

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	24
6.	Zookeeper – 120 Minutes	27
7.	Kafka cluster – 90 Minutes	46
8.	Administration – 60 Minutes	71
9.	Movement of Partition from a Broker to another Broker – 60 Minutes. (ND)	85
10.	Securing Kafka SSL – 90 Minutes	99
11.	Securing Kafka ACL – 60 Minutes	120
12.	Mirroring data between clusters – MirrorMaker – 90 Minutes	143
13.	Kafka Connector (File & JDBC) - 150 Minutes	154
14.	Schema Registry - Manage Schemas for Topics – 30 Minutes	183
15.	Performance & Tuning – 60 Minutes (ND)	188
16.	Using REST Proxy	210
17.	Using Kubernetes	216
19.	Errors	217
I.	{test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)	217

20. LOG verification + segment Sizing.....	219
21. Annexure Code:	220
II. DumplogSegment.....	220
III. Resources	221

Hardware:

8 GB RAM , 30 GB HDD , Centos 7 or above OS. Access to internet. (3 Machine each per participants)

Software Inventory:

- Zookeeper Version: apache-zookeeper-3.6.1-bin.tar
- Apache kafka : 2.13-3.1.0
- jdk-8u45-linux-x64.tar.gz (Any jdk above 1.8)
- Eclipse for Linux. (Any Latest version for JEE Development)
- Status : Color is Verified

Last Updated: Feb 26 - 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, kafkao.

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, kafkao and mount host folder in /opt

```
#docker network create --driver bridge spark-net
```

4 Kafka – Administration

```
#docker run --name kafkao --hostname kafkao -p 9092:9092 -p 8081:8081 -p 2181:2181 -p 9999:9999 -i -t --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt centos:7 /usr/sbin/init
```

Optional:

```
#docker run --name kafkao --hostname kafkao -p 9092:9092 -p 9081:8081 -p 9082:8082 -p 3181:2181 -p 9990:9999 -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
```

You can verify the container from a separate terminal:

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

5 Kafka – Administration

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.

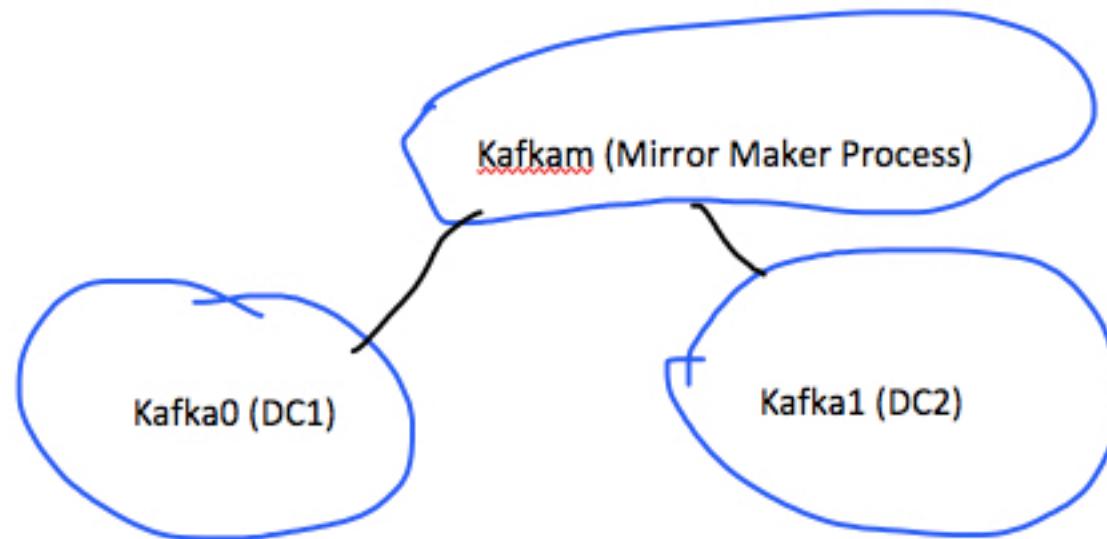
Kafka Mirror Maker: (It will execute the Kafka Mirroring process)

```
#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:

6 Kafka – Administration

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://kafkao:9092,PLAINTEXT_HOST://localhost:8081
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
```

// Changes End

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

2. Installation of Kafka – 90 Mins

You need to install java before installing zookeeper and Kafka.

Installation of Java

```
#tar -xvf jdk-8u45-linux-x64.tar.gz -C /opt
```

Set the above path in the PATH variable and JAVA_HOME

```
export JAVA_HOME=/opt/jdk
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

Include in the profile as follow:

```
#vi ~/.bashrc
```

9 Kafka – Administration

```
[root@tos opt]# more ~/.bashrc
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

export JAVA_HOME=/opt/jdk1.8.0_45

export PATH=$PATH:$JAVA_HOME/bin
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
[root@tos opt]#
```

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-3.6.1-bin.tar -C /opt  
#mv a*zookeeper-3.6.1* /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
```

```
[root@tos opt]# /opt/zookeeper/bin/zkServer.sh start  
ZooKeeper JMX enabled by default  
Using config: /opt/zookeeper/bin/.../conf/zoo.cfg  
Starting zookeeper ... STARTED  
[root@tos opt]# █
```

Option II (Part of the Apache Kafka)

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

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`:

```
#telnet localhost 2181
```

```
➤ srvr
```

```
[root@tos opt]# telnet localhost 2181
Trying :1...
Connected to localhost.
Escape character is '^].
srvr
Zookeeper version: 3.4.12-e5259e437540f349646870ea94dc2658c4e44b3b, built on 03/
27/2018 03:55 GMT
Latency min/avg/max: 0/0/0
Received: 2
Sent: 1
Connections: 1
Outstanding: 0
Zxid: 0x0
Mode: standalone
Node count: 4
Connection closed by foreign host.
[root@tos opt]#
```

After zookeeper installation, let us install the Kafka Broker.

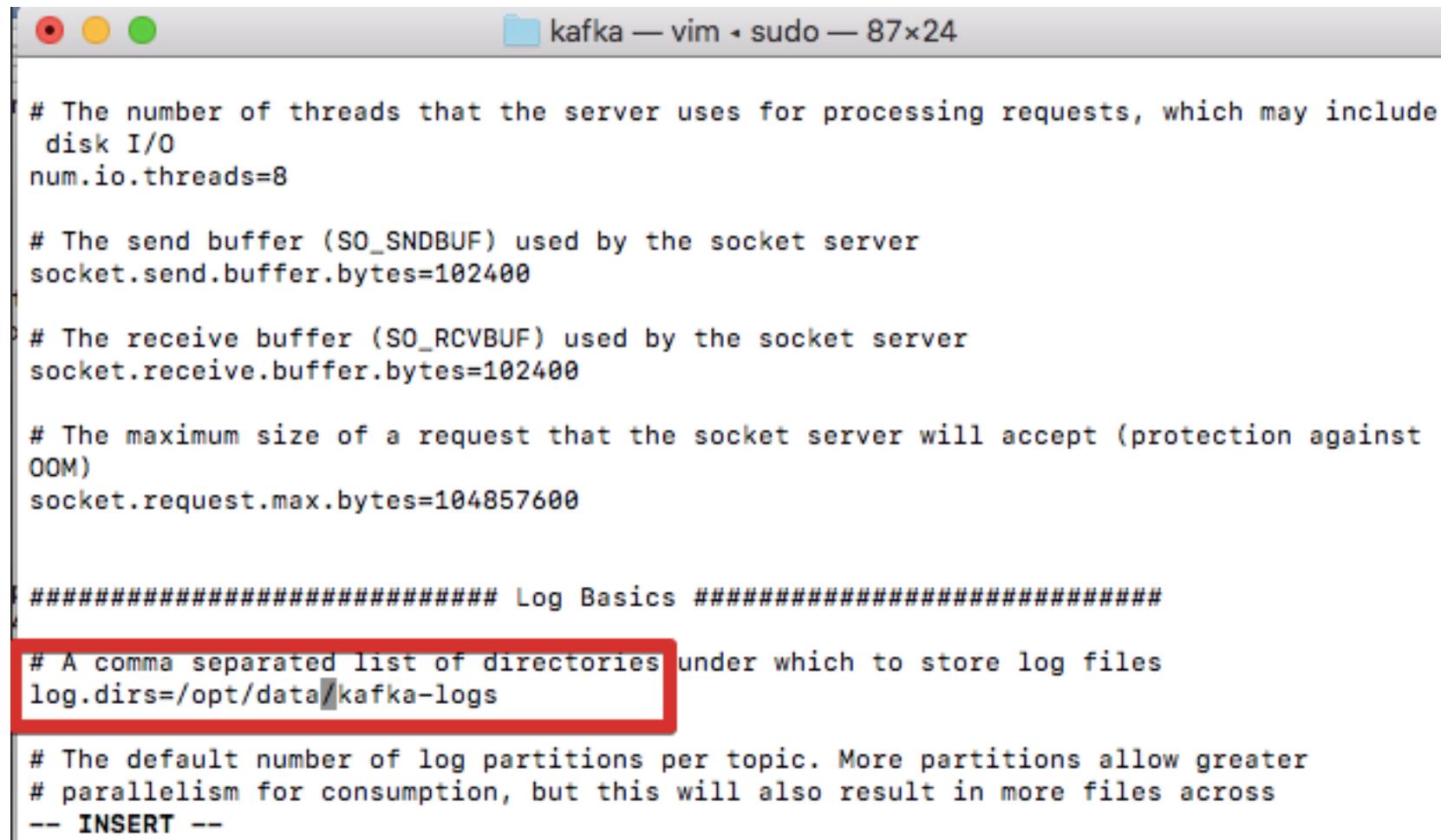
Installation of Kafka Broker

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 -zxf kafka_2.12-* -C /opt
# mv kafka_2.1* /opt/kafka
# mkdir /opt/data/kafka-logs
```

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



```
# The number of threads that the server uses for processing requests, which may include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600

#####
##### Log Basics #####
#####

# A comma separated list of directories under which to store log files
log.dirs=/opt/data/kafka-logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
-- INSERT --
```

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
```

```
[root@tos opt]# /opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
[root@tos opt]# jps
3476 Kafka
3499 Jps
2895 QuorumPeerMain
[root@tos opt]#
```

```
#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.

```
#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"
```

Lab Installation completes End here.

3. Installation Confluent Kafka (Local) – 30 Minutes

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

Installing Confluent kafka

Inflate the confluent kafka compress file as shown below:

```
#tar -xvf confluent-7.0.1.tar -C /opt
```

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};
```

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

```
[root@tos opt]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test  
Created topic "test".  
[root@tos opt]# /opt/kafka/bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic test  
Topic:test      PartitionCount:1      ReplicationFactor:1      Configs:  
          Topic: test      Partition: 0      Leader: 0      Replicas: 0      Isr: 0  
[root@tos opt]#
```

Verify the no of partition in the output.

list and describe topic.

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

```
[root@tos ~]# /opt/kafka/bin/kafka-topics.sh --zookeeper localhost:2181 --list  
__consumer_offsets  
my-failsafe-topic  
test
```

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 localhost:9092 --topic test
```

List all topics (prints only topic names)

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

```
[root@tos scripts]# jps
12960 Jps
12314 Kafka
12043 QuorumPeerMain
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper tos.master.com:2181 --topic
test
test
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
CustomerCountry
__consumer_offsets
henry-topic
my-failsafe-topic
my-kafka-topic
my-kafka-topic1
test
test-topic
```

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

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

Describe all topics (prints details about the topics)

```
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092
```

```
[root@tos scripts]#
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test
Topic:test      PartitionCount:1          ReplicationFactor:1        Configs:
          Topic: test      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181
Topic:CustomerCountry  PartitionCount:1          ReplicationFactor:1        Configs:
          Topic: CustomerCountry  Partition: 0      Leader: 1      Replicas: 1      Isr: 1
Topic:_consumer_offsets  PartitionCount:50         ReplicationFactor:1        Configs:segment.bytes=104857600,cleanup.policy=compact,compression.type=producer
          Topic: __consumer_offsets   Partition: 0      Leader: 1      Replicas: 1      Isr: 1
          Topic: __consumer_offsets   Partition: 1      Leader: 2      Replicas: 2      Isr: 2
```

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 localhost:9092 --topic topic1 --partitions 2 --replication-factor 1
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic topic1 --partitions 2 --replication-factor 1
Created topic "topic1".
[root@tos scripts]#
```

Create a topic with replication 2.

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

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic topic2 --partitions 2 --replication-factor 2
Error while executing topic command : Replication factor: 2 larger than available brokers: 1.
[2022-01-10 07:44:51,716] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 2 larger than available brokers: 1.
(kafka.admin.TopicCommand$)
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic topic2
Error while executing topic command : Topic 'topic2' does not exist as expected
[2022-01-10 07:45:26,938] ERROR java.lang.IllegalArgumentException: Topic 'topic2' does not exist as expected
    at kafka.admin.TopicCommand$.kafka$admin$TopicCommand$$ensureTopicExists(TopicCommand.scala:542)
    at kafka.admin.TopicCommand$ZookeeperTopicService.describeTopic(TopicCommand.scala:447)
    at kafka.admin.TopicCommand$.main(TopicCommand.scala:69)
    at kafka.admin.TopicCommand.main(TopicCommand.scala)
(kafka.admin.TopicCommand$)
[root@kafka0 /]#
```

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
```

```
[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]#
```

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 localhost:9092 --topic test  
Test Message 1  
Test Message 2  
^D
```

```
#
```

```
[root@tos config]# /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test  
>hi  
>Hello  
>TEst message  
>[root@tos config]#
```

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 localhost:9092 --topic test  
--from-beginning
```

```
[base] Henrys-MacBook-Air:kafka henrypotsangbam$ /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
Test Message 1
Test Message 2
Sending third message
```

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

```
zookeeper --java - sudo — 137x24
2020-08-21 12:40:05,342 [myid:] - INFO  [main:Environment@98] - Client environment:os.memory.free=116MB
2020-08-21 12:40:05,345 [myid:] - INFO  [main:Environment@98] - Client environment:os.memory.max=228MB
2020-08-21 12:40:05,345 [myid:] - INFO  [main:Environment@98] - Client environment:os.memory.total=123MB
2020-08-21 12:40:05,355 [myid:] - INFO  [main:ZooKeeper@1005] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@4926097b
2020-08-21 12:40:05,361 [myid:] - INFO  [main:X509Util@77] - Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation
2020-08-21 12:40:05,378 [myid:] - INFO  [main:ClientCnxnSocket@239] - jute.maxbuffer value is 1048575 Bytes
2020-08-21 12:40:05,389 [myid:] - INFO  [main:ClientCnxn@1703] - zookeeper.request.timeout value is 0. feature.enabled=false
Welcome to ZooKeeper!
2020-08-21 12:40:05,407 [myid:localhost:2181] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1154] - Opening socket connection to server localhost/127.0.0.1:2181.
2020-08-21 12:40:05,407 [myid:localhost:2181] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1156] - SASL config status: Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2020-08-21 12:40:05,505 [myid:localhost:2181] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@986] - Socket connection established, initiating session, client: /127.0.0.1:50738, server: localhost/127.0.0.1:2181
2020-08-21 12:40:05,517 [myid:localhost:2181] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1420] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000075b4060002, negotiated timeout = 30000
WATCHER:::
WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0]
```

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_eve
nt_notification, zookeeper]
[zk: localhost:2181(CONNECTED) 2]

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

[[zookeeper, brokers, consumer_group_metadata, 1, brokers
[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 availabl
e. (org.apache.kafka.clients.NetworkClient)
[2020-08-21 12:52:15,093] 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.NetworkCli
ent)
^CProcessed a total of 0 messages
(base) Henrys-MacBook-Air:zookeeper henrynottingham$
```

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
```

```
*@0 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 /
```

Verify who is the leader:

<http://localhost:8080/commands/>

Click on Leader:

```
{  
    "is_leader" : false,  
    "leader_id" : 2,  
    "leader_ip" : "localhost",  
    "command" : "leader",  
    "error" : null  
}
```

here, ZK node ID 2 is the leader.

