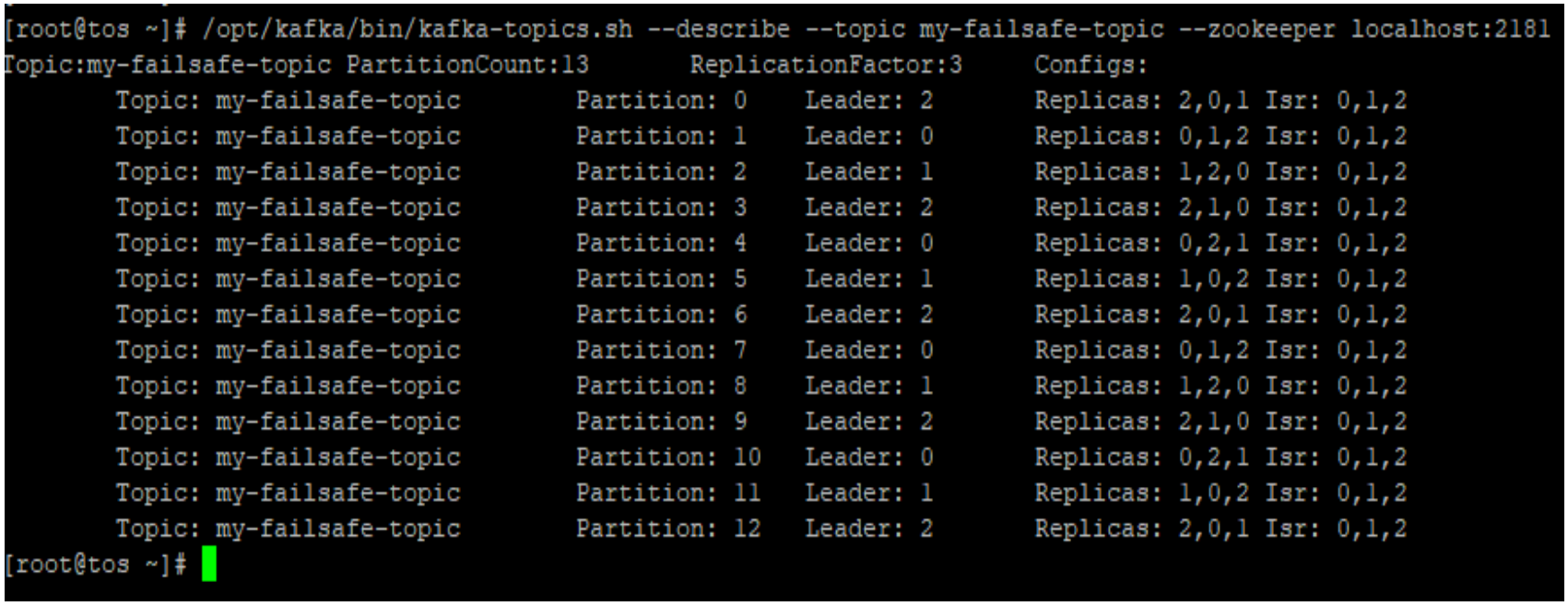
2nd Kafka Consumer gets m10, m12, m13

```
[root@tos ~]#  /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server local
host:9094,localhost:9092 --topic my-failsafe-topic    --consumer-property group.
id=mygroup
message 5
message 6
message 7
m 10
m11
m 15
```

We killed one consumer, sent seven more messages, and saw Kafka spread the load to remaining consumers. ***Kafka consumer failover works!***

Create Kafka Describe Topic Script
You can use kafka-topics.sh to see how the Kafka topic is laid out among the Kafka brokers. The ---describe will show partitions, ISRs, and broker partition leadership.

```
#/opt/kafka/bin/kafka-topics.sh --describe --topic my-failsafe-topic --bootstrap-server kaf
ka0:9092
```



Run describe-topics
We are going to lists which broker owns (leader of) which partition, and list replicas and ISRs of each partition. ISRs are replicas that are up to date. Remember there are 13 topics.

Topology of Kafka Topic Partition Ownership
Notice how each broker gets a share of the partitions as leaders and followers. Also, see how Kafka replicates the partitions on each broker.

Test Broker Failover by killing 1st server
Let's kill the first broker, and then test the failover.

Kill the first broker

```
$ kill `ps aux | grep java | grep server-0.properties | tr -s " " | cut -d " " -f2`
```

```
[2018-05-17 17:38:07,157] INFO [ThrottledRequestReaper-Request]: Shutdown
completed (kafka.server.ClientQuotaManager$ThrottledRequestReaper)
[2018-05-17 17:38:07,158] INFO [SocketServer brokerId=0] Shutting down soc
ket server (kafka.network.SocketServer)
[2018-05-17 17:38:07,267] INFO [SocketServer brokerId=0] Shutdown complete
d (kafka.network.SocketServer)
[2018-05-17 17:38:07,286] INFO [KafkaServer id=0] shut down completed (kaf
ka.server.KafkaServer)
[root@tos kafka-config]#
```

You can stop the first broker by hitting CTRL-C in the broker terminal or by running the above command.
Now that the first Kafka broker has stopped, let's use Kafka topics describe to see that new leaders were elected!

Run describe-topics again to see leadership change

```
[root@tos ~]# /opt/kafka/bin/kafka-topics.sh --describe --topic my-failsafe-topic --zookeeper localhost:2181
Topic:my-failsafe-topic PartitionCount:13       ReplicationFactor:3     Configs:
        Topic: my-failsafe-topic        Partition: 0    Leader: 2       Replicas: 2,0,1 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 1    Leader: 1       Replicas: 0,1,2 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 2    Leader: 1       Replicas: 1,2,0 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 3    Leader: 2       Replicas: 2,1,0 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 4    Leader: 2       Replicas: 0,2,1 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 5    Leader: 1       Replicas: 1,0,2 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 6    Leader: 2       Replicas: 2,0,1 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 7    Leader: 1       Replicas: 0,1,2 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 8    Leader: 1       Replicas: 1,2,0 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 9    Leader: 2       Replicas: 2,1,0 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 10   Leader: 2       Replicas: 0,2,1 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 11   Leader: 1       Replicas: 1,0,2 Isr: 1,2
        Topic: my-failsafe-topic        Partition: 12   Leader: 2       Replicas: 2,0,1 Isr: 1,2
[root@tos ~]#
```

Notice how Kafka spreads the leadership over the 2nd and 3rd Kafka brokers.

Create consolidated scripts as mention below. It will help you to start zookeeper and kafka using a single script.
Start 3 Brokers.
#vi /opt/scripts/start3Brokers.sh
#!/usr/bin/env bash
Start Zookeeper.
/opt/zookeeper/bin/zkServer.sh start

#Start Kafka Server 0
/opt/kafka/bin/kafka-server-start.sh "/opt/kafka-config/config/server-0.properties" &

#Start Kafka Server 1
/opt/kafka/bin/kafka-server-start.sh "/opt/kafka-config/config/server-1.properties" &

```
#Start Kafka Server 2
/opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-2.properties" &

echo "Start zookeeper & 3 Brokers Successfully"


to Stop All 3 Brokers.
# vi stop3Brokers.sh
#!/usr/bin/env bash
# Stop Zookeeper & 3 Servers.
/opt/zookeeper/bin/zkServer.sh stop
/opt/kafka/bin/kafka-server-stop.sh
echo "Stop zookeeper & 3 servers Successfully"

---------------------------------- Lab Ends here ------------------
```

8. Securing Kafka ACL – 60 Minutes

Kafka SASL/PLAIN without SSL

We will perform the following activity in a single node broker only. We will enable ACL in a single broker.
To run a secure broker, two steps need to be performed.

1. Configure the Kafka brokers with ACL using JAAS file
2. Kafka Clients – Pass the Credentials

First, we need to let the broker know authorized users' credentials. This will be stored in a JAAS file.

1.Configure the Kafka brokers and Kafka Clients

Add a JAAS configuration file for each Kafka broker. Create a kafka_plain_jaas.conf file as specified below:

Prepare the /opt/kafka/config/kafka_jaas.conf with below contents. Use vi editor.

```
KafkaServer {
org.apache.kafka.common.security.plain.PlainLoginModule required
   username="admin"
    password="admin"
```

```
    user_admin="admin"
    user_alice="alice"
    user_bob="bob"
    user_charlie="charlie";
};
```

Let's understand the content of ***kafka_plain_jaas.conf*** file and how Kafka Brokers and Kafka Clients use it.

KafkaServer Section:

The KafkaServer section defines four users: admin, alice, bob and charlie. The properties username and password are used by the broker to initiate connections to other brokers. In this example, admin is the user for inter-broker communication. The set of properties user_{userName} defines the passwords for all users that connect to the broker and the broker validates all client connections including those from other brokers using these properties.

This file needs to be passed in as a JVM config option when running the broker, using -Djava.security.auth.login.config=[path_to_jaas_file].