You can also 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.

```
config — vi server.properties — 80x24

# The maximum size of a log segment file. When this size is reached a new log se
gment will be created.
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they can be deleted a
ccording
# to the retention policies
log.retention.check.interval.ms=300000

#####
# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# directory within the zk cluster.
zookeeper.connect=localhost:2181,localhost:2182,localhost:2183

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000

@
"server.properties" 136L, 6884C written
```

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

```
[2020-08-24 08:41:05,479] INFO starting (kafka.server.KafkaServer)
[2020-08-24 08:41:05,482] INFO Connecting to zookeeper on localhost:2181,localhost:2182,localhost:2183 (kafka.server.KafkaServer)
[2020-08-24 08:41:05,538] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181,localhost:2182,localhost:2183. (kafka.zookeeper.ZooKeeperClient)
[2020-08-24 08:41:05,564] INFO Client environment:zookeeper.version=3.5.7-f0fdd52973d373ffd9c86b81d99842dc2c7f660e
, built on 02/10/2020 11:30 GMT (org.apache.zookeeper.ZooKeeper)

[2020-08-24 08:47:42,893] INFO Kafka version: 2.5.0 (org.apache.kafka.common.utils.AppInfoParser)
[2020-08-24 08:47:42,893] INFO Kafka commitId: 66563e712b0b9f84 (org.apache.kafka.common.utils.AppInfoParser)
[2020-08-24 08:47:42,895] INFO Kafka startTimeMs: 1598239962881 (org.apache.kafka.common.utils.AppInfoParser)
[2020-08-24 08:47:42,902] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
(base) Henrys-MacBook-Air:~ Henry$
```

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
```

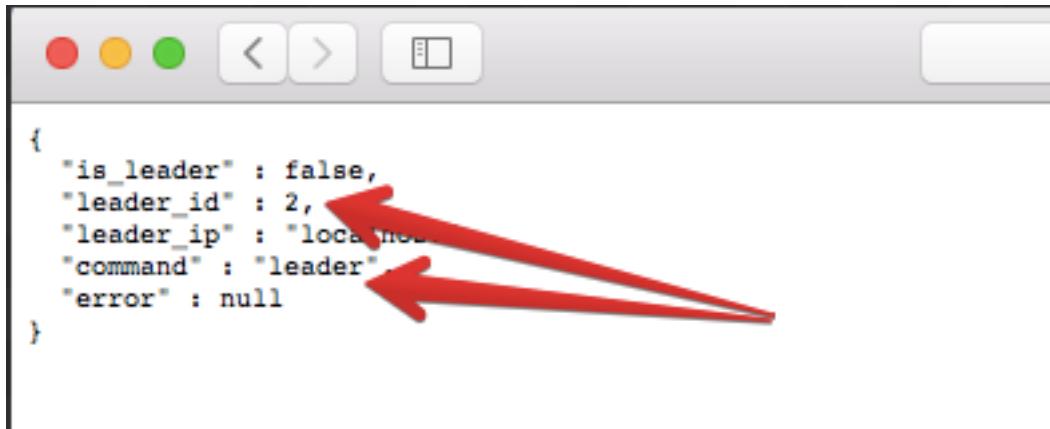
```
[(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testz
>Test Message 1
Test Message 2
[>>(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092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



```
{  
    "is_leader" : false,  
    "leader_id" : 2,  
    "leader_ip" : "localhost",  
    "command" : "leader"  
}  
}
```

```
[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/zoo
keeper/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/..
/lib/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/../*.jar:/opt/zookeeper/bin/../*.jar:/opt/zookeeper/bin/..
/zookeeper-server/src/main/resources/lib/*.jar:/opt/zookeeper/bin/..//conf: -Xmx1000m -com.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.

```
{  
    "is_leader": false,  
    "leader_id": 3,  
    "leader_ip": "localhost",  
    "command": "leader",  
    "error": null  
}
```

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":{}}, "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:participant
server.2=localhost:2889:participant
server.3=localhost:2890: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.properties
[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.properties
[root@tos opt]# cp kafka/config/server.properties kafka-config/config/server-2.properties
[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://tos.master.com:9092
advertised.listeners=PLAINTEXT://tos.master.com:9092
log.dirs=/opt/data/kafka-logs/kafka-0
```

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

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

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

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

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

```
broker.id=2
listeners=PLAINTEXT://tos.master.com:9094
advertised.listeners=PLAINTEXT://tos.master.com: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.

```
[root@tos:/opt/kafka-config] [2018-05-16 15:17:50,404] INFO [GroupCoordinator 2]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2018-05-16 15:17:50,411] INFO [GroupCoordinator 2]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2018-05-16 15:17:50,564] INFO [GroupMetadataManager brokerId=2] Removed 0 expired offsets in 140 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-05-16 15:17:50,637] INFO [ProducerId Manager 2]: Acquired new producerId block (brokerId:2,blockStartProducerId:4000,blockEndProducerId:4999) by writing to Zk with path version 5 (kafka.coordinator.transaction.ProducerIdManager)
[2018-05-16 15:17:50,777] INFO [TransactionCoordinator id=2] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2018-05-16 15:17:50,788] INFO [TransactionCoordinator id=2] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2018-05-16 15:17:51,303] INFO [Transaction Marker Channel Manager 2]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2018-05-16 15:17:51,574] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2018-05-16 15:17:51,723] INFO Kafka version : 1.1.0 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:17:51,723] INFO Kafka commitId : fdcf75ea326b8e07 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:17:51,728] INFO [KafkaServer id=2] started (kafka.server.KafkaServer)
[2018-05-16 15:15:08,676] INFO [Transaction Marker Channel Manager 1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2018-05-16 15:15:09,267] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2018-05-16 15:15:09,281] INFO Kafka version : 1.1.0 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:15:09,282] INFO Kafka commitId : fdcf75ea326b8e07 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:15:09,285] INFO [KafkaServer id=1] started (kafka.server.KafkaServer)
```

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

Create a replicated topic my-failsafe-topic

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 localhost:9092 --replication-factor 3 --partitions 13 --topic my-failsafe-topic
```

```
^C[root@tos ~]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:21  
--replication-factor 3 --partitions 13 --topic my-failsafe-topic  
Created topic "my-failsafe-topic".  
[root@tos ~]#
```

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 tos.master.com:9094,tos.  
master.com: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 localhost: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 tos.master.com:9092,tos.master.com: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.

```
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 localhost:9092,localhost:9093 --topic my-failsafe-topic
>How are you?
>Great Kafka is working
>
```

Consumer Console

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost: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)

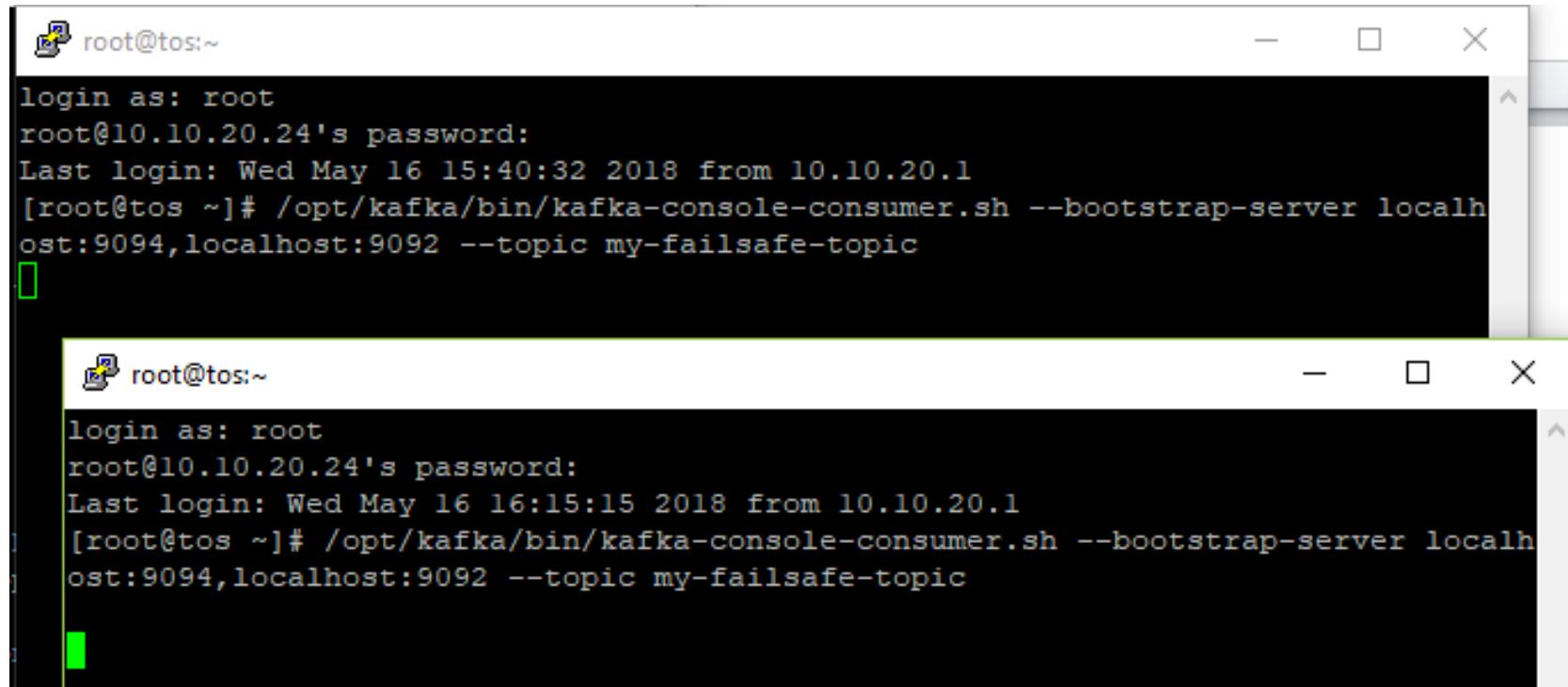
Consumer 1.

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

Consumer 2.

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

The two consumer consoles will be as shown below



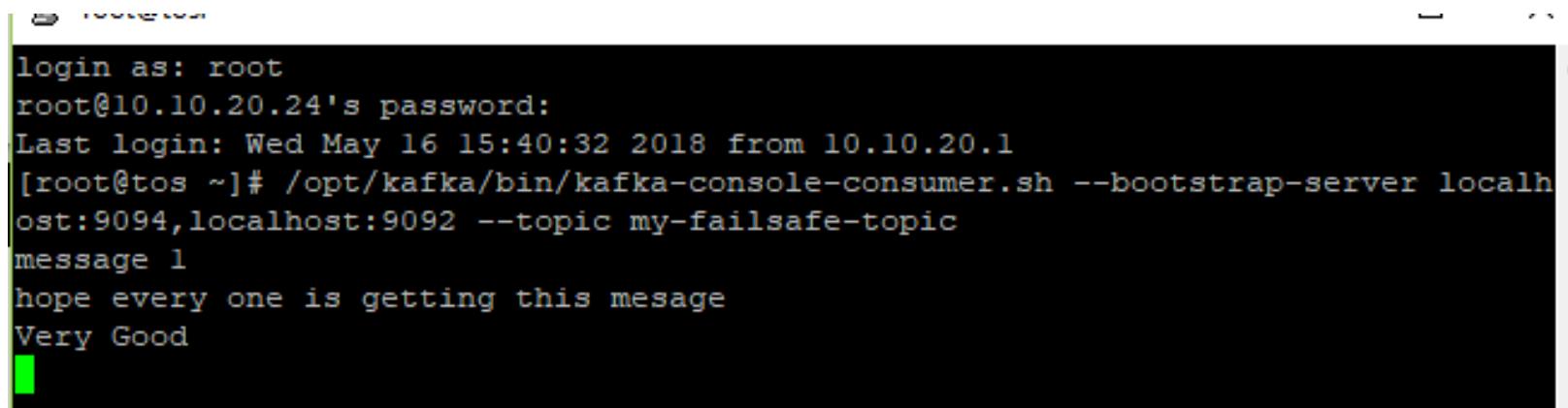
The image displays two separate terminal windows, each showing the output of a Kafka consumer console. Both terminals are running as root on a host with IP 10.10.20.24, connected to a local Kafka instance at ports 9094 and 9092, and reading from the topic 'my-failsafe-topic'. The terminals are identical in appearance, with a dark background and light-colored text. The top terminal has a blue border, and the bottom terminal has a green border.

```
root@tos:~  
login as: root  
root@10.10.20.24's password:  
Last login: Wed May 16 15:40:32 2018 from 10.10.20.1  
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic  
  
root@tos:~  
login as: root  
root@10.10.20.24's password:  
Last login: Wed May 16 16:15:15 2018 from 10.10.20.1  
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic
```

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 localhost:9092,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.

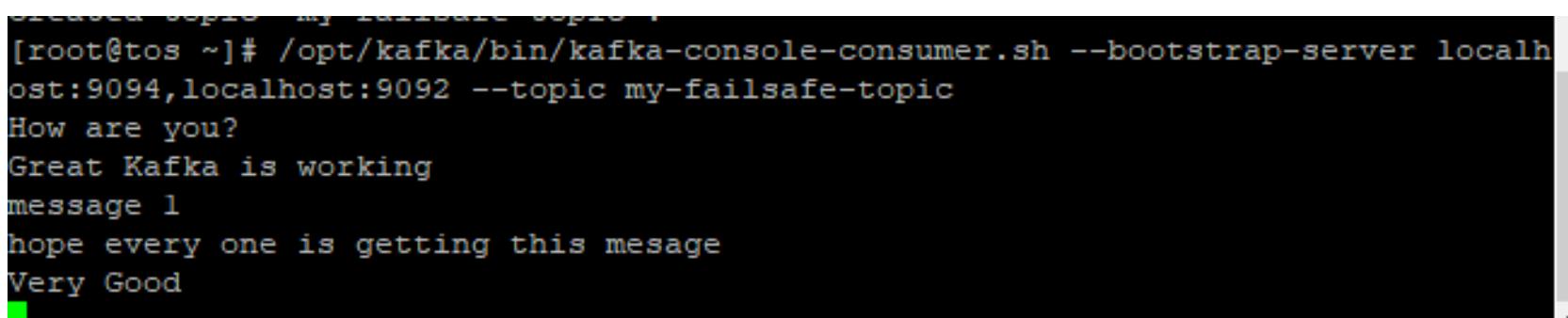


A terminal window showing the output of a Kafka consumer. The session starts with root login information, followed by the command to run the consumer. The consumer then prints several messages it has received from the topic 'my-failsafe-topic'.

```
login as: root
root@10.10.20.24's password:
Last login: Wed May 16 15:40:32 2018 from 10.10.20.1
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic
message 1
hope every one is getting this mesage
Very Good
```

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



A terminal window showing the output of a Kafka consumer. The session starts with root login information, followed by the command to run the consumer. The consumer then prints several messages it has received from the topic 'my-failsafe-topic'.

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic
How are you?
Great Kafka is working
message 1
hope every one is getting this mesage
Very Good
```

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  
localhost:9094,localhost: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 tos.master.com:9092,tos.master.co  
m:9093 --topic my-failsafe-topic
```

Now send eight messages from the Kafka producer console.

Producer Console

```
[root@tos ~]#  
[root@tos ~]#  
[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  
>
```

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 localhost: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 localhost: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 localhost: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 localhost: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:9092,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 localhost:9092
```

```
[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: 0,1,2
      Topic: my-failsafe-topic      Partition: 1      Leader: 0      Replicas: 0,1,2 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 2      Leader: 1      Replicas: 1,2,0 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 3      Leader: 2      Replicas: 2,1,0 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 4      Leader: 0      Replicas: 0,2,1 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 5      Leader: 1      Replicas: 1,0,2 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 6      Leader: 2      Replicas: 2,0,1 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 7      Leader: 0      Replicas: 0,1,2 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 8      Leader: 1      Replicas: 1,2,0 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 9      Leader: 2      Replicas: 2,1,0 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 10     Leader: 0      Replicas: 0,2,1 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 11     Leader: 1      Replicas: 1,0,2 Isr: 0,1,2
      Topic: my-failsafe-topic      Partition: 12     Leader: 2      Replicas: 2,0,1 Isr: 0,1,2
[root@tos ~]#
```

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 socket server (kafka.network.SocketServer)
[2018-05-17 17:38:07,267] INFO [SocketServer brokerId=0] Shutdown completed (kafka.network.SocketServer)
[2018-05-17 17:38:07,286] INFO [KafkaServer id=0] shut down completed (kafka.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. Administration – 60 Minutes

The following activities will be performed in this lab:

- Create a replicated topic and understand its attributes.
- Increases Partition.
- Alter configuration of topic.

You need to start the first broker using startABroker.sh

```
#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]#
```

List and describe Topics

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.

Create topic with default settings, if not done earlier

```
#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic topic1 --partitions 2 --replication-factor 1
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic topic1 --partitions 2 --replication-factor 1
Created topic "topic1".
[root@tos scripts]#
```

List only single topic named "test" (prints only topic name), if you have not created it.

Execute the following script to create it.

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

Determine whether a particular topic exists?

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

List all topics (prints only topic names)

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

```
[root@tos scripts]# jps
12960 Jps
12314 Kafka
12043 QuorumPeerMain
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper tos.master.com:2181 --topic test
test
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
CustomerCountry
__consumer_offsets
henry-topic
my-failsafe-topic
my-kafka-topic
my-kafka-topic1
test
test-topic
```

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

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

Who is the leader of the partition 0 of topic test?. Verify that.

Describe all topics (prints details about the topics)

```
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092
[root@tos scripts]#
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test
Topic:test      PartitionCount:1      ReplicationFactor:1      Configs:
          Topic: test      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181
Topic:CustomerCountry      PartitionCount:1      ReplicationFactor:1      Configs:
          Topic: CustomerCountry      Partition: 0      Leader: 1      Replicas: 1      Isr: 1
Topic:__consumer_offsets      PartitionCount:50      ReplicationFactor:1      Configs:segment.bytes=104857600,cleanup.policy=compact,compression.type=producer
          Topic: __consumer_offsets      Partition: 0      Leader: 1      Replicas: 1      Isr: 1
          Topic: __consumer_offsets      Partition: 1      Leader: 2      Replicas: 2      Isr: 2
```

List info for topics which have under replicated count. (You may not have records)

```
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --under-replicated-partitions
```

```
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --under-replicated-partitions
```

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --under-replicated-partitions
Topic: my-failsafe-topic    Partition: 0    Leader: 0    Replicas: 0    Isr: 0
Topic: my-failsafe-topic    Partition: 1    Leader: 0    Replicas: 0,1,2 Isr: 0
Topic: my-failsafe-topic    Partition: 2    Leader: 0    Replicas: 1,2,0 Isr: 0
Topic: my-failsafe-topic    Partition: 3    Leader: 0    Replicas: 2,1,0 Isr: 0
Topic: my-failsafe-topic    Partition: 4    Leader: 0    Replicas: 0,2,1 Isr: 0
Topic: my-failsafe-topic    Partition: 5    Leader: 0    Replicas: 1,0,2 Isr: 0
Topic: my-failsafe-topic    Partition: 6    Leader: 0    Replicas: 2,0,1 Isr: 0
Topic: my-failsafe-topic    Partition: 7    Leader: 0    Replicas: 0,1,2 Isr: 0
Topic: my-failsafe-topic    Partition: 8    Leader: 0    Replicas: 1,2,0 Isr: 0
Topic: my-failsafe-topic    Partition: 9    Leader: 0    Replicas: 2,1,0 Isr: 0
Topic: my-failsafe-topic    Partition: 10   Leader: 0    Replicas: 0,2,1 Isr: 0
Topic: my-failsafe-topic    Partition: 11   Leader: 0    Replicas: 1,0,2 Isr: 0
Topic: my-failsafe-topic    Partition: 12   Leader: 0    Replicas: 2,0,1 Isr: 0
Topic: my-kafka-topic      Partition: 0    Leader: 0    Replicas: 0,1,2 Isr: 0
```

List info for topics whose leader for a partition is not available

```
/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --unavailable-partitions
```

```
/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --unavailable-partitions
```

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --unavailable-partitions
Topic: my-example-topic Partition: 0    Leader: none    Replicas: 2    Isr: 2
[root@kafka0 /]#
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --unavailable-partitions
Topic: CustomerCountry Partition: 0 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 0 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 1 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 3 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 4 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 6 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 7 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 9 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 10 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 12 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 13 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 15 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 16 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 18 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 19 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 21 Leader: 1 Replicas: 1 Isr: 1
Topic: __consumer_offsets Partition: 22 Leader: 2 Replicas: 2 Isr: 2
Topic: __consumer_offsets Partition: 24 Leader: 1 Replicas: 1 Isr: 1
```

Add Partition to Topic

What does the tool do?

In Kafka partitions act as the unit of parallelism: messages of a single topic are distributed to multiple partitions that can be stored and served on different servers. Upon creation of a topic, the number of partitions for this topic has to be specified. Later on more partitions may be needed for this topic when the volume of this topic increases. This tool helps to add more partitions for a specific topic and also allow manual replica assignment of the added partitions.

Increase number of partitions for topic.

How many partitions does topic1 has?