Create an initialize scripts in /opt/scripts/kafkasecurity.sh file with below contents to setup kafka_jaas.conf file location

#vi /opt/scripts/kafkasecurity.sh

Update with the following content.

```
#!/usr/bash
export KAFKA_HOME=/opt/kafka
export ZOOKEEPER_HOME=/opt/zookeeper
export KAFKA_PLAIN_PARAMS="-
Djava.security.auth.login.config=/opt/kafka/config/kafka_jaas.conf"
export KAFKA_OPTS="$KAFKA_PLAIN_PARAMS $KAFKA_OPTS"
export PATH="$PATH:$ZOOKEEPER_HOME/bin:$KAFKA_HOME/bin:$JAVA_HOME/bin"
```

Use the following interpreter in case of ***bash*** interpreter issue.

#!/bin/bash

Second, the following needs to be added to the broker properties file (e.g. server.properties) it defines the accepted protocol and also the ACL authorizer used by the broker:

Step 4: Update ***/opt/kafka/config/server.properties*** for all brokers. In our case only one broker.

```
# Use below for SASL/PLAIN only (No SSL)
# Using SASL_PLAINTEXT as we do not have SSL
authorizer.class.name=kafka.security.authorizer.AclAuthorizer

listeners=SASL_PLAINTEXT://kafka0:9092
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
advertised.listeners=SASL_PLAINTEXT://kafka0:9092
super.users=User:admin
```

```
#     listeners = PLAINTEXT://your.host.name:9092
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
listeners=SASL_PLAINTEXT://tos.master.com:9092
security.inter.broker.protocol= SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
# Hostname and port the broker will advertise to producers and consumers. If not
 set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the v
alue
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=SASL_PLAINTEXT://tos.master.com:9092

super.users=User:admin
```

Comment the following line.

```
# Using SASL_PLAINTEXT as we do not have SSL
#authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
listeners=SASL_PLAINTEXT://localhost:9092
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
advertised.listeners=SASL_PLAINTEXT://localhost:9092
super.users=User:admin


#listeners=PLAINTEXT://0.0.0.0:9092,PLAINTEXT_HOST://0.0.0.0:8082
#advertised.listeners=PLAINTEXT://localhost:9092,PLAINTEXT_HOST://localhost:8082
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
```

This is to initialize the environment HOME.
#cd /opt/scripts
#source ./kafkasecurity.sh

Before starting the server, let us clean the zookeeper/kafka log data so that there won't be any conflict with configuration of the previous cluster ***(/opt/data/zookeeper*** and (***/opt/data/kafka-logs).***

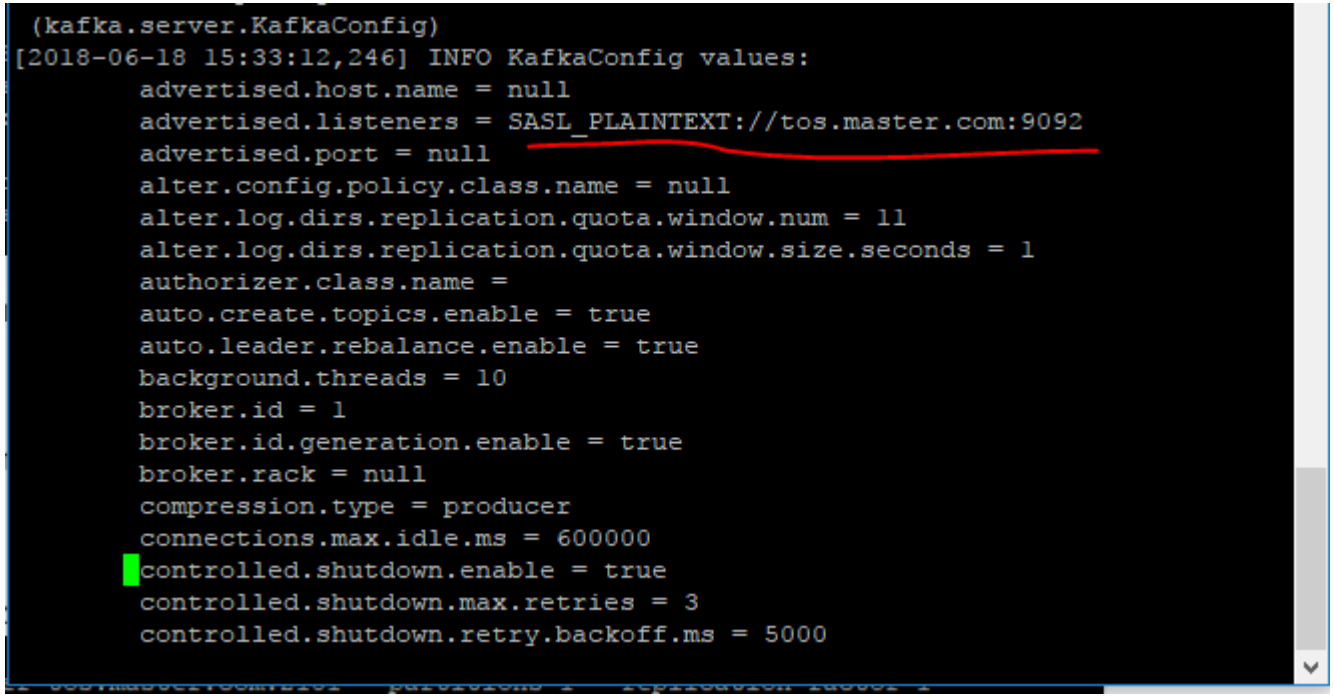
```
[root@tos zookeeper]# ls
version-2  zookeeper_server.pid
[root@tos zookeeper]# pwd
/opt/data/zookeeper
[root@tos zookeeper]# ls
version-2  zookeeper_server.pid
[root@tos zookeeper]# rm -fr *
[root@tos zookeeper]#
```

Delete the data folder of the zookeeper as shown above

Restart ZK/Kafka Services or
Start our broker (from the console that executed : source ./ kafkasecurity.sh)
sh startABroker.sh

```
[root@tos scripts]# sh startABroker.sh
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
Started Successfully
[root@tos scripts]# jps
2551 Kafka
2584 Jps
2286 QuorumPeerMain
[root@tos scripts]#
```

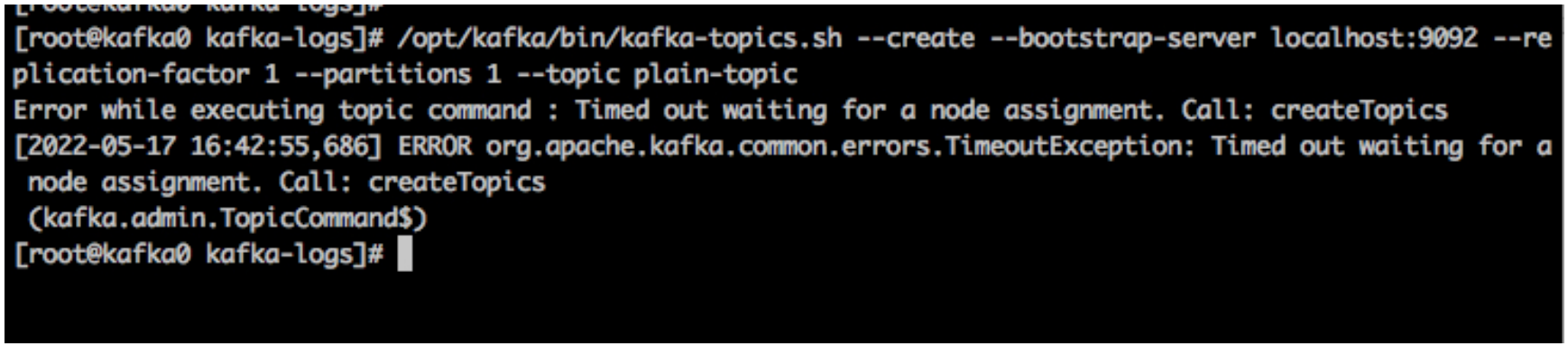
Validate the logs: ***/opt/kafka/logs/server.log*** in case there is any error.
Note : You can include sh startABroker.sh inside the kafkasecurity file in case the environments are not set properly in some OS.



All the commands shown below should be executed from the path */opt/scripts*.

Create a topic:

#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic plain-topic



It should generate an error like as shown above.

Run Kafka console producer.

Before running the Kafka console Producer configure the producer.properties file as shown:

vi producer.properties
Update with the following entries.

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="alice" \
 password="alice";
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
```

----- Code Ends Here // Exclude this line.

#/opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic plain-topic --producer.config producer.properties

Send some messages in the producer console.

```
Message 1
Message 2
Message 3
```

```
[2020-08-30 19:14:56,662] WARN [Producer clientId=console-producer] Error while fetching metadata with corr
elation id 3 : {plain-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2020-08-30 19:14:56,668] ERROR [Producer clientId=console-producer] Topic authorization failed for topics
[plain-topic] (org.apache.kafka.clients.Metadata)
[2020-08-30 19:14:56,670] ERROR Error when sending message to topic plain-topic with key: null, value: 9 by
tes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [plain-topic]
>[2020-08-30 19:14:56,768] WARN [Producer clientId=console-producer] Error while fetching metadata with cor
relation id 4 : {plain-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2020-08-30 19:14:56,769] ERROR [Producer clientId=console-producer] Topic authorization failed for topics
[plain-topic] (org.apache.kafka.clients.Metadata)
[2020-08-30 19:14:56,769] ERROR Error when sending message to topic plain-topic with key: null, value: 9 by
tes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [plain-topic]
>[2020-08-30 19:14:56,872] WARN [Producer clientId=console-producer] Error while fetching metadata with cor
relation id 5 : {plain-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2020-08-30 19:14:56,873] ERROR [Producer clientId=console-producer] Topic authorization failed for topics
[plain-topic] (org.apache.kafka.clients.Metadata)
[2020-08-30 19:14:56,873] ERROR Error when sending message to topic plain-topic with key: null, value: 9 by
tes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [plain-topic]
>
```

Error since we have not authorized to access the topic.

The security configuration still does not give specific permissions to our Kafka users (except for admin who is a super user). These permissions are defined using the ACL command (bin/kafka-acls.sh). To verify existing ACLs run:

#cd /opt/scripts
#vi admin.properties

Update the following content in it.
// Begin - Exclude it

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="admin" \
 password="admin";
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
```

// Ends Here – Exclude it

Verify the ACL configure using the Admin credentials.

#/opt/kafka/bin/kafka-acls.sh --bootstrap-server kafka0:9092 --command-config admin.properties --list

This returns no ACL definitions. You have handled authentication, but have not provided any authorization rules to define the users able run specific APIs and access certain Kafka resources.

```
[root@kafka0 kafka-logs]# cd /opt/scripts/
[root@kafka0 scripts]# vi admin.properties
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config admin.properties --li
st
[root@kafka0 scripts]#
```

Assign the permission:

#/opt/kafka/bin/kafka-acls.sh --bootstrap-server kafka0:9092 --command-config admin.properties --add --allow-principal User:alice --operation All --topic plain-topic

```
[root@kafka0 config]# cd /opt/scripts
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh  --bootstrap-server localhost:9092 --command-config admin.properties --
add --allow-principal User:alice --operation All --topic plain-topic
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:alice, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

[root@kafka0 scripts]#
```

Now you can verify the acl granted to alice user

```
#/opt/kafka/bin/kafka-acls.sh --bootstrap-server kafka0:9092 --command-config
admin.properties --list
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config admin.properties --l
ist
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:alice, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

[root@kafka0 scripts]#
```

Now you can try sending message again from the producer console. Type one or two messages in the producer console.

```
# cd /opt/scripts
#/opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic plain-
topic --producer.config producer.properties
```

Send some messages in the producer console.

```
Hello Message
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic plain-topic --producer
.config producer.properties
>Hello
>Message
>
```

Now, it should be able to sent message to the topic as shown above.
Now we can try consuming these messages further from the consumer console, open a separate terminal for this.

Run Kafka console consumer

Consuming from Topic using bob user, hence grant ACL to bob.
Next, you need to let user bob consume (or fetch) from topic plain-topic using the Fetch API, as a member of the bob-group consumer group. Bob's ACL for fetching from topic test is:
Principal bob is Allowed Operation Read From Host * On Topic plain-topic .
or

```
# /opt/kafka/bin/kafka-acls.sh  --bootstrap-server=kafka0:9092 -command-config
admin.properties --add --allow-principal User:bob --operation Read --topic plain-topic
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh  --bootstrap-server=localhost:9092 -command-config admin.properties --a
dd --allow-principal User:bob --operation Read --topic plain-topic
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

[root@kafka0 scripts]#
```

Bob needs a second ACL for committing offsets to group bob-group (using the OffsetCommit API):

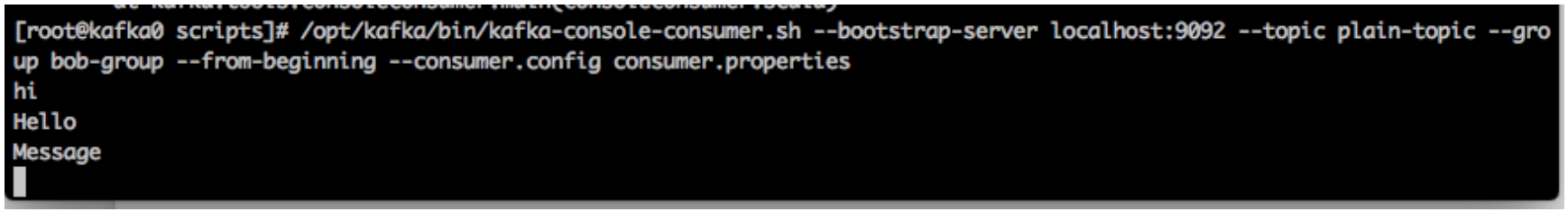
Principal bob is Allowed Operation Read From Host * On Group bob-group.
or
$ /opt/kafka/bin/kafka-acls.sh --bootstrap-server=kafka0:9092 -command-config admin.properties --add --allow-principal User:bob --operation Read --group bob-group

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh  --bootstrap-server=localhost:9092 -command-config admin.properties --a
dd --allow-principal User:bob --operation Read --group bob-group
Adding ACLs for resource `ResourcePattern(resourceType=GROUP, name=bob-group, patternType=LITERAL)`:
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=bob-group, patternType=LITERAL)`:
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)

[root@kafka0 scripts]#
```

By granting these permissions to Bob, he can now consume messages from topic test as a member of bob-group.
Before running Kafka console consumer configure the consumer.properties file as shown:

```
# vi consumer.properties

sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="bob" \
 password="bob";
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
```

$ /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic plain-topic --group bob-group --from-beginning --consumer.config consumer.properties



Sent a few messages from the producer console, and you will see that messages are being consume on the console...

Describing consumer groups

Open a new console for the Charlie user.
Lastly, user charlie needs permission to retrieve committed offsets from group bob-group (using the OffsetFetch API). According to the table above, Charlie's first ACL for fetching offsets from this consumer group is:

Principal charlie is Allowed Operation Describe From Host * On Group bob-group.

or

$ /opt/kafka/bin/kafka-acls.sh --bootstrap-server=kafka0:9092 -command-config admin.properties --add --allow-principal User:charlie --operation Describe --group bob-group

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh --bootstrap-server=localhost:9092 -command-config admin.properties --ad
d --allow-principal User:charlie --operation Describe --group bob-group
Adding ACLs for resource `ResourcePattern(resourceType=GROUP, name=bob-group, patternType=LITERAL)`:
        (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=bob-group, patternType=LITERAL)`:
        (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)

[root@kafka0 scripts]#
```

Principal charlie is Allowed Operation Describe From Host * On Topic test.

or

$ /opt/kafka/bin/kafka-acls.sh --bootstrap-server=kafka0:9092 -command-config admin.properties --add --allow-principal User:charlie --operation Describe --topic plain-topic

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh --bootstrap-server=localhost:9092 -command-config admin.properties --ad
d --allow-principal User:charlie --operation Describe --topic plain-topic
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
        (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)
        (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

[root@kafka0 scripts]#
```

The following file contains the client configuration for group.

vi group.properties

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="charlie" \
 password="charlie";
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
```

Now Charlie is able to get the proper listing of offsets in the group:

```
$ /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server kafka0:9092 --describe --
group bob-group --command-config group.properties
```

```
[root@tos scripts]# vi group.properties
[root@tos scripts]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.
com:9092 --describe --group bob-group --command-config group.properties
Note: This will not show information about old Zookeeper-based consumers.

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
                                HOST            CLIENT-ID
plain-topic     0          5               5               0               consumer-1-6450
d497-85b1-4113-80bc-985c4045b185 /10.10.20.24    consumer-1
[root@tos scripts]#
```

or

```
[root@kafka0 scripts]# vi group.properties
[root@kafka0 scripts]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group bob-grou
p --command-config group.properties

Consumer group 'bob-group' has no active members.

GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            C
LIENT-ID
bob-group       plain-topic     0          4               4               0               -               -               -
[root@kafka0 scripts]#
```

As you can see above the details of the consumer group, Here the log end offset is showing as 3/5 that is the current offset are same with that of the end offset. This is an ideal cluster, where there is no lagging in the production and consumption.

The above ACLs grants enough permissions for this use case to run.
To summarize, configured ACLs execute the following command.