```
# /opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092 --topic topic1
```

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic topic1
Topic: topic1  TopicId: 2x9806j8TPea0_5hLV_ng PartitionCount: 2      ReplicationFactor: 1      Configs:
  Topic: topic1  Partition: 0    Leader: 0      Replicas: 0      Isr: 0
  Topic: topic1  Partition: 1    Leader: 0      Replicas: 0      Isr: 0
[root@kafka0 /]#
```

As per the above, its 2. Let us increase to 4.

```
# /opt/kafka/bin/kafka-topics.sh --alter --bootstrap-server kafka0:9092 --topic topic1 --partitions 4
```

Verify the result:

```
# /opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092 --topic topic1
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --alter --zookeeper localhost:2181 --topic topic1 --partitions 4
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering
of the messages will be affected
Adding partitions succeeded!
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic topic1
Topic:topic1    PartitionCount:4    ReplicationFactor:1    Configs:
          Topic: topic1    Partition: 0    Leader: 0        Replicas: 0    Isr: 0
          Topic: topic1    Partition: 1    Leader: 0        Replicas: 0    Isr: 0
          Topic: topic1    Partition: 2    Leader: 0        Replicas: 0    Isr: 0
          Topic: topic1    Partition: 3    Leader: 0        Replicas: 0    Isr: 0
[root@tos scripts]#
```

As observe above, its having 4 partitions now.

Delete Topic.

When topic deletion is enabled in the broker (delete.topic.enable), topics can be deleted using the Kafka Topics tool.

Delete topic, **topic1**

```
#/opt/kafka/bin/kafka-topics.sh --delete --bootstrap-server kafka0:9092 --topic topic1
#/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic topic1
Topic topic1 is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
CustomerCountry
__consumer_offsets
henry-topic
my-failsafe-topic
my-kafka-topic
my-kafka-topic1
test
test-topic
[root@tos scripts]#
```

Change topic configuration.

Kafka Confings tool can be used to modify topic configuration:

- Add new config options
- Change existing config options
- Remove config options

Create a topic and verify it before going ahead

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

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic topic1 --partitions 2 --replication-factor 1
Created topic "topic1".
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic topic1
Topic:topic1    PartitionCount:2      ReplicationFactor:1      Configs:
          Topic: topic1    Partition: 0    Leader: 0        Replicas: 0    Isr: 0
          Topic: topic1    Partition: 1    Leader: 0        Replicas: 0    Isr: 0
[root@tos scripts]#
```

Add new option or change existing option

```
#/opt/kafka/bin/kafka-configs.sh --alter --bootstrap-server localhost:9092 --entity-name topic1 --entity-type topics --add-config cleanup.policy=compact
```

```
((base) Henrys-MacBook-Air:scripts henrypotsangbam$ sudo /opt/kafka/bin/kafka-configs.sh --alter --bootstrap-server localhost:9092 --entity-name topic1 --entity-type topics --add-config cleanup.policy=compact
Completed updating config for topic topic1.
```

Remove existing option

```
/opt/kafka/bin/kafka-configs.sh --alter --bootstrap-server localhost:9092 --entity-name topic1 --entity-type topics --delete-config cleanup.policy
```

```
(base) Henrys-MacBook-Air:scripts henrypotsangbam$ /opt/kafka/bin/kafka-configs.sh --alter --bootstrap-server localhost:9092 --entity-name topic1 --entity-type topics --delete-config cleanup.policy
Completed updating config for topic topic1.
(base) Henrys-MacBook-Air:scripts henrypotsangbam$
```

Determine the consumer group details.

Start the single broker in case its stop else skip this step.

```
#sh startABroker.sh
```

Create a producer and consume from two separate consoles.

Let us create a topic, benchmark-consumer for this lab.

```
#/opt/kafka/bin/kafka-topics.sh --create \
--bootstrap-server kafka:9092 \
--replication-factor 1 \
--partitions 1 \
--topic benchmark-consumer
```

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

```
#/opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark-consumer \
--num-records 45 \
--record-size 100 \
```

```
--throughput 15 \
--producer-props \
acks=1 \
bootstrap.servers=localhost:9092 \
buffer.memory=674 \
compression.type=none \
batch.size=8
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark
-consumer \
> --num-records 45 \
> --record-size 100 \
> --throughput 15 \
> --producer-props \
> acks=1 \
> bootstrap.servers=tos.master.com:9092 \
> buffer.memory=674 \
> compression.type=none \
> batch.size=8
45 records sent, 14.985015 records/sec (0.00 MB/sec), 24.04 ms avg latency, 343.
00 ms max latency, 6 ms 50th, 140 ms 95th, 343 ms 99th, 343 ms 99.9th.
[root@tos scripts]#
```

Open 2 console and start consuming from the above topic, ensure that consumer groups are different for each consumer terminal.

Create the following two properties.

```
#cd /opt/scripts  
#vi consumer.properties  
group.id=tos-1
```

```
#vi consumer1.properties  
group.id=tos-2
```

Terminal 1:

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic  
benchmark-consumer --from-beginning --consumer.config consumer.properties
```

Terminal 2:

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic  
benchmark-consumer --from-beginning --consumer.config consumer1.properties
```

For example, list new consumer groups:

```
# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list
```

For any group listed, you can get more details by changing the --list parameter to --describe and adding the --group parameter. This will list all the topics that the group is consuming, as well as the offsets for each topic partition.

Get the group of consumer as listed above.

```
#/opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe  
--group console-consumer-90417
```

```
[root@tos ~]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.ma  
ster.com:9092 --list  
Note: This will not show information about old Zookeeper-based consumers.  
console-consumer-90417  
console-consumer-45224  
[root@tos ~]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.ma  
ster.com:9092 --describe --group console-consumer-90417  
Note: This will not show information about old Zookeeper-based consumers.  
  
TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG          CO  
NSUMER-ID  
benchmark-consumer 0      94            94            0            co  
nsumer-1-cal34c4f-ad6a-4380-a62d-07c314ffldaf /192.168.139.129 consumer-1  
[root@tos ~]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --desc  
ribe --group console-consumer-90417  
Note: This will not show information about old Zookeeper-based consumers.  
  
TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG          CONSUMER-ID  
HOST           CLIENT-ID  
benchmark-consumer 0      94            94            0            consumer-1-cal34c4f-ad6  
a-4380-a62d-07c314ffldaf /192.168.139.129 consumer-1  
[root@tos ~]#
```

what is the current offset : 94

Log end offset : 94. Both are same means there is no lag

Lab End Here

9. Movement of Partition from a Broker to another Broker – 60 Minutes. (ND)

In this lab, we will add a new Kafka Broker to a cluster and move existing topics's partitions to the new Broker with all the messages intact.

To move partitions from old Brokers to newly added Brokers, first add Broker one by one to the existing cluster. Once the Brokers are added, run the partition reassignment to move the topic partitions from old to new Brokers, and remove the old brokers if needed.

At this point we have 3 brokers cluster and all the brokers are started as shown below: else start with the following script

```
#sh start3Brokers.sh
```

```
[root@tos scripts]# jps  
2224 Kafka  
2225 Kafka  
2226 Kafka  
3221 Jps  
2218 QuorumPeerMain  
[root@tos scripts]# █
```

Create the following Topic if not done before:

Or complete the following lab before proceeding ahead:

Writing a Kafka Producer in Java (Development Lab)

The following are the steps to add a Broker i.e 4th and perform partition reassignment of a single topic (the same can be followed for all other topics):

1. Create a log directory
2. Configure server properties
3. Create script to start server 4.

Let us list the topics created in the current cluster.

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

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper tos.master.com:2181
CustomerCountry
__consumer_offsets
henry-topic
my-failsafe-topic
my-kafka-topic
my-kafka-topic1
test
test-topic
[root@tos scripts]#
```

Out of the many topics listed above, we will only move the “my-kafka-topic”.

Determine the no Of brokers:

Zookeeper manages the leader election and other coordination things for a Kafka cluster. So Zookeeper has a list of all the Kafka brokers in the cluster.

```
# cd /opt/kafka/bin  
#sh zookeeper-shell.sh localhost:2181 ls /brokers/ids
```

```
[root@kafka0 bin]# sh zookeeper-shell.sh localhost:2181 ls /brokers/ids  
Connecting to localhost:2181  
  
WATCHER::  
  
WatchedEvent state:SyncConnected type:None path:null  
[0, 1, 2]  
[root@kafka0 bin]# pwd  
/opt/kafka/bin  
[root@kafka0 bin]#
```

There are 3 brokers as listed above.

You can determine the current partition leader for the mention topic so that we can compare it after rebalance.

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

Note: If the replication is less than 3 increase it to 3 by executing the command mention in the Administration Lab.

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

The following are the configuration of the 4th Kafka broker :

1. Create a log directory

```
#mkdir /opt/data/kafka-logs/kafka-3
```

2. Configure server properties

```
# cp /opt/kafka/config/server.properties /opt/kafka-config/config/server-3.properties
```

Update the /opt/kafka-config/config/server-3.properties with the following properties, ensure that rest of the attributes are remain default.

```
broker.id=3
listeners=PLAINTEXT://tos.master.com:9095
advertised.listeners=PLAINTEXT://tos.master.com:9095
log.dirs=/opt/data/kafka-logs/kafka-3
```

Create script to start server 4.

```
#vi /opt/kafka-config/start-4th-server.sh
```

Update with the following entries in the above file.

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

```
## Run Kafka
```

```
/opt/kafka/bin/kafka-server-start.sh "/opt/kafka-config/config/server-3.properties"
```

Start the broker.

```
#/opt/kafka-config/start-4th-server.sh
```

```
[2018-06-16 21:21:32,512] INFO [ExpirationReaper-3-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2018-06-16 21:21:32,512] INFO [ExpirationReaper-3-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2018-06-16 21:21:32,545] INFO [ExpirationReaper-3-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2018-06-16 21:21:32,592] INFO [GroupCoordinator 3]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2018-06-16 21:21:32,601] INFO [GroupCoordinator 3]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2018-06-16 21:21:32,636] INFO [GroupMetadataManager brokerId=3] Removed 0 expired offsets in 29 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
[2018-06-16 21:21:32,653] INFO [ProducerId Manager 3]: Acquired new producerId block (brokerId:3,blockStartProducerId:37000,blockEndProducerId:3799
9) by writing to Zk with path version 38 (kafka.coordinator.transaction.ProducerIdManager)
[2018-06-16 21:21:32,710] INFO [TransactionCoordinator id=3] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2018-06-16 21:21:32,720] INFO [TransactionCoordinator id=3] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2018-06-16 21:21:33,207] INFO [Transaction Marker Channel Manager 3]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2018-06-16 21:21:33,381] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessTh
read)
[2018-06-16 21:21:33,502] INFO Kafka version : 1.1.0 (org.apache.kafka.common.utils.AppInfoParser)
[2018-06-16 21:21:33,502] INFO Kafka commitId : fdcf75ea326b8e07 (org.apache.kafka.common.utils.AppInfoParser)
[2018-06-16 21:21:33,506] INFO [KafkaServer id=3] started (kafka.server.KafkaServer)
```

Note

Run this tool one topic at a time. Ensure all the Brokers are up and running for all the partitions so that the topics are in healthy state. Else, the partition reassignment might not be successful.

The following is the current topic description: Open a new console and execute the following.

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

```
Last login: Sat Jun 16 20:32:59 2018 from 10.10.20.1
[root@tos ~]# /opt/kafka/bin/kafka-topics.sh --describe --topic my-kafka-topic --zookeeper tos.master.com:2181
Topic:my-kafka-topic    PartitionCount:13      ReplicationFactor:3      Configs:
  Topic: my-kafka-topic    Partition: 0      Leader: 1      Replicas: 1,2,0 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 1      Leader: 2      Replicas: 2,0,1 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 2      Leader: 0      Replicas: 0,1,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 3      Leader: 1      Replicas: 1,0,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 4      Leader: 2      Replicas: 2,1,0 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 5      Leader: 0      Replicas: 0,2,1 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 6      Leader: 1      Replicas: 1,2,0 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 7      Leader: 2      Replicas: 2,0,1 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 8      Leader: 0      Replicas: 0,1,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 9      Leader: 1      Replicas: 1,0,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 10     Leader: 2      Replicas: 2,1,0 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 11     Leader: 0      Replicas: 0,2,1 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 12     Leader: 1      Replicas: 1,2,0 Isr: 2,0,1
[root@tos ~]#
```

1. 4th Server i.e broker id 3 is not part of the ISR automatically.

In the above description, Leader 1 is having more partition, so we will move partition 12 to the new server.