$ /opt/kafka/bin/kafka-acls.sh --bootstrap-server=kafka0:9092 --command-config admin.properties --list

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-acls.sh --bootstrap-server=localhost:9092  --command-config admin.properties --list
Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=bob-group, patternType=LITERAL)`:
       (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)
       (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=plain-topic, patternType=LITERAL)`:
       (principal=User:bob, host=*, operation=READ, permissionType=ALLOW)
       (principal=User:charlie, host=*, operation=DESCRIBE, permissionType=ALLOW)
       (principal=User:alice, host=*, operation=WRITE, permissionType=ALLOW)
       (principal=User:alice, host=*, operation=ALL, permissionType=ALLOW)

[root@kafka0 scripts]#
```

Errors:

```
[root@tos scripts]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server
tos.master.com:9092 --topic plain-topic --from-beginning --consumer.config consu
mer.properties
[2018-06-27 21:37:46,223] WARN [Consumer clientId=consumer-1, groupId=console-co
nsumer-47559] Error while fetching metadata with correlation id 2 : {plain-topic
=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2018-06-27 21:37:46,232] ERROR Unknown error when running consumer:  (kafka.too
ls.ConsoleConsumer$)
org.apache.kafka.common.errors.GroupAuthorizationException: Not authorized to ac
cess group: console-consumer-47559
```

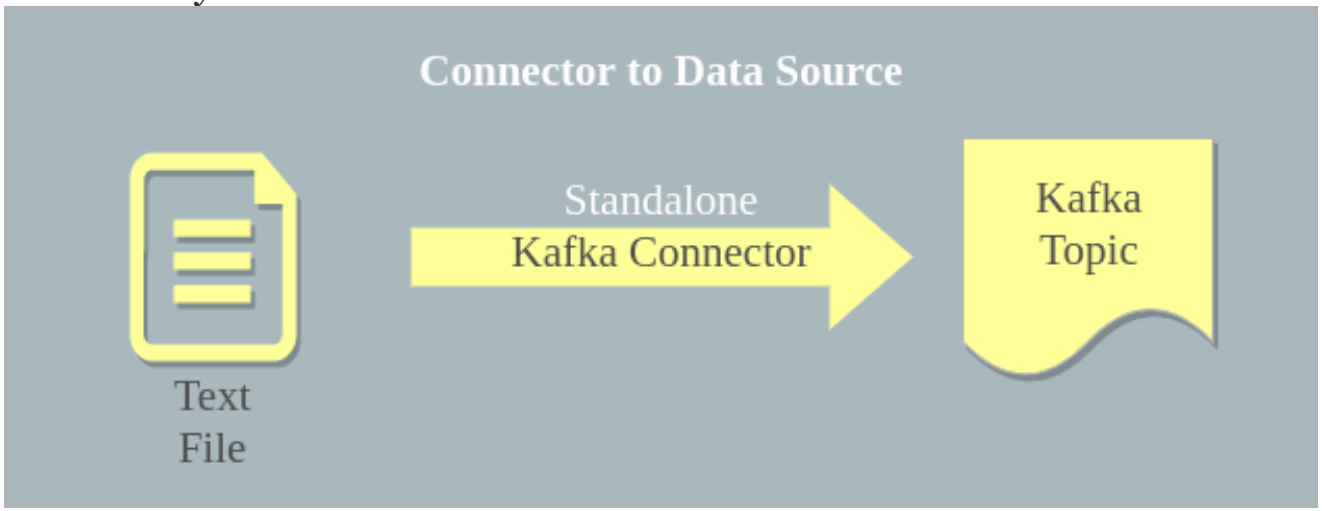
Grant proper authorization.
Ensure that kafkasecurity.sh is only executed for the server start up scripts not in any of the other console.

---- ---------- ----------

Lab Ends Here

9. Kafka Connector (File) - 60 Minutes

Apache Kafka Connector – Connectors are the components of Kafka that could be setup to listen the changes that happen to a data source like a file or database, and pull in those changes automatically.



When working with Kafka you might need to write data from a local file to a Kafka topic. This is actually very easy to do with Kafka Connect. Kafka Connect is a framework that provides scalable and reliable streaming of data to and from Apache Kafka. With Kafka Connect, writing a file's content to a topic requires only a few simple steps

Starting Kafka and Zookeeper
We will use the standalone Broker this example.
#cd /opt/scripts
#sh startABroker.sh

```
[root@tos scripts]# ls
custom-reassignment.json  server2.log       stopZookeeper.sh
old.jso                   start3Brokers.sh  topics-to-move.json
server0.log               startABroker.sh   zookeeper.out
server1.log               stop3Brokers.sh
[root@tos scripts]# sh startABroker.sh
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
Started Successfully
[root@tos scripts]# jps
2209 QuorumPeerMain
2474 Kafka
2493 Jps
[root@tos scripts]#
```

Creating a Topic to Write to, the message will be fetch from the file and publish to the topic.

```
#cd /opt/scripts
#/opt/kafka/bin/kafka-topics.sh \
 --create \
 --bootstrap-server kafka0:9092 \
 --replication-factor 1 \
 --partitions 1 \
 --topic my-kafka-connect
```

```
[root@tos bin]#
[root@tos bin]# cd /opt/scripts
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh \
>    --create \
>    --zookeeper tos.master.com:2181 \
>    --replication-factor 1 \
>    --partitions 1 \
>    --topic my-kafka-connect
Created topic "my-kafka-connect".
[root@tos scripts]#
```

Creating a Source Config File

Since we are reading the contents of a local file and writing to Kafka, this file is considered our “source”. Therefore, we will use the FileSource connector. We must create a configuration file to use with this connector. For this most part you can copy the example available in $KAFKA_HOME/config/connect-file-source.properties. Below is an example of our my-file-source.properties file.

Open a new file.
#vi my-file-source.properties

Paste the following instruction in the above file.

```
#my-file-source.properties config file
name=local-file-source
connector.class=com.github.mmolimar.kafka.connect.fs.FsSourceConnector
policy.class=com.github.mmolimar.kafka.connect.fs.policy.SimplePolicy
tasks.max=1
fs.uris=file:///tmp/my-test.txt
topic=my-kafka-connect
file_reader.class=com.github.mmolimar.kafka.connect.fs.file.reader.TextFileReader
file_reader.batch_size=0
```

This file indicates that we will use the FileStreamSource connector class, read data from the /tmp/my-test.txt file, and publish records to the my-kafka-connect Kafka topic. We are also only using 1 task to push this data to Kafka, since we are reading/publishing a single file.

Creating a Worker Config File.

Processes that execute Kafka Connect connectors and tasks are called workers. Since we are reading data from a single machine and publishing to Kafka, we can use the simpler of the two types, standalone workers (as opposed to distributed workers). You can find a sample config file for standalone workers in $KAFKA_HOME/config/connect-standalone.properties. We will call our file my-standalone.properties.

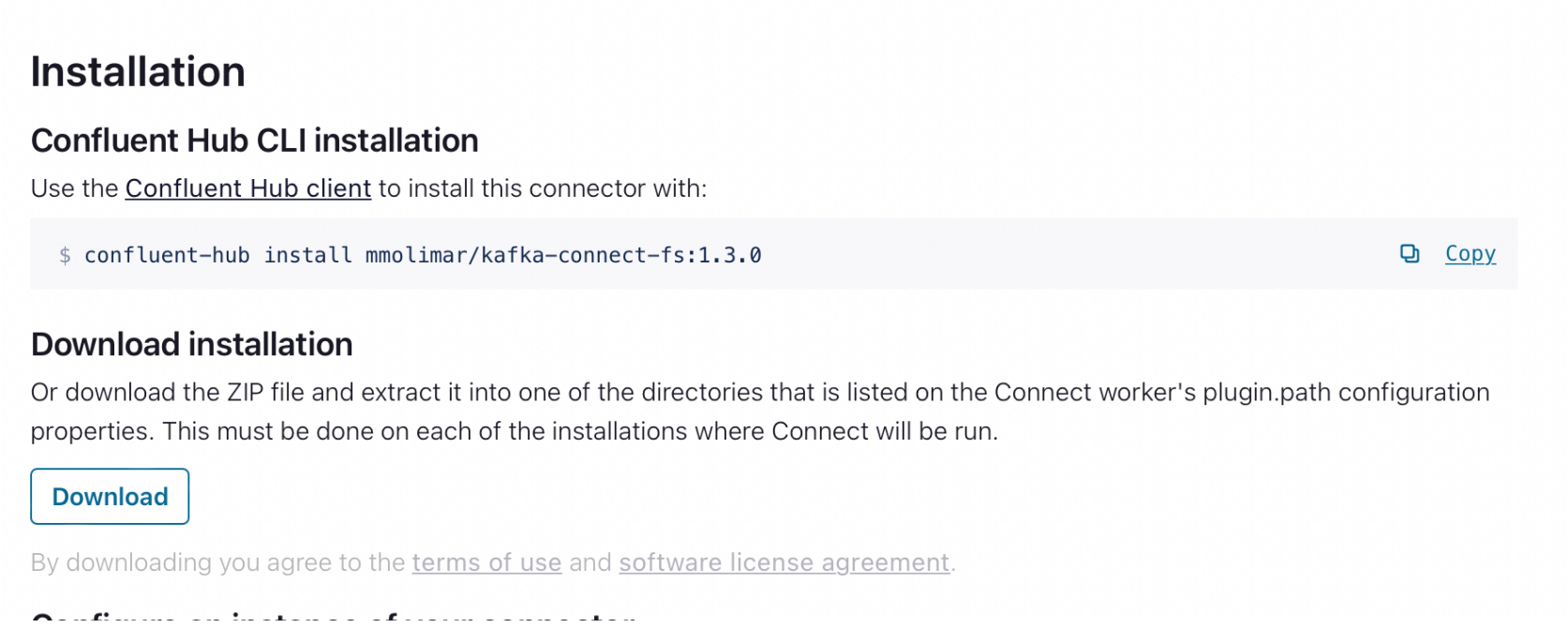
Create a file.
#vi my-standalone.properties

Update with the following contents.

```
#bootstrap kafka servers
bootstrap.servers=kafka0:9092

# specify input data format
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter

# local file storing offsets and config data
offset.storage.file.filename=/tmp/connect.offsets
plugin.path=/software/plugins
```

The main change in this example in comparison to the default is the key.converter and value.converter settings. Since our file contains simple text, we use the StringConverter types.

Download the file connector.
https://www.confluent.io/hub/mmolimar/kafka-connect-fs



Installation

Confluent Hub CLI installation

Use the Confluent Hub client to install this connector with:

```
$ confluent-hub install mmolimar/kafka-connect-fs:1.3.0
```

Copy

Download installation

Or download the ZIP file and extract it into one of the directories that is listed on the Connect worker's plugin.path configuration properties. This must be done on each of the installations where Connect will be run.

Download

By downloading you agree to the terms of use and software license agreement.

Download the above file.
#mkdir -p /software/plugins/

Rename the folder:
#mv mmolimar-kafka-connect-fs-1.3.0/ /software/plugins/kafka-file

Open a terminal.

Our input file /tmp/my-test.txt will be read in a single process to the Kafka my-kafka-connect topic. Here is a look at the file contents:

Create a file and paste the following text.
#vi /tmp/my-test.txt
This Message is from Test File.
It will be consumed by the Kafka Connector.

Running Kafka Connect.

Now it is time to run Kafka Connect with our worker and source configuration files. As mentioned before we will be running Kafka Connect in standalone mode. Here is an example of doing this with our custom config files:

#/opt/kafka/bin/connect-standalone.sh my-standalone.properties my-file-source.properties

```
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:35,218] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:36,226] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:37,241] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:38,250] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:39,256] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:40,258] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:55:41,272] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
```

```
[2018-06-17 12:56:53,991] WARN Couldn't find file /tmp/my-test.txt for FileStrea
mSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.fil
e.FileStreamSourceTask:108)
[2018-06-17 12:56:55,535] INFO Cluster ID: rwk6R3n9TYSCArCsSs8V4w (org.apache.ka
fka.clients.Metadata:265)
[2018-06-17 12:57:53,852] INFO WorkerSourceTask{id=local-file-source-0} Committi
ng offsets (org.apache.kafka.connect.runtime.WorkerSourceTask:328)
[2018-06-17 12:57:53,854] INFO WorkerSourceTask{id=local-file-source-0} flushing
 0 outstanding messages for offset commit (org.apache.kafka.connect.runtime.Work
erSourceTask:345)
[2018-06-17 12:57:54,062] INFO WorkerSourceTask{id=local-file-source-0} Finished
 commitOffsets successfully in 208 ms (org.apache.kafka.connect.runtime.WorkerSo
urceTask:427)
```

Open another terminal and execute the following consumer to consume the message.

Reading from the Kafka Topic
If we read from the Kafka topic that we created earlier, we should see the 2 lines in the source file that were written to Kafka:

#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic my-kafka-connect --from-beginning

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server tos.ma
ster.com:9092 --topic my-kafka-connect --from-beginning
This Message is from Test File.
It will be consumed by the Kafka Connector.
```

We have successfully configured, file connector in standalone mode.

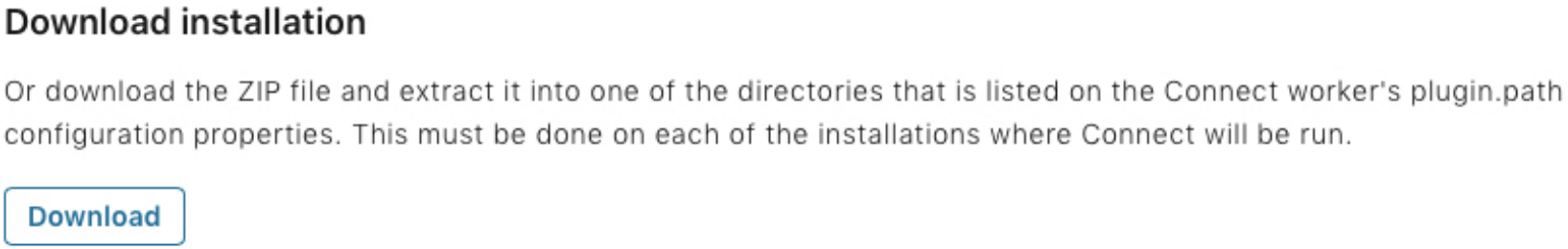
Next, let us configure JDBC in a distributed mode.

10. Kafka Connector (JDBC) - 60 Minutes

Kafka Connect – JDBC Connector.

Download the connector from the following url.

Tools - https://www.confluent.io/hub/confluentinc/kafka-connect-jdbc



Extract the file:
yum install unzip
unzip confluent*zip

```
[root@4f5607f478bd software]# pwd
/software
[root@4f5607f478bd software]# ls
README.md  confluentinc-kafka-connect-jdbc-10.0.0  hbase-2.3.2-bin.tar  log.txt  master.txt
[root@4f5607f478bd software]# ls confluentinc-kafka-connect-jdbc-10.0.0/
assets  doc  etc  lib  manifest.json
[root@4f5607f478bd software]#
```

Rename the folder:
#mv confluentinc-kafka-connect-jdbc-10.0.0/ /software/plugins/kafka-jdbc

Add this to the plugin path in your Connect properties file.
For example, `plugin.path=/software/plugins`. Kafka Connect finds the plugins using its plugin path.

Start the Connect workers with that configuration. Connect will discover all connectors defined within those plugins.

We can create create /software/connect-distributed.properties file to specify the worker properties as follows:

Note that the `plugin.path` is the path that we need to place the library that we downloaded.
#vi /software/connect-distributed.properties

Update the following content in the above file.