2. The following is the data log details on 1 broker (for reference only):

```
[root@tos ~]# ls -l /opt/data/kafka-logs/kafka-1/
```

```
(base) Henrys-MacBook-Air:~ henrypotsangbam$ ls -l /opt/data/kafka-logs/kafka-1/
total 32
-rw-r--r-- 1 root wheel 10485760 Aug 28 21:00 00000000000000000000.index
-rw-r--r-- 1 root wheel 0 Aug 28 21:00 00000000000000000000.log
-rw-r--r-- 1 root wheel 10485756 Aug 28 21:00 00000000000000000000.timeindex
-rw-r--r-- 1 root wheel 0 Aug 28 15:03 cleaner-offset-checkpoint
-rw-r--r-- 1 root wheel 0 Aug 28 21:00 leader-epoch-checkpoint
[-rw-r--r-- 1 root wheel 4 Aug 29 13:05 log-start-offset-checkpoint
[-rw-r--r-- 1 root wheel 88 Aug 29 11:58 meta.properties
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-0
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-1
drwxr-xr-x 6 root wheel 192 Aug 29 11:58 my-failsafe-topic-10
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-11
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-12
drwxr-xr-x 6 root wheel 192 Aug 29 11:58 my-failsafe-topic-2
[drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-3
drwxr-xr-x 6 root wheel 192 Aug 29 11:58 my-failsafe-topic-4
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-5
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-6
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-7
drwxr-xr-x 6 root wheel 192 Aug 29 11:58 my-failsafe-topic-8
drwxr-xr-x 7 root wheel 224 Aug 29 11:58 my-failsafe-topic-9
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-0
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-1
drwxr-xr-x 6 root wheel 192 Aug 29 13:04 my-kafka-topic-10
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-11
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-12
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-2
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-3
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-4
drwxr-xr-x 6 root wheel 192 Aug 29 13:04 my-kafka-topic-5
[drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-6
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-7
drwxr-xr-x 6 root wheel 192 Aug 29 13:04 my-kafka-topic-8
drwxr-xr-x 6 root wheel 192 Aug 29 13:02 my-kafka-topic-9
-rw-r--r-- 1 root wheel 544 Aug 29 13:05 recovery-point-offset-checkpoint
-rw-r--r-- 1 root wheel 544 Aug 29 13:06 replication-offset-checkpoint
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

```
#ls -l /opt/data/kafka-logs/kafka-1/my-kafka-topic-12
```

```
[root@tos ~]# ls -l /opt/data/kafka-logs/kafka-1/my-kafka-topic-12
total 16
-rw-r--r--. 1 root root 10485760 Jun 16 20:33 00000000000000000000000000000000.index
-rw-r--r--. 1 root root      202 Jun 16 17:34 00000000000000000000000000000000.log
-rw-r--r--. 1 root root 10485756 Jun 16 20:33 00000000000000000000000000000000.timeindex
-rw-r--r--. 1 root root       10 Jun 16 20:33 00000000000000000000000000000002.snapshot
-rw-r--r--. 1 root root        8 Jun 16 17:11 leader-epoch-checkpoint
[root@tos ~]#
```

We will move the partition 12 (Or choose any partition available in Node 0 to move) to 1,2 and 3 server.

Create a file named custom-reassignment.json with the following content:

```
{"version":1,"partitions":[{"topic":"my-kafka-topic","partition":12,"replicas":[1,2,3]}]}
```

Run the following command which will give you proposed partition reassignment:

Note: Ensure new broker ids are provided in the broker list for the partitions to be reassigned to the new brokers from the old brokers. (In the below example, the ID of the new Broker is list1011)

```
# /opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 --reassignment-json-file custom-reassignment.json --execute
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-reassign-partitions.sh --zookeeper tos.master.com:2181 --reassignment-json-file custom-reassignment.json --execute
Current partition replica assignment

{"version":1,"partitions":[{"topic":"my-kafka-topic","partition":3,"replicas":[1,0,2],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":6,"replicas":[1,2,0],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":12,"replicas":[1,2,0],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":4,"replicas":[2,1,0],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":8,"replicas":[0,1,2],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":1,"replicas":[2,0,1],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":10,"replicas":[2,1,0],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":9,"replicas":[1,0,2],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":0,"replicas":[1,2,0],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":7,"replicas":[2,0,1],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":5,"replicas":[0,2,1],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":2,"replicas":[0,1,2],"log_dirs":["any","any","any"]}, {"topic":"my-kafka-topic","partition":11,"replicas":[0,2,1],"log_dirs":["any","any","any"]}]

Save this to use as the --reassignment-json-file option during rollback
Successfully started reassignment of partitions.
[root@tos scripts]#
```

The `--verify` option can be used with the tool to check the status of the partition reassignment. Note that the same `expand-cluster-reassignment.json` (used with the `--execute` option) should be used with the `--verify` option:

```
#/opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 --reassignment-json-file custom-reassignment.json --verify
```

or

```
/opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server kafka:9092 --reassignment-json-file custom-reassignment.json --verify
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-reassign-partitions.sh --zookeeper tos.master.com:2181 --reassignment-json-file custom-reassignment.json --verify
Status of partition reassignment:
Reassignment of partition my-kafka-topic-12 completed successfully
[root@tos scripts]#
```

Run describe to check the partition reassignment:

```
#/opt/kafka/bin/kafka-topics.sh --describe --topic my-kafka-topic --bootstrap-server localhost:9092 r
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --describe --topic my-kafka-topic --zookeeper tos.master.com:2181
Topic:my-kafka-topic    PartitionCount:13    ReplicationFactor:3    Configs:
    Topic: my-kafka-topic    Partition: 0    Leader: 1    Replicas: 1,2,0 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 1    Leader: 2    Replicas: 2,0,1 Isr: 1,0,2
    Topic: my-kafka-topic    Partition: 2    Leader: 0    Replicas: 0,1,2 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 3    Leader: 1    Replicas: 1,0,2 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 4    Leader: 2    Replicas: 2,1,0 Isr: 1,0,2
    Topic: my-kafka-topic    Partition: 5    Leader: 0    Replicas: 0,2,1 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 6    Leader: 1    Replicas: 1,2,0 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 7    Leader: 2    Replicas: 2,0,1 Isr: 1,0,2
    Topic: my-kafka-topic    Partition: 8    Leader: 0    Replicas: 0,1,2 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 9    Leader: 1    Replicas: 1,0,2 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 10   Leader: 2    Replicas: 2,1,0 Isr: 1,0,2
    Topic: my-kafka-topic    Partition: 11   Leader: 0    Replicas: 0,2,1 Isr: 2,0,1
    Topic: my-kafka-topic    Partition: 12   Leader: 1    Replicas: 1,2,3 Isr: 2,1,3
[root@tos scripts]#
```

Previous to shifting:

```
Last login: Sat Jun 16 20:32:59 2018 from 10.10.20.1
[root@tos ~]# /opt/kafka/bin/kafka-topics.sh --describe --topic my-kafka-topic --zookeeper tos.master.com:2181
Topic:my-kafka-topic    PartitionCount:13    ReplicationFactor:3    Configs:
  Topic: my-kafka-topic    Partition: 0    Leader: 1        Replicas: 1,2,0 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 1    Leader: 2        Replicas: 2,0,1 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 2    Leader: 0        Replicas: 0,1,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 3    Leader: 1        Replicas: 1,0,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 4    Leader: 2        Replicas: 2,1,0 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 5    Leader: 0        Replicas: 0,2,1 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 6    Leader: 1        Replicas: 1,2,0 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 7    Leader: 2        Replicas: 2,0,1 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 8    Leader: 0        Replicas: 0,1,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 9    Leader: 1        Replicas: 1,0,2 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 10   Leader: 2        Replicas: 2,1,0 Isr: 1,0,2
  Topic: my-kafka-topic    Partition: 11   Leader: 0        Replicas: 0,2,1 Isr: 2,0,1
  Topic: my-kafka-topic    Partition: 12   Leader: 1        Replicas: 1,2,0 Isr: 2,0,1
[root@tos ~]#
```

Check the data logs on new brokers to verify that the partitions have moved:

```
# ls -l /opt/data/kafka-logs/kafka-3/my-kafka-topic-12
```

```
[root@tos scripts]# ls -l /opt/data/kafka-logs/kafka-3/my-kafka-topic-12
total 8
-rw-r--r--. 1 root root 10485760 Jun 16 21:53 00000000000000000000.index
-rw-r--r--. 1 root root      202 Jun 16 21:53 00000000000000000000.log
-rw-r--r--. 1 root root 10485756 Jun 16 21:53 00000000000000000000.timeindex
-rw-r--r--. 1 root root       8 Jun 16 21:53 leader-epoch-checkpoint
[root@tos scripts]#
```

You can verify the consumer that all messages are intact.

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

```
[root@tos scripts]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server tos.master.com:9094,tos.master.com:9092 --topic my-kafka-topic --from-beginning
Hello Kafka 1529149269561
Hello Kafka 1529149269562
Hello Kafka 1529150682676
France
France
Hello Kafka 1529150682679
Hello Kafka 1529150682680
Hello Kafka 1529148792458
Hello Kafka 1529150682678
How are you
Hello Kafka 1529149269564
Hope it works
Hello Kafka 1529148792459
Hello Kafka 1529149269560
Hello Kafka 1529150682677
hello
Hello Kafka 1529148792461
how
Hello Kafka 1529148792457
Hello Kafka 1529148792460
Hello Kafka 1529149269563
```

Lab Ends Here

10. Securing Kafka SSL – 90 Minutes

Enable Encryption and Authentication using SSL.

In this lab we will configure SSL between client and the broker, then ACL will be enable accordingly.

Take a back up of the server.properties before making any changes. [cp server.properties server.properties_org]

Apache Kafka allows clients to connect over SSL. By default, SSL is disabled but can be turned on as needed.

You need to shut down all the brokers before going ahead with this lab.

[hints: jps and kill -9 all kafka/zookeeper processes]

```
# yum install openssl
```

Before going ahead with the SSL configuration let us verify that TLS is configured or not.

```
#openssl s_client -debug -connect localhost:9092 -tls1
```

```
[root@hp ~]# openssl s_client -debug -connect localhost:9092 -tls1
socket: Connection refused
connect:errno=111
[root@hp ~]#
```

```
No client certificate CA names sent
---
SSL handshake has read 0 bytes and written 0 bytes
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol : TLSv1
    Cipher   : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    Key-Ag  : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1598776918
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
---
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

If you get the above output, then it's not configured.

Generate SSL key and certificate for each Kafka broker

The first step of deploying one or more brokers with the SSL support is to generate the key and the certificate for each machine in the cluster. You can use Java's keytool utility

to accomplish this task. We will generate the key into a temporary keystore initially so that we can export and sign it later with CA. (Create the folder if its not there)

Change the hp.com parameter with that of your hostname

```
# cd /opt/scripts/cer
```

```
# keytool -keystore server.keystore.jks -alias hp.com -validity 365 -genkey -keyalg RSA
```

Enter the following parameters:

Password: kafka213

Enter Other details appropriately.

Accept the final password as same as above.

```
[root@hp opt]#  
[root@hp opt]# keytool -keystore server.keystore.jks -alias hp.com -validity 365  
-genkey -keyalg RSA  
Enter keystore password:  
Re-enter new password:  
What is your first and last name?  
[Unknown]: Apache  
What is the name of your organizational unit?  
[Unknown]: tos.com  
What is the name of your organization?  
[Unknown]: tos  
What is the name of your City or Locality?  
[Unknown]: Mumbai  
What is the name of your State or Province?  
[Unknown]: Maharashtra  
What is the two-letter country code for this unit?  
[Unknown]: IN  
Is CN=Apache, OU=tos.com, O=tos, L=Mumbai, ST=Maharashtra, C=IN correct?  
[no]: yes  
  
Enter key password for <hp.com>  
      (RETURN if same as keystore password):  
Re-enter new password:  
[root@hp opt]# pwd  
/opt  
[root@hp opt]# ls  
apache-drill-1.12.0  kafka  scripts  server.keystore.jks  zeppelin  
[root@hp opt]# █
```

You need to specify two parameters in the above command:

1. **keystore**: the keystore file that stores the certificate. The keystore file contains the private key of the certificate; therefore, it needs to be kept safely.
2. **validity**: the valid time of the certificate in days.

Input all the necessary required attribute. Accept the password as same as above.

The following command can be run afterwards to verify the contents of the generated certificate:

```
#keytool -list -v -keystore server.keystore.jks
```

```
[root@hp opt]# keytool -list -v -keystore server.keystore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: hp.com
Creation date: 22 May, 2018
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Apache, OU=tos.com, O=tos, L=Mumbai, ST=Maharashtra, C=IN
Issuer: CN=Apache, OU=tos.com, O=tos, L=Mumbai, ST=Maharashtra, C=IN
Serial number: 666383b2
Valid from: Tue May 22 15:58:12 IST 2018 until: Wed May 22 15:58:12 IST 2019
Certificate fingerprints:
    MD5: A2:88:C5:3E:84:DA:00:89:0A:18:9D:B1:73:3B:D0:77
        SHA1: 12:0B:5E:FC:33:64:AF:71:44:D5:FC:22:98:61:AB:F5:69:B7:A8:8F
        SHA256: 5F:BD:EE:25:BC:10:CE:B1:7B:DA:84:3B:64:FF:15:29:1B:F2:9F:A5:2E:BE:2F:2C:
2A:E7:8D:92:F2:1B:76:79
    Signature algorithm name: SHA256withRSA
    Version: 3
```

Configure your own CA

After the first step, each machine in the cluster has a public-private key pair, and a certificate to identify the machine. The certificate, however, is unsigned, which means that an attacker can create such a certificate to pretend to be any machine. Therefore, it is important to prevent forged certificates by signing them for each machine in the cluster. A certificate authority (CA) is responsible for signing certificates. CA works like a government that issues passports—the government stamps (signs) each passport so that the passport becomes difficult to forge. Other governments verify the stamps to ensure the passport is authentic. Similarly, the CA signs the certificates, and the cryptography guarantees that a signed certificate is computationally difficult to forge. Thus, as long as the CA is a genuine and trusted authority, the clients have high assurance that they are connecting to the authentic machines.

Use Password: EnjoyKafka

If openssl is not found, install using - yum install openssl

```
#openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

The generated CA is simply a public-private key pair and certificate, and it is intended to sign other certificates.

```
[root@hp opt]# openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'ca-key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
Verify failure
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:IN
State or Province Name (full name) []:Maharashtra
Locality Name (eg, city) [Default City]:Goregaon
Organization Name (eg, company) [Default Company Ltd]:tos
Organizational Unit Name (eg, section) []:tos
Common Name (eg, your name or your server's hostname) []:tos.com
Email Address []:life.skolur@gmail.com
[root@hp opt]#
```

Enter the necessary details.

```
[root@hp opt]# ls
apache-drill-1.12.0  ca-cert  ca-key  kafka  scripts  server.keystore.jks  zeppelin
[root@hp opt]# pwd
/opt
[root@hp opt]#
```

The next step is to add the generated CA to the **clients' truststore** so that the clients can trust this CA:

Enter. (Password – kafka213) whenever the system ask for :

```
# keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
```

```
[root@hp opt]# keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert.p12
Enter keystore password:
Keystore password is too short - must be at least 6 characters
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maharashtra, C=IN
Issuer: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maharashtra, C=IN
Serial number: da8cbd2ee73744c2
Valid from: Tue May 22 16:03:11 IST 2018 until: Wed May 22 16:03:11 IST 2019
Certificate fingerprints:
        MD5: 7B:62:8C:92:1A:81:B9:B5:44:1B:10:63:98:32:AA:DD
        SHA1: 18:F7:AB:08:2F:84:6B:0A:91:5D:02:66:9B:84:6A:CA:E9:6B:69:8C
        SHA256: F9:75:E2:BB:44:6C:D5:32:FC:91:10:5D:D9:80:20:9B:40:9F:6F:A8:0B:86:25:5A:BC:19:4C:F5:04:CC:7B:2E
        Signature algorithm name: SHA256withRSA
        Version: 3

Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: CD 8A 8F 57 41 3A FB C8    CF E1 36 D8 F8 F5 27 F2  ...WA:....6...
0010: DF 92 B8 54                ...T
]
]
```

Trust this certificate? [no]: yes

Enter. (Password – kafka213)

Note: If you configure the Kafka brokers to require client authentication by setting `ssl.client.auth` to be "requested" or "required" on the [Kafka brokers config](#) then you must provide a truststore for the Kafka brokers as well and it should have all the CA certificates that clients' keys were signed by.

```
# keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
```

```
[root@hp opt]# ls
apache-drill-1.12.0  ca-key          kafka      server.keystore.jks
ca-cert              client.truststore.jks  scripts   zeppelin
[root@hp opt]# keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maha
rashtra, C=IN
Issuer: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maha
rashtra, C=IN
Serial number: da8cbd2ee73744c2
Valid from: Tue May 22 16:03:11 IST 2018 until: Wed May 22 16:03:11 IST 2019
Certificate fingerprints:
    MD5: 7B:62:8C:92:1A:81:B9:B5:44:1B:10:63:98:32:AA:DD
        SHA1: 18:F7:AB:08:2F:84:6B:0A:91:5D:02:66:9B:84:6A:CA:E9:6B:69:8C
        SHA256: F9:75:E2:BB:44:6C:D5:32:FC:91:10:5D:D9:80:20:9B:40:9F:6F:A8:0B:86:25:5A:
BC:19:4C:F5:04:CC:7B:2E
        Signature algorithm name: SHA256withRSA
        Version: 3

Extensions:
```

Password: kafka213

In contrast to the keystore in step 1 that stores each machine's own identity, the truststore of a client stores all the certificates that the client should trust. Importing a certificate into one's truststore also means trusting all certificates that are signed by that certificate. As the analogy above, trusting the government (CA) also means trusting all passports (certificates) that it has issued. This attribute is called the chain of trust, and it is particularly useful when deploying SSL on a large Kafka cluster. You

can sign all certificates in the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way all machines can authenticate all other machines.

```
[root@hp opt]# ls
apache-drill-1.12.0  ca-key          kafka    server.keystore.jks  zeppelin
ca-cert              client.truststore.jks  scripts  server.truststore.jks
[root@hp opt]#
```

Signing the certificate

The next step is to sign all certificates generated by step 1 with the CA generated in step 2. First, you need to export the certificate from the keystore: replace hp.com with that of your hostname which you enter before.

```
# keytool -keystore server.keystore.jks -alias hp.com -certreq -file cert-file
```

```
[root@hp opt]# keytool -keystore server.keystore.jks -alias hp.com -certreq -file cert-file
Enter keystore password:
[root@hp opt]#
```

Password: kafka213

Then sign it with the CA:

```
# openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreateserial
```

```
[root@hp opt]# openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed  
-days 365 -CAcreateserial  
Signature ok  
subject=/C=IN/ST=Maharashtra/L=Mumbai/O=tos/OU=tos.com/CN=Apache  
Getting CA Private Key  
Enter pass phrase for ca-key:  
[root@hp opt]#
```

Password: EnjoyKafka

Finally, you need to import both the certificate of the CA and the signed certificate into the keystore: Accept Yes for all Queries Y/N.

All password: kafka213

```
#keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
```

```
[root@hp opt]# keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert  
Enter keystore password:  
Owner: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maharashtra, C=IN  
Issuer: EMAILADDRESS=life.skolur@gmail.com, CN=tos.com, OU=tos, O=tos, L=Goregaon, ST=Maharashtra, C=IN  
Serial number: da8cbd2ee73744c2  
Valid from: Tue May 22 16:03:11 IST 2018 until: Wed May 22 16:03:11 IST 2019  
Certificate fingerprints:  
MD5: 7B:62:8C:92:1A:81:B9:B5:44:1B:10:63:98:32:AA:DD  
SHA1: 18:F7:AB:08:2F:84:6B:0A:91:5D:02:66:9B:84:6A:CA:E9:6B:69:8C  
SHA256: F9:75:E2:BB:44:6C:D5:32:FC:91:10:5D:D9:80:20:9B:40:9F:6F:A8:0B:86:25:5A:  
BC:19:4C:F5:04:CC:7B:2E  
Signature algorithm name: SHA256withRSA  
Version: 3
```

```
#keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

```
[root@hp opt]# keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
Enter keystore password:
Certificate was added to keystore
[root@hp opt]#
```

All password: kafka213

The definitions of the parameters are the following:

3. keystore: the location of the keystore
4. ca-cert: the certificate of the CA
5. ca-key: the private key of the CA
6. ca-password: the passphrase of the CA
7. cert-file: the exported, unsigned certificate of the server
8. cert-signed: the signed certificate of the server

Till now, we have configured the server key and certificate. Then we can configure the Broker to use SSL

Configuring Kafka Brokers

Kafka Brokers support listening for connections on multiple ports. We need to configure the following property in server.properties, which must have one or more comma-separated values:

Server properties is in /opt/kafka/config. [Ensure that you take a copy of the file before making any changes i.e cp server.properties server.properties_bak. We will be using the copy to roll back after this lab]

listeners

If SSL is not enabled for inter-broker communication, both PLAINTEXT and SSL ports will be necessary. Update the entry as shown below. Substitute with your hostname.

```
listeners= PLAINTEXT://tos.master.com:9092,SSL://tos.master.com:8092
```

```
##### 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://tos.master.com:9092,SSL://tos.master.com:8092

# Hostname and port the broker will advertise to producers and consumers. If not
set,
```

We have enable SSL on port no 8092. This broker can be connected using plain and SSL protocol.

Comment all the following highlighted entries:

```
# 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://your.host.name:9092
listeners= PLAINTEXT://kafka0:9092,SSL://kafka0:8092
#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
```

Following SSL configs are needed on the broker side. Add the following in the server.properties.

```
ssl.keystore.location=/opt/scripts/cer/server.keystore.jks
ssl.keystore.password=kafka213
ssl.key.password=kafka213
ssl.truststore.location=/opt/scripts/cer/server.truststore.jks
ssl.truststore.password=kafka213
```



```
root@tos:/opt/kafka/config
# EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://tos.master.com:9092,SSL://tos.master.com:8092

# 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=PLAINTEXT://tos.master.com:9092

# Maps listener names to security protocols, the default is for them to be the s
ame. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_
PLAINTEXT,SASL_SSL:SASL_SSL

#SSL Related Setting
ssl.keystore.location=/opt/scripts/server.keystore.jks
ssl.keystore.password=kafka213
ssl.key.password=kafka213
ssl.truststore.location=/opt/scripts/server.truststore.jks
ssl.truststore.password=kafka213
```

You need to ensure that path to the generated key are in /opt/scripts/cer/ folder.

Note: ssl.truststore.password is technically optional but highly recommended. If a password is not set access to the truststore is still available, but integrity checking is disabled.

Shutdown the broker if started earlier.

Start the broker now.

```
#cd /opt/scripts
#sh startABroker.sh
```

```
[root@tos scripts]#  
[root@tos scripts]# sh startABroker.sh  
ZooKeeper JMX enabled by default  
Using config: /opt/zookeeper/bin/../conf/zoo.cfg  
Starting zookeeper ... already running as process 2240.  
Started Successfully  
[root@tos scripts]#
```

Once you start the broker you should be able to see in the server.log with addresses:
PLAINTEXT -> EndPoint(192.168.64.1,9092,PLAINTEXT),SSL ->
EndPoint(192.168.64.1,9093,SSL)
/opt/kafka/log/server.properties.

```
leader.imbalance.per.broker.percentage = 10  
listener.security.protocol.map = PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAIN  
TEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL  
listeners = PLAINTEXT://tos.master.com:9092,SSL://tos.master.com:8092  
log.cleaner.backoff.ms = 15000  
log.cleaner.dedupe.buffer.size = 134217728  
log.cleaner.delete.retention.ms = 86400000
```

```
3372036854 // 580 / ms. (kafka.log.LogManager)  
[2018-06-24 23:56:03,801] INFO Awaiting socket connections on tos.master.com:909  
2. (kafka.network.Acceptor)  
[2018-06-24 23:56:03,882] INFO Awaiting socket connections on tos.master.com:809  
2. (kafka.network.Acceptor)  
[2018-06-24 23:56:04,167] INFO [SocketServer brokerId=0] Started 2 acceptor thre
```

You can even verify using the following command to find out the listening port no.

```
[root@hp logs]# lsof -i:8092
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
java    4688 root  215u  IPv6  50162      0t0  TCP hp.com:8092 (LISTEN)
[root@hp logs]#
```

To check quickly if the server keystore and truststore are setup properly you can run the following command

```
#openssl s_client -debug -connect localhost:8092
```

or

```
#openssl s_client -debug -connect kafka:8092
```

(Note: TLSv1 should be listed under ssl.enabled.protocols or –ts1 parameter require)
In the output of this command you should see server's certificate:

```
[root@hp logs]# openssl s_client -debug -connect hp.com:8092 -tls1
CONNECTED(00000003)
write to 0x1e06950 [0x1e4ed03] (155 bytes => 155 (0x9B))
0000 - 16 03 01 00 96 01 00 00-92 03 01 5b 03 f6 b9 39  .....[...9
0010 - f7 ba 86 d0 24 7d c2 85-8e 32 c1 12 66 38 60 63  ....$}....2..f8^c
0020 - e4 5a 83 0f b6 6c ad 04-69 54 d3 00 00 4c c0 14  .Z...l..iT...L..
0030 - c0 0a 00 39 00 38 00 88-00 87 c0 0f c0 05 00 35  ...9.8.....5
0040 - 00 84 c0 13 c0 09 00 33-00 32 00 9a 00 99 00 45  .....3.2.....E
0050 - 00 44 c0 0e c0 04 00 2f-00 96 00 41 c0 12 c0 08  .D...../....A....
0060 - 00 16 00 13 c0 0d c0 03-00 0a 00 07 c0 11 c0 07  .....
0070 - c0 0c c0 02 00 05 00 04-00 ff 01 00 00 1d 00 0b  .....
0080 - 00 04 03 00 01 02 00 0a-00 08 00 06 00 19 00 18  .....
0090 - 00 17 00 23 00 00 00 0f-00 01 01  ...#.....
read from 0x1e06950 [0x1e4a7b3] (5 bytes => 5 (0x5))
0000 - 16 03 01 05 5b  ....[
read from 0x1e06950 [0x1e4a7b8] (1371 bytes => 1371 (0x55B))
0000 - 02 00 00 4d 03 01 5b 03-f6 b9 b1 c1 a2 24 1a 90  ...M..[.....$..
0010 - 15 5e 2e d8 63 0f e7 7b-6b 1c ae 0c 54 4c fa d9  .^..c..{k...TL..
0020 - e4 ca 8e 6d d6 83 20 5b-03 f6 b9 0f 35 70 21 8c  ...m.. [.....5p!.
0030 - 39 fa b7 6f a2 ce b1 a1-de 87 9b 76 a2 80 85 22  9..o.....v...""
0040 - 6a 45 bb eb 5c a5 b4 c0-13 00 00 05 ff 01 00 01  jE..\.....
0050 - 00 0b 00 03 73 00 03 70-00 03 6d 30 82 03 69 30  ....s..p..m0..i0
0060 - 82 02 51 a0 03 02 01 02-02 04 66 63 83 b2 30 0d  ..Q.....fc..0.
0070 - 06 09 2a 86 48 86 f7 0d-01 01 0b 05 00 30 65 31  ..*.H.....0e1
0080 - 0b 30 09 06 03 55 04 06-13 02 49 4e 31 14 30 12  .0....U....IN1.0.
0090 - 06 03 55 04 08 13 0b 4d-61 68 61 72 61 73 74 68  ..U....Maharasth
00a0 - 72 61 31 0f 30 0d 06 03-55 04 07 13 06 4d 75 6d  ral.0...U....Mum
00b0 - 62 61 69 31 0c 30 0a 06-03 55 04 0a 13 03 74 6f  bail.0...U....to
00c0 - 73 31 10 30 0e 06 03 55-04 0b 13 07 74 6f 73 2e  sl.0...U....tos.
```

```
-----END CERTIFICATE-----
subject=/C=IN/ST=Maharashtra/L=Mumbai/O=tos/OU=tos.com/CN=Apache
issuer=/C=IN/ST=Maharashtra/L=Mumbai/O=tos/OU=tos.com/CN=Apache
---
No client certificate CA names sent
Server Temp Key: ECDH, secp521r1, 521 bits
---
SSL handshake has read 1435 bytes and written 357 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
Protocol : TLSv1
Cipher   : ECDHE-RSA-AES128-SHA
Session-ID: 5B03F6B90F3570218C39FAB76FA2CEB1A1DE879B76A28085226A45BBEB5CA5B4
Session-ID-ctx:
Master-Key: B3C76F24E11BD5F6D66B6605606AED4342923EF46908AD2938B82CB04CE6C75D9091C5729
9462745D40C921A3B67DFDC
Key-Ag  : None
Krb5 Principla: None
PSK identity: None
PSK identity hint: None
Start Time: 1526986425
Timeout   : 7200 (sec)
Verify return code: 18 (self signed certificate)
---
```

If the certificate does not show up or if there are any other error messages, then your keystore is not setup properly.

Or

```
--  
[(base) Henrys-MacBook-Air:logs henrypotsangbam$ sudo lsof -i:8092  
[Password:  
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME  
java 1465 root 137u IPv6 0xe3f7550241995151 0t0 TCP *:8092 (LISTEN)  
(base) Henrys-MacBook-Air:logs henrypotsangbam$ ]
```



Listening

That is about configuring SSL.

Let us now configure ACL. Before going ahead with the next section, you need to roll back the server.properties.

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

11. 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.

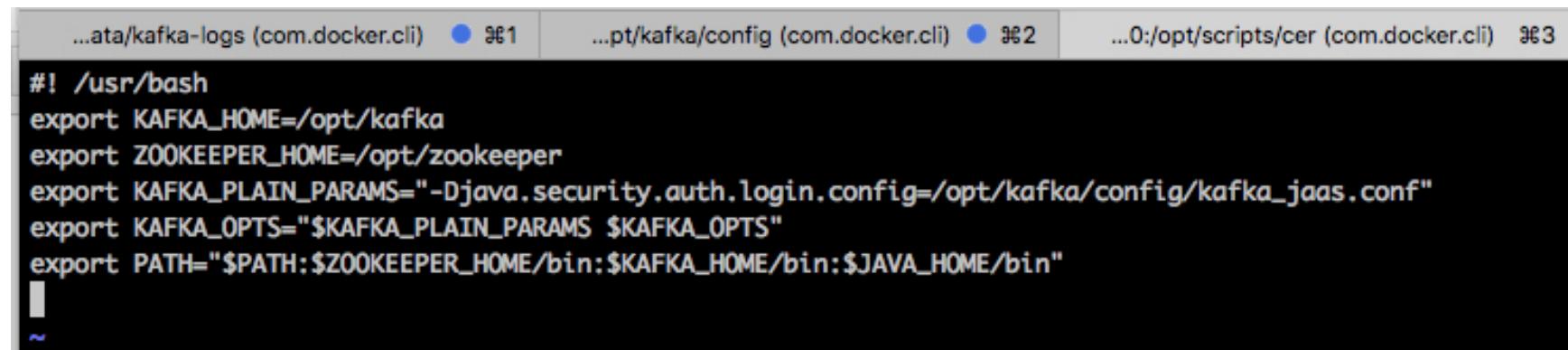
This file needs to be passed in as a JVM config option when running the broker, using -Djava.security.auth.login.config=[path_to_jaas_file].

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"
```



The screenshot shows a terminal window with three tabs at the top:

- ...ata/kafka-logs (com.docker.cli) ● 961
- ...pt/kafka/config (com.docker.cli) ● 962
- ...0:/opt/scripts/cer (com.docker.cli) 963

The main pane displays the script 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 any 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://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://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
# Map listener names to security protocols - the default is for them to be the s
```

Comment the following line.

```
# use Ssl for inter.broker only (not SSL)
# 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.

```
(kafka.server.KafkaConfig)
[2018-06-18 15:33:12,246] INFO KafkaConfig values:
    advertised.host.name = null
    advertised.listeners = SASL_PLAINTEXT://tos.master.com:9092
    advertised.port = null
    alter.config.policy.class.name = null
    alter.log.dirs.replication.quota.window.num = 11
    alter.log.dirs.replication.quota.window.size.seconds = 1
    authorizer.class.name =
    auto.create.topics.enable = true
    auto.leader.rebalance.enable = true
    background.threads = 10
    broker.id = 1
    broker.id.generation.enable = true
    broker.rack = null
    compression.type = producer
    connections.max.idle.ms = 600000
    controlled.shutdown.enable = true
    controlled.shutdown.max.retries = 3
    controlled.shutdown.retry.backoff.ms = 5000
```

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 localhost:9092 --replication-factor 1 --partitions 1 --topic plain-topic
```

```
[root@kafka0 kafka-logs]# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic plain-topic
Error while executing topic command : Timed out waiting for a node assignment. Call: createTopics
[2022-05-17 16:42:55,686] ERROR org.apache.kafka.common.errors.TimeoutException: Timed out waiting for a
node assignment. Call: createTopics
(kafka.admin.TopicCommand$)
[root@kafka0 kafka-logs]#
```

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 localhost: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 correlation 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 bytes 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 correlation 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 bytes 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 correlation 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 bytes 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 localhost: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 --list
[root@kafka0 scripts]#
```

Assign the permission:

```
# /opt/kafka/bin/kafka-acls.sh --bootstrap-server localhost: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=WWRITE, 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 localhost: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=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 localhost: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=localhost: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 --add --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=localhost: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 --add --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 localhost:9092 --topic plain-topic --group bob-group --from-beginning --consumer.config consumer.properties
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic plain-topic --group bob-group --from-beginning --consumer.config consumer.properties
hi
Hello
Message
```

Sent a few messages from the producer console, and you will see that messages are being consumed 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=localhost: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]#
```

This permission alone is not enough to meet the use case. If Charlie runs the consumer group command, he isn't able to see any rows in the output. Charlie needs to read (fetch) offsets of topics in the consumer group. To do this, he must have Describe access to all topics in the group. According to the table above, this second ACL is:

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

or

```
$ /opt/kafka/bin/kafka-acls.sh --bootstrap-server=localhost: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 localhost: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-group --command-config group.properties

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

GROUP      TOPIC      PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG          CONSUMER-ID  HOST
CLIENT-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=localhost: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 consumer.properties
[2018-06-27 21:37:46,223] WARN [Consumer clientId=consumer-1, groupId=console-consumer-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.tools.ConsoleConsumer$)
org.apache.kafka.common.errors.GroupAuthorizationException: Not authorized to access group: console-consumer-47559
```

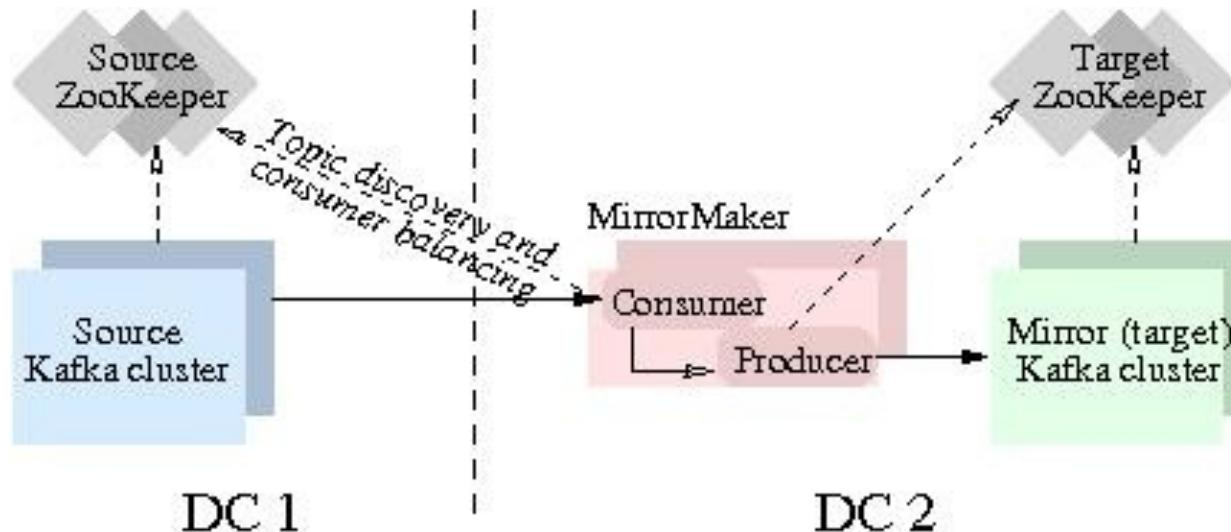
Grant proper authorization.

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

Lab Ends Here

12. Mirroring data between clusters – MirrorMaker – 90 Minutes

In this lab we will demonstrate the replication of message across the cluster for DR.