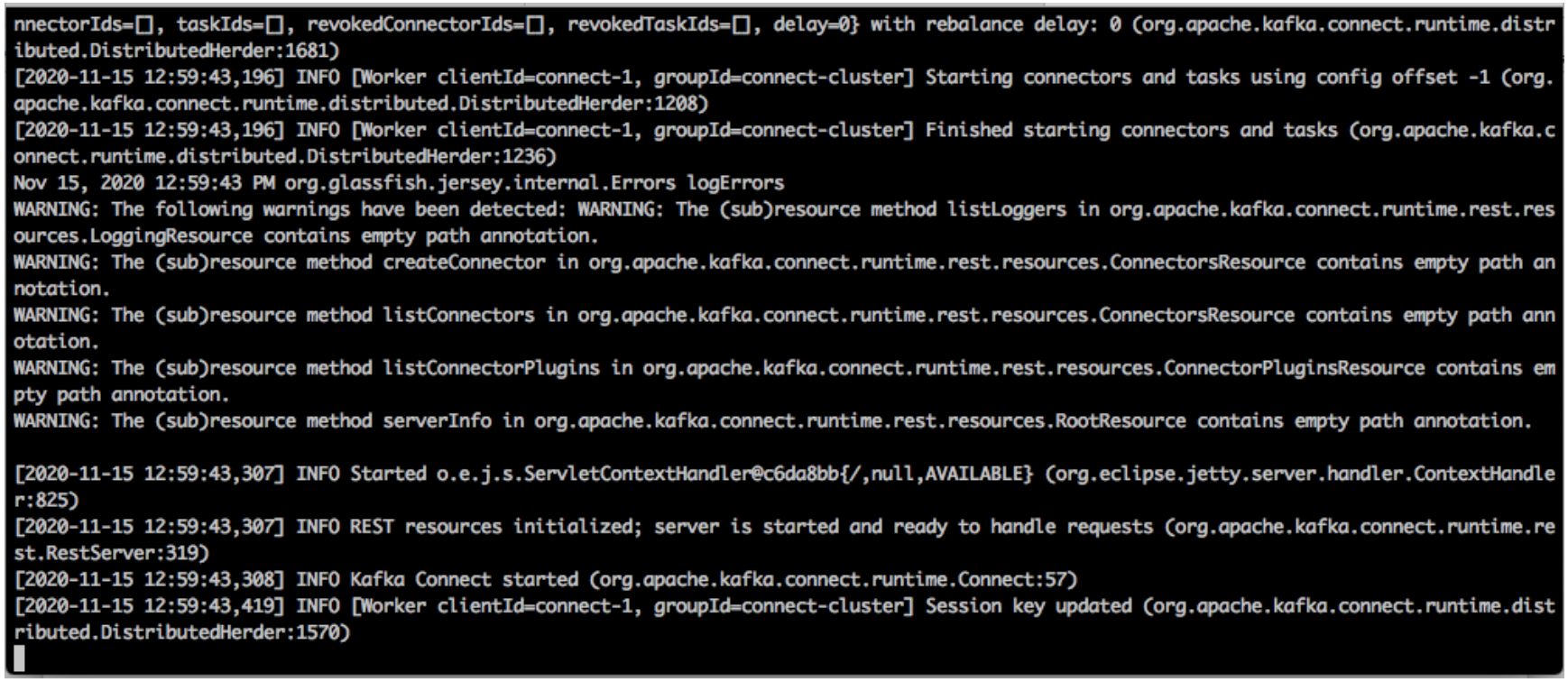
```
# A list of host/port pairs to use for establishing the initial connection to the Kafka cluster.
bootstrap.servers= kafka0:9092

# unique name for the cluster, used in forming the Connect cluster group. Note that this must not conflict with consumer group IDs
group.id=connect-cluster

# The converters specify the format of data in Kafka and how to translate it into Connect data.
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true

# Topic to use for storing offsets. This topic should have many partitions and be replicated and compacted.
offset.storage.topic=connect-offsets
offset.storage.replication.factor=1

# Topic to use for storing connector and task configurations; note that this should be a single partition, highly replicated,
config.storage.topic=connect-configs
config.storage.replication.factor=1

# Topic to use for storing statuses. This topic can have multiple partitions and should be replicated and compacted.
status.storage.topic=connect-status
status.storage.replication.factor=1

# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000

plugin.path=/software/plugins
```

Note that the `plugin.path` is the path that we need to place the library that we downloaded.

Start the kafka connector

Go to the bin folder of the kafka installation or specify the full path.

```
#/opt/kafka/bin/connect-distributed.sh /software/connect-distributed.properties
```



After running the connector we can confirm that connector's REST endpoint is accessible, and we can confirm that JDBC connector is in the plugin list by calling http://localhost:8083/connector-plugins

curl http://localhost:8083/connector-plugins

```
[root@4f5607f478bd software]# curl http://localhost:8083/connector-plugins
[{"class":"io.confluent.connect.jdbc.JdbcSinkConnector","type":"sink","version":"10.0.0"},{"class":"io.confluent.connect.jdbc.JdbcSourceConnec
tor","type":"source","version":"10.0.0"},{"class":"org.apache.kafka.connect.file.FileStreamSinkConnector","type":"sink","version":"2.6.0"},{"c
lass":"org.apache.kafka.connect.file.FileStreamSourceConnector","type":"source","version":"2.6.0"},{"class":"org.apache.kafka.connect.mirror.M
irrorCheckpointConnector","type":"source","version":"1"},{"class":"org.apache.kafka.connect.mirror.MirrorHeartbeatConnector","type":"source","
version":"1"},{"class":"org.apache.kafka.connect.mirror.MirrorSourceConnector","type":"source","version":"1"}][root@4f5607f478bd software]#
[root@4f5607f478bd software]#
```

Install the postgress db. We will configure postgressdb to store some data which will be transfer by connect to kafka topic.

Refer url for downloading the postgresdb software : https://www.postgresql.org/download/.

For Centos 7:

[If you are using docker, initiate a container for PostgressDB:
#docker run -it --name postgresdb --privileged -p 5432:5432 centos:7 /usr/sbin/init
docker exec -it postgresdb bash
Add both the container in same network.
#docker network connect spark-net postgresdb
]

The following command will install and initialize the postgres DB
#yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Or

yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-aarch64/pgdg-redhat-repo-latest.noarch.rpm

#yum install -y postgresql13-server
#/usr/pgsql-13/bin/postgresql-13-setup initdb

Enable and start the service:

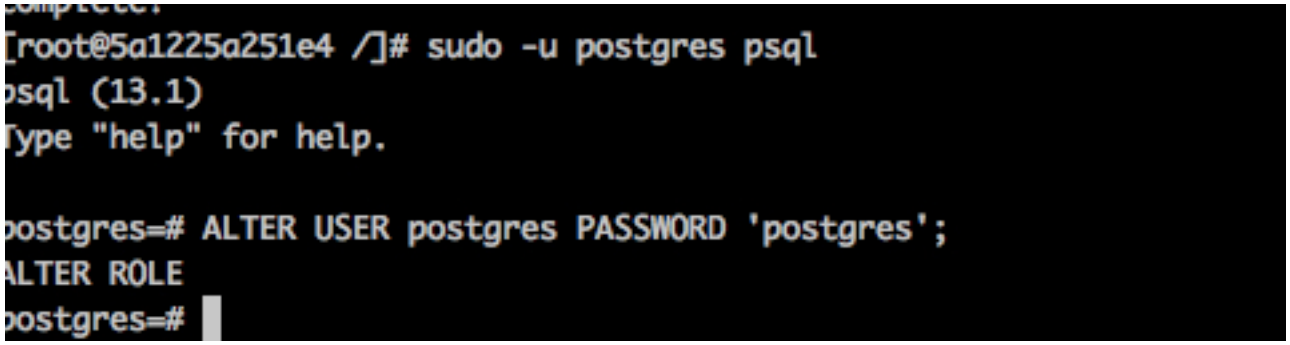
systemctl enable postgresql-13
systemctl start postgresql-13

The configuration for the plugin is stored in /software/jdbc-source.json file on kafka broker server.
It contents is as follows: (replace localhost with container IP of postgress DB if you are using docker)

```
{
    "name": "jdbc_source_connector_postgresql_01",
    "config": {
        "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
        "connection.url": "jdbc:postgresql://localhost:5432/postgres",
        "connection.user": "postgres",
        "connection.password": "postgres",
        "topic.prefix": "postgres-01-",
        "poll.interval.ms" : 3600000,
        "mode":"bulk"
    }
}
```

Start Postgress DB CLI: On the postgres DB server.

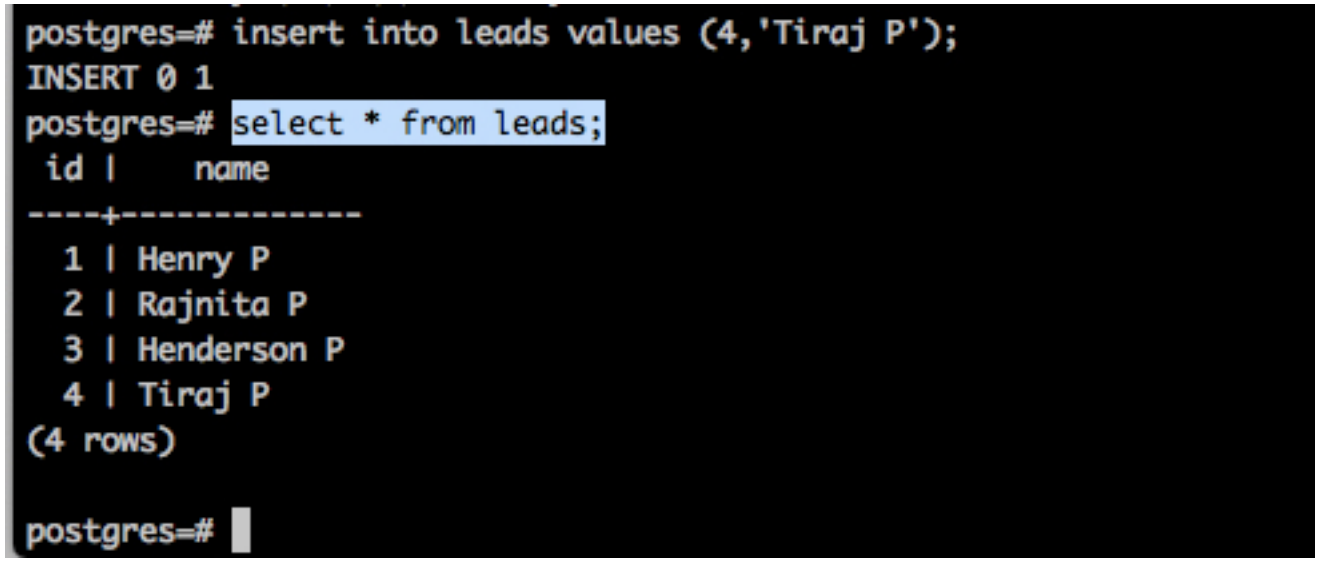
sudo -u postgres psql
ALTER USER postgres PASSWORD 'postgres';



if sudo doesn't exists in your system, install (yum install sudo)

Create a table and insert few records: Execute all the bellows SQL commands from the Postgresdb CLI.