As shown in the above picture, there are two cluster; a kafka cluster in DC1 and another kafka cluster in DC2

How to set up a mirror

Setting up a mirror is easy - simply start up the mirror-maker processes after bringing up the target cluster. At minimum, the mirror maker takes one or more consumer configurations, a producer configuration and either a whitelist or a blacklist. You need to

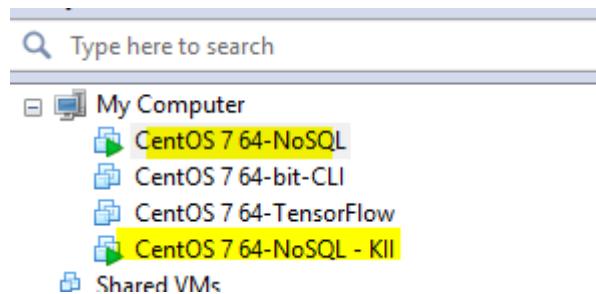
point the consumer to the source cluster's ZooKeeper, and the producer to the mirror cluster's ZooKeeper (or use the broker.list parameter).

Create One more Kafka VM or docker Container, which will act as 2nd DC.

- DC 1
- DC 2

Replication will happen from DC 1 to DC 2.

If you are working using VM player/ Workstation, then it should have 2 VMs as shown below:



DC 1 VM

Start all three brokers.

DC2 VM

Rename the hostname – tos.slave.com

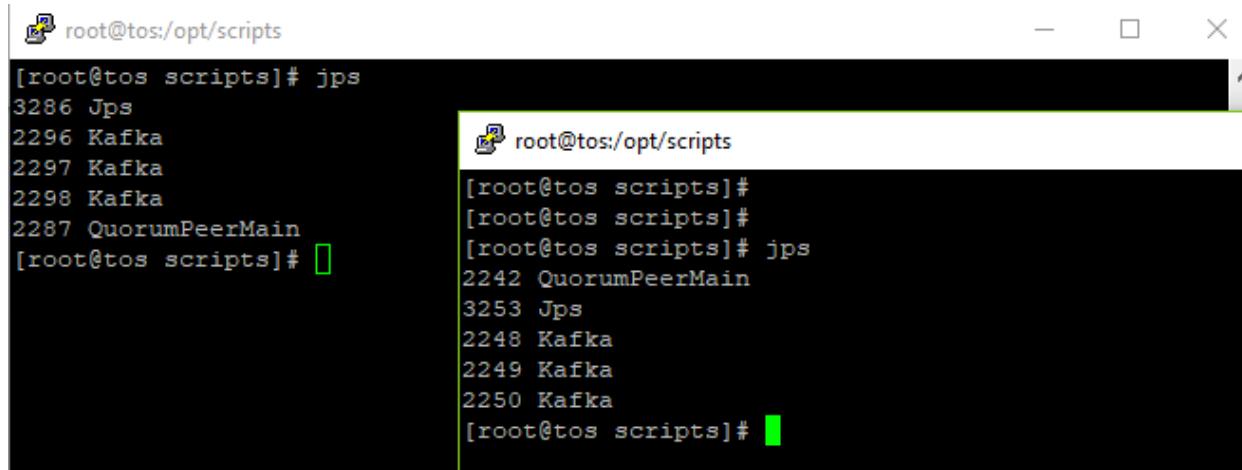
Configure connectivity between the two hosts using /etc/hosts.

You should be able to ping between the two Vms. i.e include in the host file as shown below on both the VM.

```
[root@tos ~]# more /etc/hosts
127.0.0.1    localhost
::1          localhost
10.10.20.24  tos.master.com
10.10.20.26  tos.slave.com
[root@tos ~]#
```

Update the hosts name and its IP addresses accordingly.

Start all three brokers on DC2.



The image shows two terminal windows side-by-side. Both windows have a title bar with a user icon, 'root@tos:/opt/scripts'. The left window's title bar is darker than the right one. Both windows show the command 'jps' being run at the root prompt. The left window displays processes with IDs 3286, 2296, 2297, 2298, and 2287, each associated with the 'Kafka' process name. The right window displays processes with IDs 2242, 3253, 2248, 2249, and 2250, also each associated with the 'Kafka' process name.

```
[root@tos scripts]# jps
3286 Jps
2296 Kafka
2297 Kafka
2298 Kafka
2287 QuorumPeerMain
[root@tos scripts]#
```

```
[root@tos scripts]#
[root@tos scripts]#
[root@tos scripts]# jps
2242 QuorumPeerMain
3253 Jps
2248 Kafka
2249 Kafka
2250 Kafka
[root@tos scripts]#
```

Let us create the following configuration in the data center, DC2 vm. Change the directory to /opt/scripts

Create consumer.props

In this example, the file that lists the properties and values for the consumers that will read messages from the topics in Apache Kafka is named consumer.props. It contains this list:

```
group.id=cg.1  
bootstrap.servers= tos.master.com:9093  
shallow.iterator.enable=false
```

The file that lists the properties and values for the producers that will publish messages to topics in Kafka DC2 is named producer.props. It contains this list:

```
metadata.broker.list= tos.slave.com:9092  
bootstrap.servers = tos.slave.com:9092  
producer.type=sync  
compression.codec=none  
auto.create.topics.enable=true
```

Verify the topic on DC1

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

```
[root@tos kafka]# bin/kafka-topics.sh --zookeeper tos.master.com:2181 -list
CustomerCountry
__consumer_offsets
henry-topic
my-failsafe-topic
my-kafka-topic
test
test-topic
[root@tos kafka]#
```

Here is an example showing how to mirror a single topic (named test-topic) from an input cluster:

If the test-topic is not present, ensure that you have created on the DC1:

```
#/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 2 --partitions 8 --topic test-topic
```

```
[root@tos bin]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper tos.master.com:2181 --replication-factor 2 --partitions 8 --topic test-topic
Created topic "test-topic".
[root@tos bin]#
```

Let us configure replication across the DC using Mirror maker. You need to execute the following on the DC2. It can be executed from any of the DC, however we have created configuration files in the DC2 hence we will be executing this from the /opt/scripts folder of DC2 vm

```
#/opt/kafka/bin/kafka-mirror-maker.sh --consumer.config consumer.props --producer.config producer.props --whitelist test-topic
```

```
[root@tos kafka]# bin/kafka-mirror-maker.sh --consumer.config consumer.props --producer.config producer.props --whitelist test-topic
WARNING: The default partition assignment strategy of the new-consumer-based mirror maker will change from 'range' to 'roundrobin' in an upcoming release (so that better load balancing can be achieved). If you prefer to make this switch in advance of that release add the following to the corresponding new-consumer config: 'partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor'
[2018-06-13 23:59:17,883] WARN The configuration 'auto.create.topics.enable' was supplied but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[2018-06-13 23:59:17,884] WARN The configuration 'compression.codec' was supplied but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[2018-06-13 23:59:17,884] WARN The configuration 'metadata.broker.list' was supplied but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[2018-06-13 23:59:18,108] WARN The configuration 'shallow.iterator.enable' was supplied but isn't a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
```

Note that we specify the list of topics with the `--whitelist` option. This option allows any regular expression using [Java-style regular expressions](#). So you could mirror two topics named *A* and *B* using `--whitelist 'A|B'`. Or you could mirror *all* topics using `--whitelist '*'`. Make sure to quote any regular expression to ensure the shell doesn't try to expand it as a file path. For convenience we allow the use of `,` instead of `|` to specify a list of topics. Combining mirroring with the configuration `auto.create.topics.enable=true` makes it possible to have a replica cluster that will automatically create and replicate all data in a source cluster even as new topics are added.

How to check whether a mirror is keeping up

List the consumer on the DC1

```
#/opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --list
```

```
[root@tos bin]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --list
Note: This will not show information about old Zookeeper-based consumers.
console-consumer-45816
cg.1
[root@tos bin]#
```

Note: `cg.1` is the consumer group of the MirrorMaker.

The consumer groups tool is useful to gauge how well your mirror is keeping up with the source cluster. Note that the *--bootstrap-server* argument should point to the source cluster's broker (DC1 in this scenario). For example:

```
#/opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --describe --group cg.1
```

```
cg.1
[root@tos bin]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --describe --group cg.1
Note: This will not show information about old Zookeeper-based consumers.
[root@tos bin]#
```

Or

```
(base) Henrys-MacBook-Air:~ henrypotsangbam$ /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group cg.1
[ GROUP           TOPIC      PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG    CONSUMER-ID                                     HOST
[   CLIENT-ID
cg.1            test-topic     4          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     5          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     2          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     6          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     3          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     1          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     7          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
cg.1            test-topic     8          -              0              -    cg.1 -0-882629a6-a099-4093-a97e-6fec17e0790c /192.168.29.
132 cg.1 -0
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

Let us send messages on DC1 and verify it on DC2

Send messages from DC1. Ensure that you open a separate terminal for each of the console.

Type few message from the producer console.

```
#/opt/kafka/bin/kafka-console-producer.sh --broker-list tos.master.com:9092 --topic test-topic
```

```
[root@tos bin]# /opt/kafka/bin/kafka-console-producer.sh --broker-list tos.master.com:9092 --topic test
^C[root@tos bin]# /opt/kafka/bin/kafka-console-producer.sh --broker-list tos.master.com:9092 --topic test-topic
>Replicationg
>Yeah its working , You should be able to view this message on the other windows
>
```

Consume messages from a test topic: DC2

```
# /opt/kafka/bin/kafka-console-consumer.sh –bootstrap-server tos.slave.com:2181 --topic test-topic --from-beginning
```

```
Last login: Wed Jun 13 23:20:13 2018 from 10.10.20.1
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --zookeeper tos.slave.com:2181 --topic test-topic --from-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major release. Consider using the new consumer by passing [bootstrap-server] instead of [zookeeper].
Replicationg
Yeah its working , You should be able to view this message on the other windows
```

You are able to receive message from the DC1 to DC2

Verify the messages lagging again: (Connect on the Source Custer)

```
#/opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --describe --group cg.1
```

```
[root@tos bin]# /opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server tos.master.com:9092 --describe --group cg.1
Note: This will not show information about old Zookeeper-based consumers.

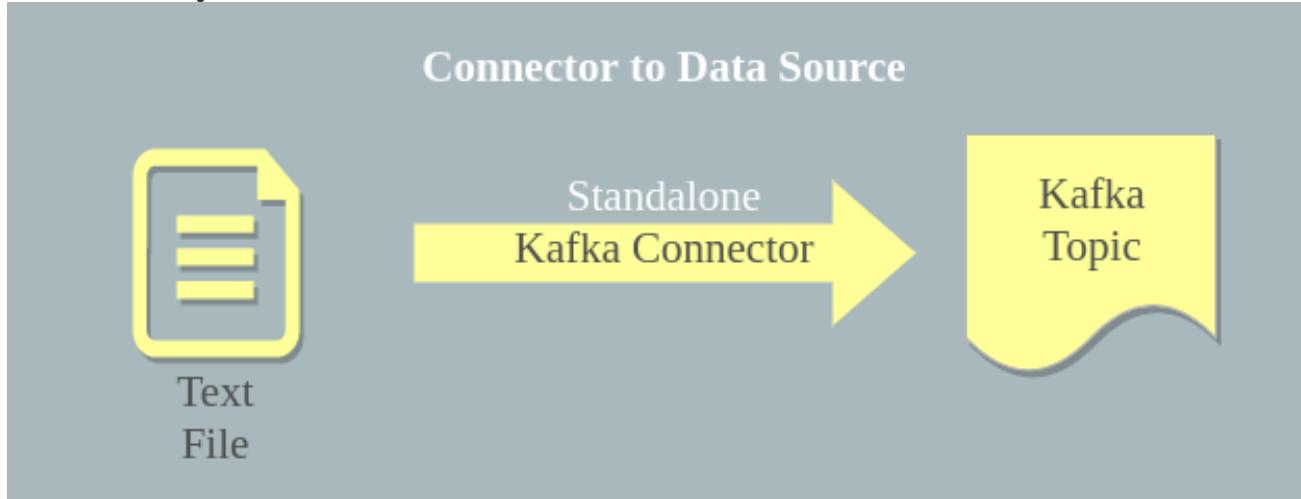
TOPIC      PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG    CONSUMER-ID          HOST
        CLIENT-ID
test-topic  4          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  5          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  2          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  6          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  3          1              1              0    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  1          1              1              0    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  7          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
test-topic  0          -              0              -    cg.1 -0-c7960852-b3ad-429d-a8f5-33e2779cba6e /10.10.20
.26   cg.1 -0
[root@tos bin]#
```

As you can verify that the messages are being replicated. The lag is showing as 0.

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

13. Kafka Connector (File & JDBC) - 150 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.json                  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 localhost: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=FileStreamSource
tasks.max=1
file=/tmp/my-test.txt
topic=my-kafka-connect
```

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=localhost:9092
```

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

```
# The internal converter used for offsets, most will always want to use the built-in default  
internal.key.converter=org.apache.kafka.connect.json.JsonConverter  
internal.value.converter=org.apache.kafka.connect.json.JsonConverter  
internal.key.converter.schemas.enable=false  
internal.value.converter.schemas.enable=false
```

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

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.

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

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.

```
[2018-06-17 12:56:53,991] WARN Couldn't find file /tmp/my-test.txt for FileStreamSourceTask, sleeping to wait for it to be created (org.apache.kafka.connect.file.FileStreamSourceTask:108)
[2018-06-17 12:56:55,535] INFO Cluster ID: rwk6R3n9TYSCArCsSs8V4w (org.apache.kafka.clients.Metadata:265)
[2018-06-17 12:57:53,852] INFO WorkerSourceTask{id=local-file-source-0} Committing 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.WorkerSourceTask: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.WorkerSourceTask: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 localhost:9092 --topic my-kafka-connect --from-beginning
```

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

Update the /tmp/my-test.txt with a new line “Hope this message arrived on the console.” . It should be shown on the consumer console automatically.

The screenshot shows two terminal windows. The top window displays the contents of a file named 'my-test.txt' which contains a test message for a Kafka connector. The bottom window shows the command used to start the Kafka connector.

```
login as: root
root@10.10.20.24's password:
Last login: Sun Jun 17 12:28:03 2018 from 10.10.20.1
[root@tos ~]# vi /tmp/my-test.txt
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server tos.master.com:9092 --topic my-kafka-connect --from-beginning
This Message is from Test File.
It will be consumed by the Kafka Connector.
Hope this message arrived on the console.
root@tos:~
```

```
root@tos:~
```

```
login as: root
root@10.10.20.24's password:
Last login: Sun Jun 17 12:56:11 2018 from 10.10.20.1
[root@tos ~]# vi /tmp/my-test.txt
[root@tos ~]#
```

Let us try with json file.

Stop the kafka connector – ctrl + C of the terminal in which we have started it.

Update the source properties files as shown below: (/opt/scripts/my-file-source.properties)

The screenshot shows a terminal window displaying the contents of a file named 'my-file-source.properties'. The file contains configuration for a local file source connector, specifying the connector class, tasks, file to read, and topic to publish to.

```
#my-file-source.properties config file
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/tmp/my-test.json
topic=my-kafka-connect
~
```

Start the Kafka connector now:

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

Update the /tmp/my-test.json with the following:

```
{  
  "name": "John",  
  "age": 30,  
  "cars": [ "Ford", "BMW", "Fiat" ]  
}
```

You will be able to see the above json document in the consumer console. Start the consumer if you have closed it earlier.

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server tos.master.com:9092 --topic my-kafka-connect --from-beginning  
This Message is from Test File.  
It will be consumed by the Kafka Connector.  
Hope this message arrived on the console.  
{  
  "name": "John",  
  "age": 30,  
  "cars": [ "Ford", "BMW", "Fiat" ]  
},
```

We have successfully configured, file connector in standalone mode.

Next, let us configure JDBC in a distributed mode.

Kafka Connect – JDBC Connector.

Download the connector from the following url.

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

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](#)

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=localhost: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
```

```
nnectorIds=[], taskIds=[], revokedConnectorIds=[], revokedTaskIds=[], delay=0} with rebalance delay: 0 (org.apache.kafka.connect.runtime.distr
ibuted.DistributedHerder:1681)
[2020-11-15 12:59:43,196] INFO [Worker clientId=connect-1, groupId=connect-cluster] Starting connectors and tasks using config offset -1 (org.
apache.kafka.connect.runtime.distributed.DistributedHerder:1208)
[2020-11-15 12:59:43,196] INFO [Worker clientId=connect-1, groupId=connect-cluster] Finished starting connectors and tasks (org.apache.kafka.c
onnect.runtime.distributed.DistributedHerder:1236)
Nov 15, 2020 12:59:43 PM org.glassfish.jersey.internal.Errors logErrors
WARNING: The following warnings have been detected: WARNING: The (sub)resource method listLoggers in org.apache.kafka.connect.runtime.rest.res
ources.LoggingResource contains empty path annotation.
WARNING: The (sub)resource method createConnector in org.apache.kafka.connect.runtime.rest.resources.ConnectorsResource contains empty path an
notation.
WARNING: The (sub)resource method listConnectors in org.apache.kafka.connect.runtime.rest.resources.ConnectorsResource contains empty path ann
otation.
WARNING: The (sub)resource method listConnectorPlugins in org.apache.kafka.connect.runtime.rest.resources.ConnectorPluginsResource contains em
pty path annotation.
WARNING: The (sub)resource method serverInfo in org.apache.kafka.connect.runtime.rest.resources.RootResource contains empty path annotation.

[2020-11-15 12:59:43,307] INFO Started o.e.j.s.ServletContextHandler@c6da8bb{/null,AVAILABLE} (org.eclipse.jetty.server.handler.ContextHandle
r:825)
[2020-11-15 12:59:43,307] INFO REST resources initialized; server is started and ready to handle requests (org.apache.kafka.connect.runtime.re
st.RestServer:319)
[2020-11-15 12:59:43,308] INFO Kafka Connect started (org.apache.kafka.connect.runtime.Connect:57)
[2020-11-15 12:59:43,419] INFO [Worker clientId=connect-1, groupId=connect-cluster] Session key updated (org.apache.kafka.connect.runtime.dist
ributed.DistributedHerder:1570)
```

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.JdbcSourceConnector","type":"source","version":"10.0.0"}, {"class":"org.apache.kafka.connect.file.FileStreamSinkConnector","type":"sink","version":"2.6.0"}, {"class":"org.apache.kafka.connect.file.FileStreamSourceConnector","type":"source","version":"2.6.0"}, {"class":"org.apache.kafka.connect.mirror.MirrorCheckpointConnector","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
#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
```

```
Complete!  
[root@5a1225a251e4 /]# /usr/pgsql-13/bin/postgresql-13-setup initdb  
Initializing database ... OK  
  
[root@5a1225a251e4 /]# systemctl enable postgresql-13  
Created symlink from /etc/systemd/system/multi-user.target.wants/postgresql-13.service to /usr/lib/systemd/system/postgresql-13.service.  
[root@5a1225a251e4 /]# systemctl start postgresql-13  
[root@5a1225a251e4 /]# systemctl status postgresql-13  
● postgresql-13.service - PostgreSQL 13 database server  
  Loaded: loaded (/usr/lib/systemd/system/postgresql-13.service; enabled; vendor preset: disabled)  
  Active: active (running) since Sun 2020-11-15 13:21:23 UTC; 6s ago  
    Docs: https://www.postgresql.org/docs/13/static/  
  Process: 322 ExecStartPre=/usr/pgsql-13/bin/postgresql-13-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)  
 Main PID: 327 (postmaster)  
   CGroup: /docker/5a1225a251e4011bfe46ec657ce6ede8168c9c54e77cecc29df0c095aeebbd37/docker/5a1225a251e4011bfe46ec657ce6ede8168c9c54e77cecc29df0c095aeebbd37/system.slice/postgresql-13.service  
         ├ 327 /usr/pgsql-13/bin/postmaster -D /var/lib/pgsql/13/data/  
         ├ 328 postgres: logger  
         ├ 330 postgres: checkpointer  
         └ 331 postgres: background writer
```

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';
```

```
complete.
[root@5a1225a251e4 /]# sudo -u postgres psql
psql (13.1)
Type "help" for help.

postgres=# ALTER USER postgres PASSWORD 'postgres';
ALTER ROLE
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;
```

```
postgres=# insert into leads values (4,'Tiraj P');
INSERT 0 1
postgres=# select * from leads;
 id |      name
----+
  1 | Henry P
  2 | Rajnita P
  3 | Henderson P
  4 | Tiraj P
(4 rows)

postgres=#
```

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.apache.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 (org.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.apache.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 (org.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 localhost:9092
```

```
[root@4f5607f478bd software]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
__consumer_offsets
connect-configs
connect-offsets
connect-status
postgres-02-leads
test
[root@4f5607f478bd software]#
```

```
#/opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost: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 localhost:9092 --topic postgres-01-leads --from-beginning
```

```
[root@4f5607f478bd software]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic postgres-02-leads --from-beginning
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "name"}], "optional": false, "name": "leads"}, "payload": {"id": 1, "name": "Henry P"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "name"}], "optional": false, "name": "leads"}, "payload": {"id": 2, "name": "Rajnita P"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "name"}], "optional": false, "name": "leads"}, "payload": {"id": 3, "name": "Henderson P"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "name"}], "optional": false, "name": "leads"}, "payload": {"id": 4, "name": "Tiraj P"}}
```

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 Transformation 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 localhost:9092  
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic postgres-02-leads --from-beginning
```

```
[root@kafka0 software]# /opt/kafka/bin/kafka-topics.sh --list --bootstrap-server localhost:9092  
__consumer_offsets  
connect-configs  
connect-offsets  
connect-status  
postgres-01-leads  
postgres-02-leads  
[root@kafka0 software]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic postgres-02-leads --from-beginning  
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "fullname"}]}, "optional": false, "name": "leads"}, "payload": {"id": 1, "fullname": "Henry P"}}  
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "fullname"}]}, "optional": false, "name": "leads"}, "payload": {"id": 2, "fullname": "Rajnita P"}}  
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "fullname"}]}, "optional": false, "name": "leads"}, "payload": {"id": 3, "fullname": "Henderson P"}}  
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "fullname"}]}, "optional": false, "name": "leads"}, "payload": {"id": 4, "fullname": "Tiraj P"}}
```

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

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/tasks
[{"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": "postgres", "task.class": "io.confluent.connect.jdbc.source.JdbcSourceTask", "connection.user": "postgres", "poll.interval.ms": "3600000", "name": "jdbc_source_connector_postgresql_01", "connection.url": "jdbc:postgresql://localhost:5432/postgres"}}][root@kafka0 /]#
```

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": "RUNNING", "worker_id": "172.18.0.2:8083"}, "tasks": [{"id": 0, "state": "RUNNING", "worker_id": "172.18.0.2:8083"}], "type": "source"}}, "jdbc_source_connector_postgresql_02": {"status": {"name": "jdbc_source_connector_postgresql_02", "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 /]#
```

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 ~]#
```

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 -----

14. Schema Registry - Manage Schemas for Topics – 30 Minutes

Pre-requisite: Install kafka server.

Download and install confluent kafka : Schema Registry only.

Start the Zookeeper and Broker.

Update the following in /opt/confluent/etc/schema-registry/schema-registry.properties

kafkastore.bootstrap.servers=PLAINTEXT://kafka0:9092

Start the Registry:

```
#cd /opt/confluent/  
#schema-registry-start /opt/confluent/etc/schema-registry/schema-registry.properties
```

```
WARNING: A provider io.confluent.kafka.schemaregistry.rest.resources.ModeResource registered in SERVER runtime does not implement any provider interfaces applicable in the SERVER runtime. Due to constraint configuration problems the provider io.confluent.kafka.schemaregistry.rest.resources.ModeResource will be ignored.  
Sep 28, 2020 7:47:55 AM org.glassfish.jersey.internal.inject.Providers checkProviderRuntime  
WARNING: A provider io.confluent.kafka.schemaregistry.rest.resources.SubjectsResource registered in SERVER runtime does not implement any provider interfaces applicable in the SERVER runtime. Due to constraint configuration problems the provider io.confluent.kafka.schemaregistry.rest.resources.SubjectsResource will be ignored.  
Sep 28, 2020 7:47:55 AM org.glassfish.jersey.internal.inject.Providers checkProviderRuntime  
WARNING: A provider io.confluent.kafka.schemaregistry.rest.resources.SubjectVersionsResource registered in SERVER runtime does not implement any provider interfaces applicable in the SERVER runtime. Due to constraint configuration problems the provider io.confluent.kafka.schemaregistry.rest.resources.SubjectVersionsResource will be ignored.  
Sep 28, 2020 7:47:55 AM org.glassfish.jersey.internal.inject.Providers checkProviderRuntime  
WARNING: A provider io.confluent.kafka.schemaregistry.rest.resources.SchemasResource registered in SERVER runtime does not implement any provider interfaces applicable in the SERVER runtime. Due to constraint configuration problems the provider io.confluent.kafka.schemaregistry.rest.resources.SchemasResource will be ignored.  
[2020-09-28 07:47:56,447] INFO HV000001: Hibernate Validator 6.0.17.Final (org.hibernate.validator.internal.util.Version:21)  
[2020-09-28 07:47:57,161] INFO Started o.e.j.s.ServletContextHandler@6b3871d6{/null,AVAILABLE} (org.eclipse.jetty.server.handler.ContextHandler:825)  
[2020-09-28 07:47:57,215] INFO Started o.e.j.s.ServletContextHandler@aa10649{/ws,null,AVAILABLE} (org.eclipse.jetty.server.handler.ContextHandler:825)  
[2020-09-28 07:47:57,272] INFO Started NetworkTrafficServerConnector@186f8716{HTTP/1.1,[http/1.1]}{0.0.0.0:8081} (org.eclipse.jetty.server.AbstractConnector:330)  
[2020-09-28 07:47:57,273] INFO Started @11223ms (org.eclipse.jetty.server.Server:399)  
[2020-09-28 07:47:57,275] INFO Server started, listening for requests... (io.confluent.kafka.schemaregistry.rest.SchemaRegistryMain:44)
```

Create a topic and let us bind schema to this.

```
# kafka-topics.sh --create --bootstrap-server kafka0:9092 --partitions 1 --replication-factor 1 --topic my-kafka  
#kafka-topics.sh --list --bootstrap-server kafka0:9092
```

Use the Schema Registry API to add a schema for the topic my-kafka.

```
#curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema": {"type": "record", "name": "Payment", "namespace": "com.tos", "fields": [{"name": "id", "type": "string"}, {"name": "amount", "type": "double"}]}' http://localhost:8081/subjects/my-kafka-value/versions
```

```
[root@kafka0 code]# kafka-topics.sh --list --bootstrap-server kafka0:9092
__consumer_offsets
_schemas
my-kafka
[root@kafka0 code]# curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema": {"type": "record", "name": "Payment", "namespace": "com.tos", "fields": [{"name": "id", "type": "string"}, {"name": "amount", "type": "double"}]} }' http://localhost:8081/subjects/my-kafka-value/versions
{"id":1}[root@kafka0 code]#
```

```
#curl -X GET http://localhost:8081/schemas/ids/1
```

```
[root@kafka0 code]# curl -X GET http://localhost:8081/schemas/ids/1
{"schema": {"type": "record", "name": "Payment", "namespace": "com.tos", "fields": [{"name": "id", "type": "string"}, {"name": "amount", "type": "double"}]} }[root@kafka0 code]#
```

You can verify the topic which maintains the schema details in the broker.

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

```
[root@kafka0 ~]# kafka-topics.sh --list --bootstrap-server kafka0:9092
__consumer_offsets
__schemas
my-kafka
transactions
[root@kafka0 ~]#
```

List all subjects associated with a given ID

Use this two-step process to find subjects associated with a given ID:

List all the subjects.

```
#curl -X GET http://localhost:8081/subjects
```

Iterate through the output from the subject list as follows, and check for the ID in the results:

```
#curl -X GET http://localhost:8081/subjects/<INSERT SUBJECT NAME>/versions/latest
```

E.x

```
#curl -X GET http://localhost:8081/subjects/my-kafka-value/versions/latest
```

```
[root@kafka0 code]# curl -X GET http://localhost:8081/subjects  
["my-kafka-value"] [root@kafka0 code]#  
[root@kafka0 code]# curl -X GET http://localhost:8081/subjects/my-kafka-value/versions/latest  
{"subject": "my-kafka-value", "version": 2, "id": 1, "schema": "{\"type\": \"record\", \"name\": \"Payment\", \"namespace\": \"com.tos\", \"fields\": [{\"name\": \"id\", \"type\": \"string\"}, {"name": \"amount\", \"type\": \"double\"}]}"} [root@kafka0 code]# █
```

#curl -X GET <http://localhost:8081/subjects>

Delete all schema versions registered under the subject “my-kafka-value”

curl -X DELETE http://localhost:8081/subjects/my-kafka-value

<https://docs.confluent.io/platform/current/schema-registry/develop/using.html>

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

15. Performance & Tuning – 60 Minutes (ND)

We are working on a single broker for this lab. We will monitor some metrics of broker using JConsole. Ensure that you have restored the server.properties before enabling Security.

Before we continue, you will need JMX enabled. It is turned off by default, so you'll want to enable JMX it as follows:

- 1.) Add the line “`export JMX_PORT=9999`” to the JMX settings section inside the `/opt/kafka/bin/kafka-run-class.sh` file at this location, where “9999” is the port you wish to use:

```
# JMX settings

export JMX_PORT=9999
if [ -z "$KAFKA_JMX_OPTS" ]; then
  KAFKA_JMX_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false"
fi

# JMX port to use
if [ $JMX_PORT ]; then
```

Or

```
export JMX_PORT=9999
```

We will use the second option for our lab. Ensure that you execute the above export in the console where you will start the broker.

2.) Restart/start the Kafka Producer.

```
#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
5076 Kafka
4805 QuorumPeerMain
5115 Jps
[root@tos scripts]#
```

3.) You can verify the JMX port using the following command.

```
#lsof -i:9999
```

```
[root@tos scripts]# lsof -i:9999
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
java    5076 root    82u  IPv6  49901      0t0    TCP *:distinct (LISTEN)
[root@tos scripts]#
```

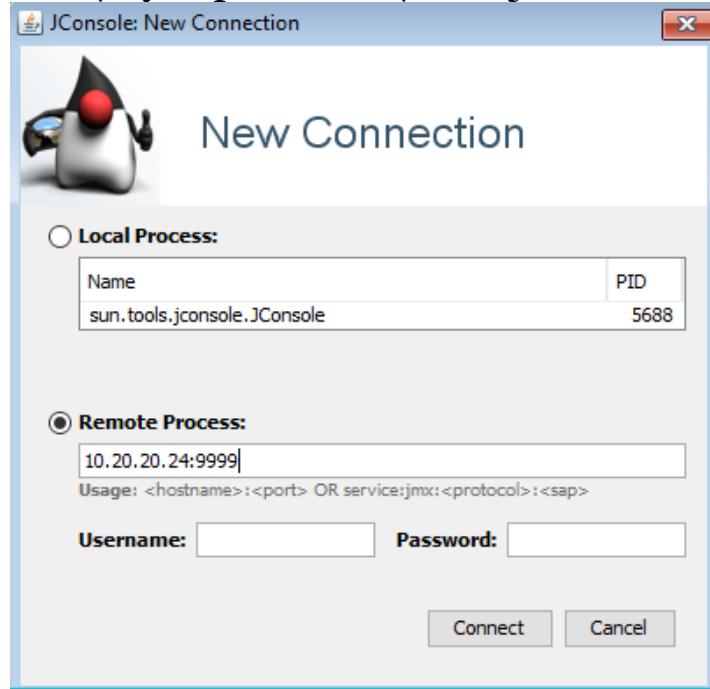
Collecting Kafka performance metrics with JConsole

JConsole is a simple Java GUI that ships with the Java Development Kit (JDK). It provides an interface for exploring the full range of metrics Kafka emits via JMX. If the JDK was installed to a directory in your system path, you can start JConsole by running: jconsole.

Otherwise, check in your_JDK_install_dir/bin

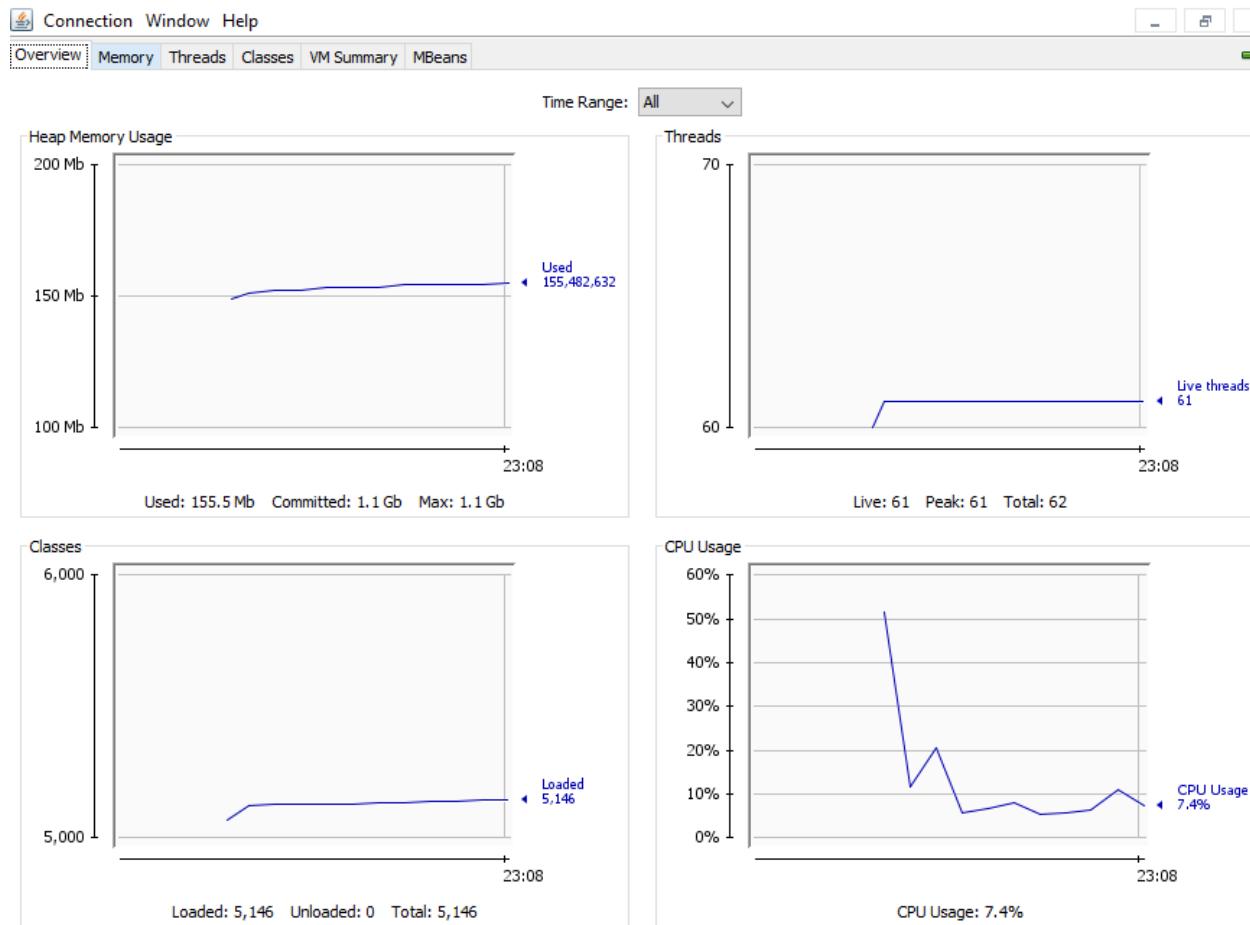
To view metrics in JConsole, you can select the relevant local process or monitor a remote process using the node's IP address (Kafka uses port 9999 for JMX by default), though it is recommended that you connect remotely, as JConsole can be resource-intensive:

```
#D:\MyExperiment\Java\jdk1.8.0_131\bin>jconsole.exe
```