#**CREATE TABLE** leads (id INTEGER **PRIMARY KEY**, name VARCHAR);
insert into leads values (1,'Henry P');
insert into leads values (2,'Rajnita P');
insert into leads values (3,'Henderson P');
insert into leads values (4,'Tiraj P');
select * from leads;



To exit : \q

Allow all to connect to postgresDB from remote server by updating the following contents:

#vi /var/lib/pgsql/13/data/**postgresql.conf**

listen_addresses = '*'

Append in the last line of the following file.

vi /var/lib/pgsql/13/data/pg_hba.conf

host all all all trust

#systemctl restart postgresql-13

Starting the JDBC Connector – On the Kafka Node.

As we operate on distributed mode we run the connectors by calling REST endpoints with the configuration JSON. We can specify the configuration payload from a file for curl command. The following command starts the connector. Execute the following command from the directory you have store the configuration json file.

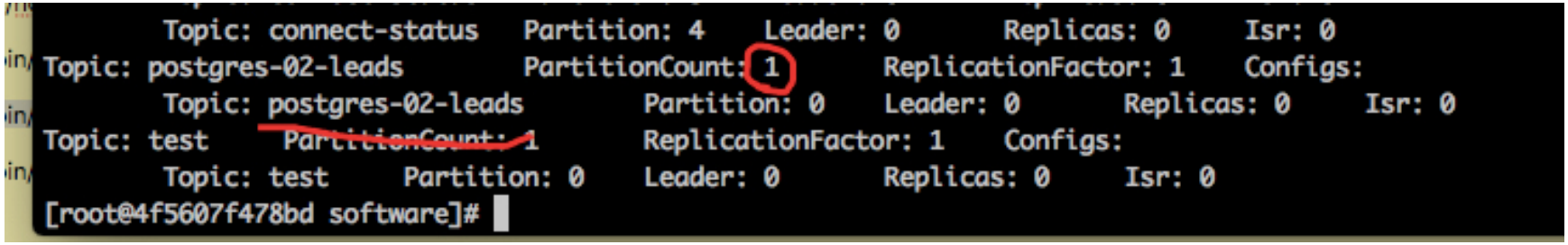
```
curl -d @"jdbc-source.json" \
-H "Content-Type: application/json" \
-X POST http://localhost:8083/connectors
```

```
        timestamp.column.name = □
        timestamp.delay.interval.ms = 0
        timestamp.initial = null
        topic.prefix = postgres-02-
        validate.non.null = true
 (io.confluent.connect.jdbc.source.JdbcSourceTaskConfig:354)
[2020-11-15 16:05:29,313] INFO Using JDBC dialect PostgreSql (io.confluent.connect.jdbc.source.JdbcSourceTask:98)
[2020-11-15 16:05:29,317] INFO Attempting to open connection #1 to PostgreSql (io.confluent.connect.jdbc.util.CachedConnectionProvider:82)
[2020-11-15 16:05:29,383] INFO Started JDBC source task (io.confluent.connect.jdbc.source.JdbcSourceTask:257)
[2020-11-15 16:05:29,384] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} Source task finished initialization and start (org.a
pache.kafka.connect.runtime.WorkerSourceTask:233)
[2020-11-15 16:05:29,388] INFO Begin using SQL query: SELECT * FROM "public"."leads" (io.confluent.connect.jdbc.source.TableQuerier:164)
[2020-11-15 16:05:29,503] WARN [Producer clientId=connector-producer-jdbc_source_connector_postgresql_04-0] Error while fetching metadata with
 correlation id 3 : {postgres-02-leads=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient:1073)
[2020-11-15 16:05:39,266] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} Committing offsets (org.apache.kafka.connect.runtime
.WorkerSourceTask:478)
[2020-11-15 16:05:39,268] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} flushing 0 outstanding messages for offset commit (o
rg.apache.kafka.connect.runtime.WorkerSourceTask:495)
[2020-11-15 16:05:39,295] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} Finished commitOffsets successfully in 28 ms (org.ap
ache.kafka.connect.runtime.WorkerSourceTask:574)
[2020-11-15 16:05:49,264] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} Committing offsets (org.apache.kafka.connect.runtime
.WorkerSourceTask:478)
[2020-11-15 16:05:49,265] INFO WorkerSourceTask{id=jdbc_source_connector_postgresql_04-0} flushing 0 outstanding messages for offset commit (o
rg.apache.kafka.connect.runtime.WorkerSourceTask:495)
```

We can see that in postgres database table leads is loaded to kafka topic- postgres-02-leads:

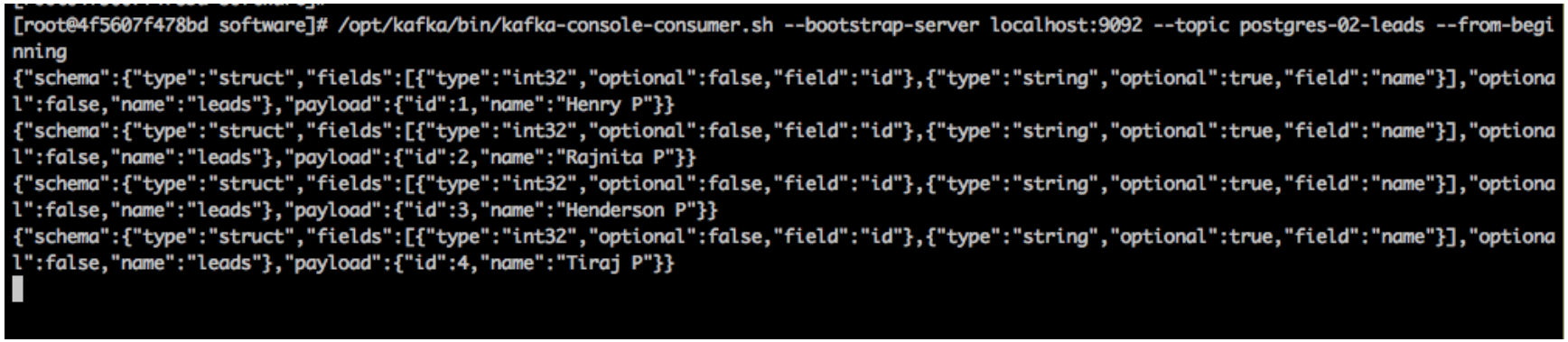
#/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092

#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092 --topic postgres-01-leads


Topic: connect-status Partition: 4 Leader: 0 Replicas: 0 Isr: 0
Topic: postgres-02-leads PartitionCount: 1 ReplicationFactor: 1 Configs:
Topic: postgres-02-leads Partition: 0 Leader: 0 Replicas: 0 Isr: 0
Topic: test PartitionCount: 1 ReplicationFactor: 1 Configs:
Topic: test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
[root@4f5607f478bd software]#

And each row in the tables are loaded as a message. You can verify using the consumer console.

#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic postgres-01-leads --from-beginning



Let us perform a Single Message Transformation.

Scenario: **Name** field should be changed to **FullName**

Solution: We will use the ReplaceField to perform the above activity.

Add another instance of connector.

Define the Single Message Tranformation parameters.

The configuration for the plugin is stored in /software/jdbc-source_smt.json file on kafka broker server. It contents is as follows: (replace localhost with container IP of postgress DB if you are using docker)

```
{
    "name": "jdbc_source_connector_postgresql_02",
    "config": {
        "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
        "connection.url": "jdbc:postgresql://localhost:5432/postgres",
        "connection.user": "postgres",
        "connection.password": "postgres",
        "topic.prefix": "postgres-02-",
        "poll.interval.ms" : 3600000,
        "mode":"bulk",
       "transforms": "RenameField",
       "transforms.RenameField.type": "org.apache.kafka.connect.transforms.ReplaceField$Value",
       "transforms.RenameField.renames": "name:fullname"
    }
}
```

In the above configuration, the field , name is replace with fullname attribute.

Start the instance

```
#curl -d @"jdbc-source_smt1.json" \
-H "Content-Type: application/json" \
-X POST http://localhost:8083/connectors
```

Verify the topic and check the message inside that topic.

#/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic postgres-02-leads --from-beginning

As shown above the message field has been transform from Name to fullname.

Execute some Queries to get information about connectors:

Get a list of active connectors.

#curl http://localhost:8083/connectors

Get configuration info for a connector.

#curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/config

```
[root@kafka0 /]# curl http://localhost:8083/connectors
["jdbc_source_connector_postgresql_01","jdbc_source_connector_postgresql_02"][root@kafka0 /]#
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/config
{"connector.class":"io.confluent.connect.jdbc.JdbcSourceConnector","mode":"bulk","topic.prefix":"postgres-01-","connection.
password":"postgres","connection.user":"postgres","poll.interval.ms":"3600000","name":"jdbc_source_connector_postgresql_01"
,"connection.url":"jdbc:postgresql://localhost:5432/postgres"}[root@kafka0 /]#
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_02/config
{"connector.class":"io.confluent.connect.jdbc.JdbcSourceConnector","mode":"bulk","transforms.RenameField.renames":"name:ful
lname","topic.prefix":"postgres-02-","connection.password":"postgres","connection.user":"postgres","poll.interval.ms":"3600
000","transforms.RenameField.type":"org.apache.kafka.connect.transforms.ReplaceField$Value","transforms":"RenameField","nam
e":"jdbc_source_connector_postgresql_02","connection.url":"jdbc:postgresql://localhost:5432/postgres"}[root@kafka0 /]#
[root@kafka0 /]#
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_02/tasks
[{"id":{"connector":"jdbc_source_connector_postgresql_02","task":0},"config":{"connector.class":"io.confluent.connect.jdbc.
JdbcSourceConnector","transforms.RenameField.renames":"name:fullname","connection.password":"postgres","transforms":"Rename
Field","mode":"bulk","topic.prefix":"postgres-02-","tables":"\"public\".\"leads\"","task.class":"io.confluent.connect.jdbc.
source.JdbcSourceTask","connection.user":"postgres","poll.interval.ms":"3600000","transforms.RenameField.type":"org.apache.
kafka.connect.transforms.ReplaceField$Value","name":"jdbc_source_connector_postgresql_02","connection.url":"jdbc:postgresql
[root@kafka0 /]#
```

Retrieve details for specific tasks (JDBC Tasks) - Get a list of tasks currently running for the connector.

#curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_02/tasks

```
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_02/tasks
[{"id":{"connector":"jdbc_source_connector_postgresql_02","task":0},"config":{"connector.class":"io.confluent.connect.jdbc.
JdbcSourceConnector","transforms.RenameField.renames":"name:fullname","connection.password":"postgres","transforms":"Rename
Field","mode":"bulk","topic.prefix":"postgres-02-","tables":"\"public\".\"leads\"","task.class":"io.confluent.connect.jdbc.
source.JdbcSourceTask","connection.user":"postgres","poll.interval.ms":"3600000","transforms.RenameField.type":"org.apache.
kafka.connect.transforms.ReplaceField$Value","name":"jdbc_source_connector_postgresql_02","connection.url":"jdbc:postgresql
://localhost:5432/postgres"}}][root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/ta
[{"id":{"connector":"jdbc_source_connector_postgresql_01","task":0},"config":{"connector.class":"io.confluent.connect.jdbc.
JdbcSourceConnector","mode":"bulk","topic.prefix":"postgres-01-","tables":"\"public\".\"leads\"","connection.password":"pos
tgres","task.class":"io.confluent.connect.jdbc.source.JdbcSourceTask","connection.user":"postgres","poll.interval.ms":"3600
000","name":"jdbc_source_connector_postgresql_01","connection.url":"jdbc:postgresql://localhost:5432/postgres"}}][root@kafk
a0 /]#
```

The current status of the connector
#curl http://localhost:8083/connectors/?expand=status

```
[root@kafka0 /]# curl http://localhost:8083/connectors/?expand=status
{"jdbc_source_connector_postgresql_01":{"status":{"name":"jdbc_source_connector_postgresql_01","connector":{"state":"RUNNIN
G","worker_id":"172.18.0.2:8083"},"tasks":[{"id":0,"state":"RUNNING","worker_id":"172.18.0.2:8083"}],"type":"source"}},"jdb
c_source_connector_postgresql_02":{"status":{"name":"jdbc_source_connector_postgresql_02","connector":{"state":"RUNNING","w
orker_id":"172.18.0.2:8083"},"tasks":[{"id":0,"state":"RUNNING","worker_id":"172.18.0.2:8083"}],"type":"source"}}}[root@kaf
ka0 /]#
```

Gets the current status of the connector, including:

- Whether it is running or restarting, or if it has failed or paused
- Which worker it is assigned to
- Error information if it has failed
- The state of all its tasks

#curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/status

```
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/status
{"name":"jdbc_source_connector_postgresql_01","connector":{"state":"RUNNING","worker_id":"172.18.0.2:8083"},"tasks":[{"id":
0,"state":"RUNNING","worker_id":"172.18.0.2:8083"}],"type":"source"}[root@kafka0 /]#
[root@kafka0 /]#
```

Retrieve details for specific tasks

curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/tasks/0/status

```
[root@kafka0 /]# curl http://localhost:8083/connectors/jdbc_source_connector_postgresql_01/tasks/0/status
{"id":0,"state":"RUNNING","worker_id":"172.18.0.2:8083"}[root@kafka0 /]#
```

---------- Labs End Here ----------

12. Errors

I. {test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)

```
[2018-05-15 23:46:40,132] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 14 : {test=LEADER_NOT_AVAILABLE} (org.apac
he.kafka.clients.NetworkClient)
[2018-05-15 23:46:40,266] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 15 : {test=LEADER_NOT_AVAILABLE} (org.apac
he.kafka.clients.NetworkClient)
^C[2018-05-15 23:46:40,394] WARN [Producer clientId=console-producer] Error whil
e fetching metadata with correlation id 16 : {test=LEADER_NOT_AVAILABLE} (org.ap
ache.kafka.clients.NetworkClient)
[root@tos opt]#  {test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkCl
ient)
bash: syntax error near unexpected token `org.apache.kafka.clients.NetworkClient
```

Solutions: /opt/kafka/config/server.properties
Update the following information.

```
# it uses the value for "listeners" if configured.  Otherwise, it will use the v
alue
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://localhost:9092
```

II Unable to connect to the server / `Topic my-example-topic not present in metadata after 60000 ms.`

```
############################# Socket Server Settings #############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Hostname and port the broker will advertise to producers and consumers. If not set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://localhost:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more d
ls
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

"server.properties" 136L, 6848C written
```

13. LOG verification + segment Sizing

14. Annexure Code:

II. DumplogSegment

/opt/kafka/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files \
/tmp/kafka-logs/my-kafka-connect-0/oooooooooooooooooo.log | head -n 4

```
[root@tos test-topic-0]# more 00000000000000000000.log
[root@tos test-topic-0]# cd ../
[root@tos kafka-logs]# cd my-kafka-connect-0/
[root@tos my-kafka-connect-0]# ls
00000000000000000000.index      00000000000000000011.snapshot
00000000000000000000.log        leader-epoch-checkpoint
00000000000000000000.timeindex
[root@tos my-kafka-connect-0]# more *log
\▒▒▒▒▒▒▒▒afka Connector.--More--(53%)

[root@tos my-kafka-connect-0]# pwd
/tmp/kafka-logs/my-kafka-connect-0
[root@tos my-kafka-connect-0]# /opt/kafka/bin/kafka-run-class.sh kafka.tools.Dum
pLogSegments --deep-iteration --print-data-log --files \
> /tmp/kafka-logs/my-kafka-connect-0/00000000000000000000.log | head -n 4
Dumping /tmp/kafka-logs/my-kafka-connect-0/00000000000000000000.log
Starting offset: 0
offset: 0 position: 0 CreateTime: 1530552634675 isvalid: true keysize: -1 values
ize: 31 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence:
-1 isTransactional: false headerKeys: [] payload: This Message is from Test File
.
offset: 1 position: 0 CreateTime: 1530552634677 isvalid: true keysize: -1 values
ize: 43 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence:
-1 isTransactional: false headerKeys: [] payload: It will be consumed by the Kaf
ka Connector.
[root@tos my-kafka-connect-0]#
```

CLI to list offset number.

#/opt/kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic IBM

Sending Message with Key and Value

#/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic IBM --property "parse.key=true" --property "key.separator=:"

#/opt/kafka/bin/Kafka-console-consumer.sh --topic IBM --bootstrap-server localhost:9092 --from-beginning \
 --property print.key=true \
 --property key.separator=":" \
 --partition 4 \
 --offset 3

```
kafka-console-consumer --topic example-topic --bootstrap-server broker:9092 \
 --from-beginning \
 --property print.key=true \
 --property key.separator="-" \
 --partition 0
```

III. Resources

https://developer.ibm.com/hadoop/2017/04/10/kafka-security-mechanism-saslplain/
https://sharebigdata.wordpress.com/2018/01/21/implementing-sasl-plain/
https://developer.ibm.com/code/howtos/kafka-authn-authz
https://jaceklaskowski.gitbooks.io/apache-kafka/content/kafka-tools-GetOffsetShell.html