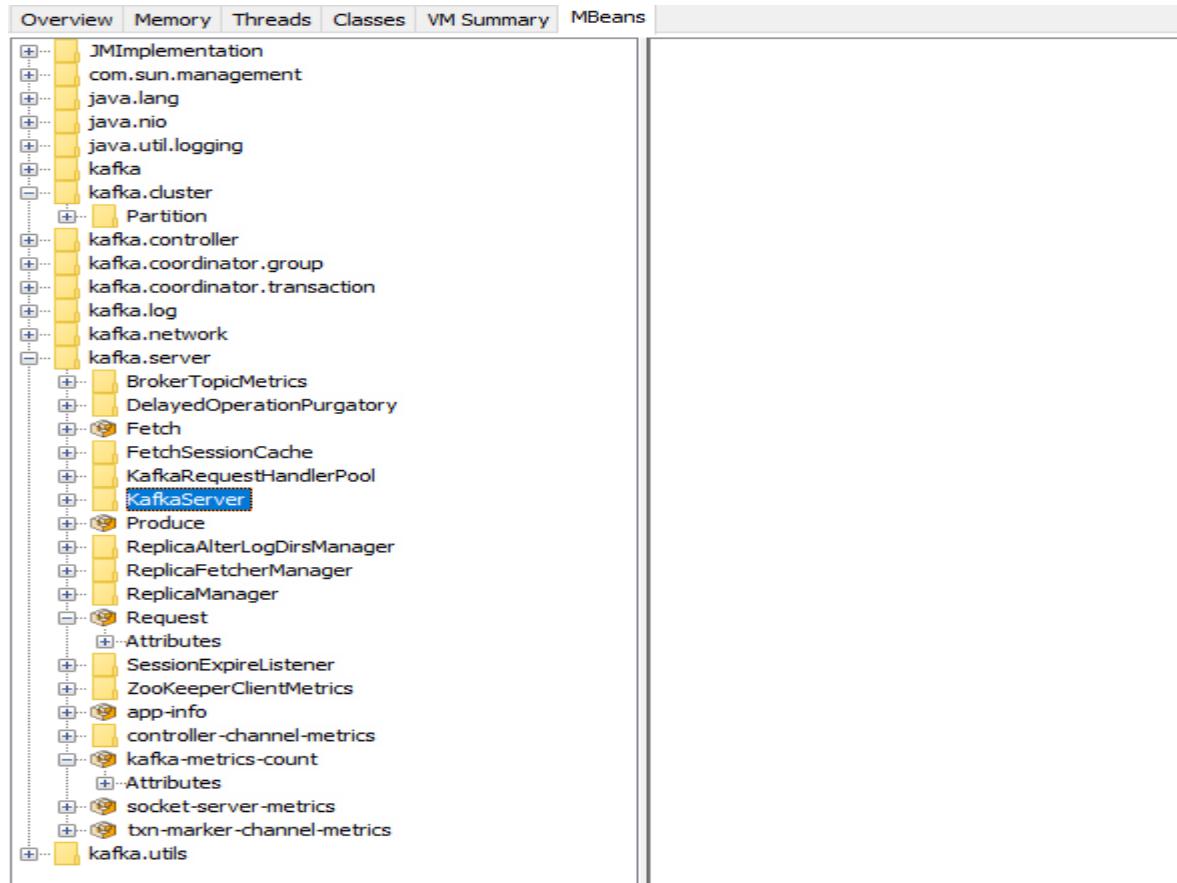


Enter your server IP i.e 10.10.20.24:9999

Click Connect and accept the default warning.



The MBeans tab brings up all the JMX paths available:



Kafka server (broker) metrics

Producer metrics

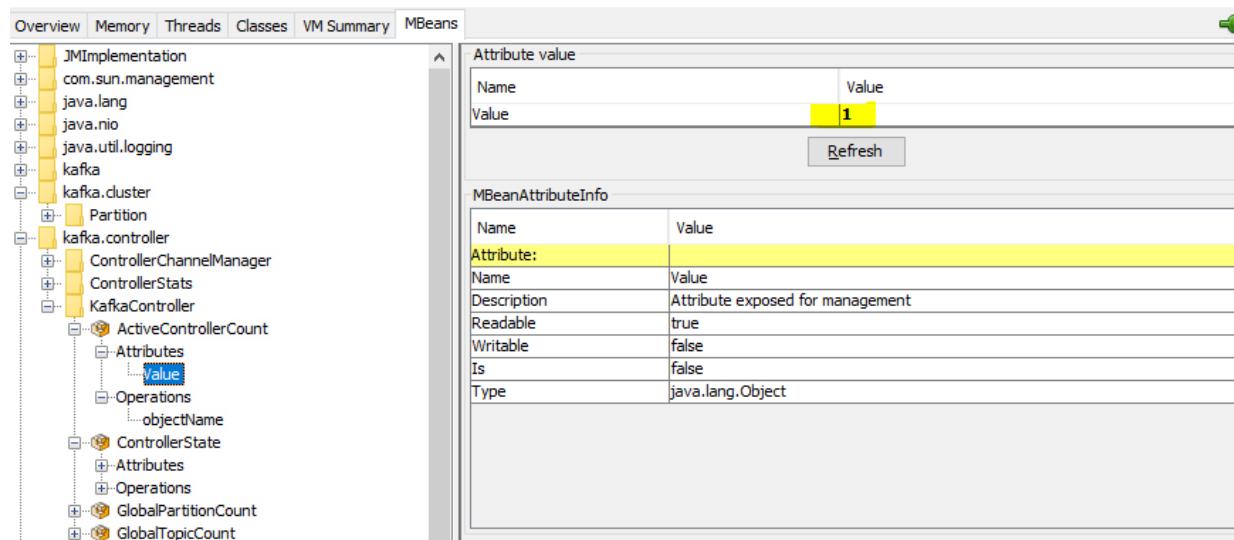
Consumer metrics

Broker metrics

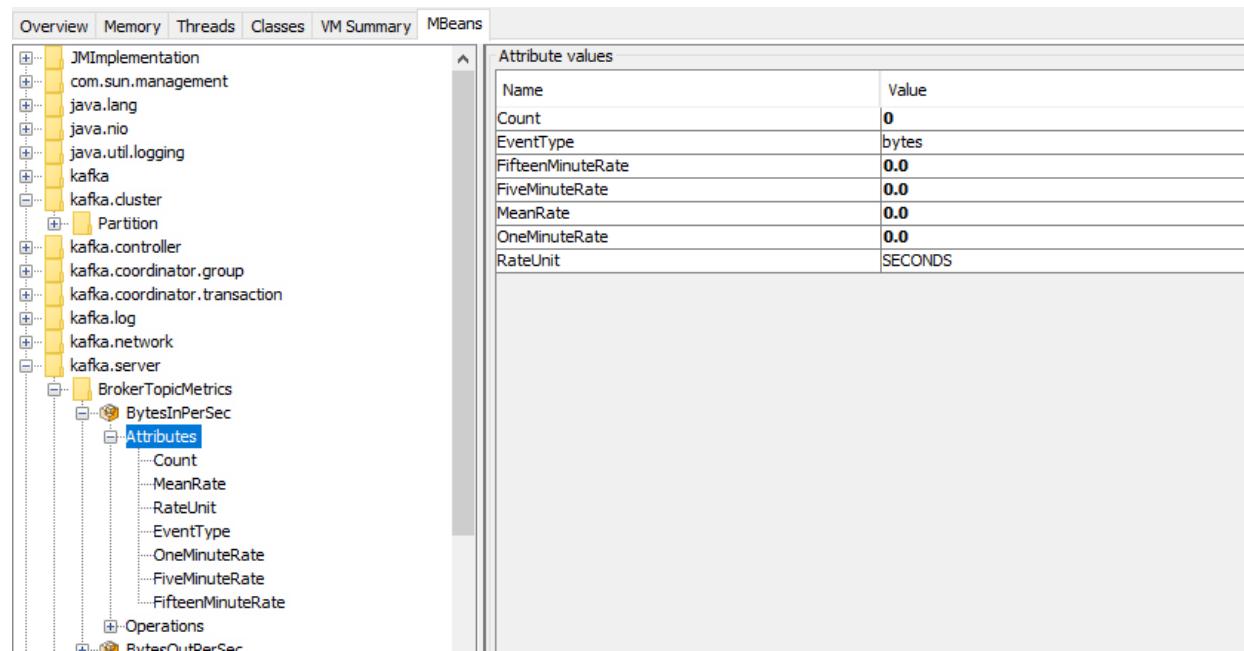
Kafka server metrics (aka broker metrics) provide a window into brokers, the backbone of the pipeline. Because all messages must pass through a Kafka broker in order to be consumed, monitoring and alerting on issues as they emerge in your broker cluster is critical. Broker metrics can be broken down into three classes:

- Kafka-emitted metrics
- host-level metrics
- JVM garbage collection metrics

ActiveControllerCount	kafka.controller:type=KafkaController, name=ActiveControllerCount	Number of active controllers in cluster	Resource: Error
-----------------------	---	---	-----------------



BytesInPerSec BytesOutPerSec	kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec	Aggregate incoming/outgoing byte rate
---------------------------------	---	---------------------------------------



Screenshot of the JConsole interface showing the MBeans tab. The left pane displays a tree view of MBeans, and the right pane shows attribute values.

MBeans Tree:

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- kafka
- kafka.cluster
 - Partition
- kafka.controller
- kafka.coordinator.group
- kafka.coordinator.transaction
- kafka.log
- kafka.network
- kafka.server
 - BrokerTopicMetrics
 - BytesInPerSec
 - BytesOutPerSec
 - Attributes
 - Count
 - MeanRate
 - RateUnit
 - EventType
 - OneMinuteRate
 - FiveMinuteRate
 - FifteenMinuteRate
 - Operations

Attribute Values Table:

Name	Value
Count	0
EventType	bytes
FifteenMinuteRate	0.0
FiveMinuteRate	0.0
MeanRate	0.0
OneMinuteRate	0.0
RateUnit	SECONDS

Consumers and producers

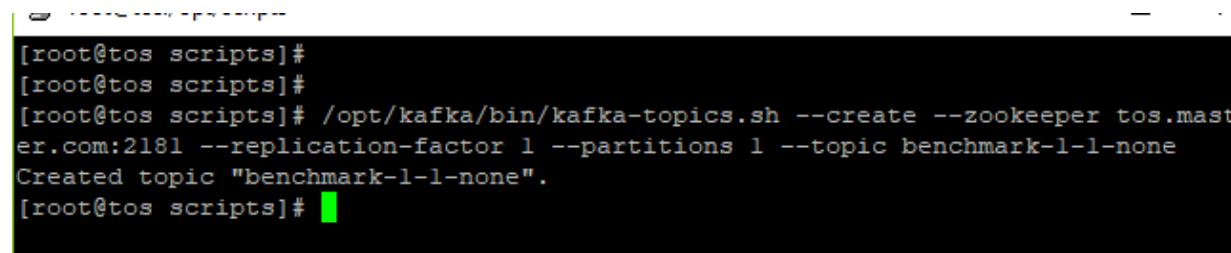
To collect JMX metrics from your consumers and producers, follow the same steps outlined above, replacing port 9999 with the JMX port for your producer or consumer, and the node's IP address.

Baseline Performance

The Kafka documentation claims that producers can push about 50MB/sec through a system with a single broker as long as the batch size is not too small (the default value of 200 should be large enough). We were able to verify this claim very quickly for Kafka 0.7.2 by running the following command on a fresh installation

```
# Producer Throughput: Single producer thread, no replication, no compression
```

```
/opt/kafka/bin/kafka-topics.sh --create \  
--bootstrap-server localhost:9092 \  
--replication-factor 1 \  
--partitions 1 \  
--topic benchmark-1-1-none
```



A terminal window showing the execution of a shell command to create a Kafka topic named "benchmark-1-1-none". The command uses the kafka-topics.sh script with specific parameters: --create, --bootstrap-server set to localhost:9092, --replication-factor 1, --partitions 1, and --topic set to benchmark-1-1-none. The output of the command shows that the topic was successfully created.

```
[root@tos scripts]#  
[root@tos scripts]#  
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper tos.mast  
er.com:2181 --replication-factor 1 --partitions 1 --topic benchmark-1-1-none  
Created topic "benchmark-1-1-none".  
[root@tos scripts]#
```

```
The above topic will be used for creating bench mark.
```

Let us execute the below benchmark that will simulate the pushing of 15000000 records.

```
/opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark-1-1-none \
--num-records 15000000 \
--record-size 100 \
--throughput 15000000 \
--producer-props \
acks=1 \
bootstrap.servers=tos.master.com:9092 \
buffer.memory=67108864 \
compression.type=none \
batch.size=8196
```

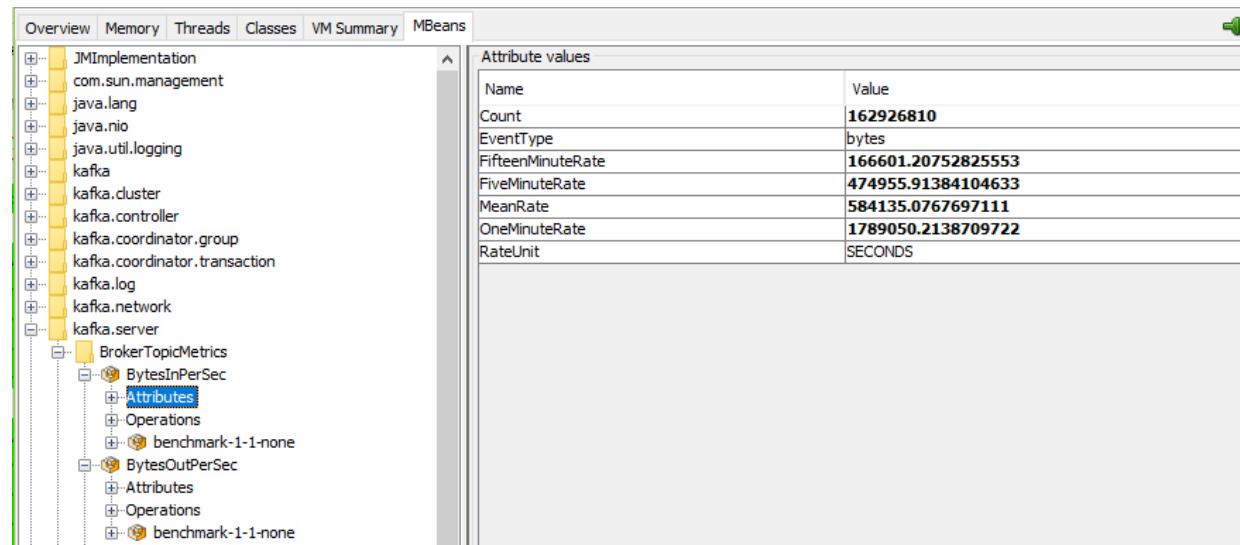
```
[root@tos scripts]# /opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark
-1-1-none \
> --num-records 15000000 \
> --record-size 100 \
> --throughput 15000000 \
> --producer-props \
> acks=1 \
> bootstrap.servers=tos.master.com:9092 \
> buffer.memory=67108864 \
> compression.type=none \
> batch.size=8196
13765 records sent, 2715.0 records/sec (0.26 MB/sec), 1451.8 ms avg latency, 407
6.0 max latency.
8436 records sent, 1547.0 records/sec (0.15 MB/sec), 6740.5 ms avg latency, 9424
.0 max latency.
9175 records sent, 1823.7 records/sec (0.17 MB/sec), 11841.4 ms avg latency, 142
```

This is because in an effort to increase availability and durability, version 0.8 introduced intra-cluster replication support, and by default a producer waits for an acknowledgement response from the broker on every message (or batch of messages if async mode is used). It is possible to mimic the old behavior, but we were not very interested in that given that we intend to use replication in production.

Performance degraded further once we started using a sample of real ~1KB sized log messages rather than the synthetic messages produced by the Kafka tool, resulting in a throughput of about 10 MB/sec.

All throughput numbers refer to uncompressed data.

Verify using the Jconsole:



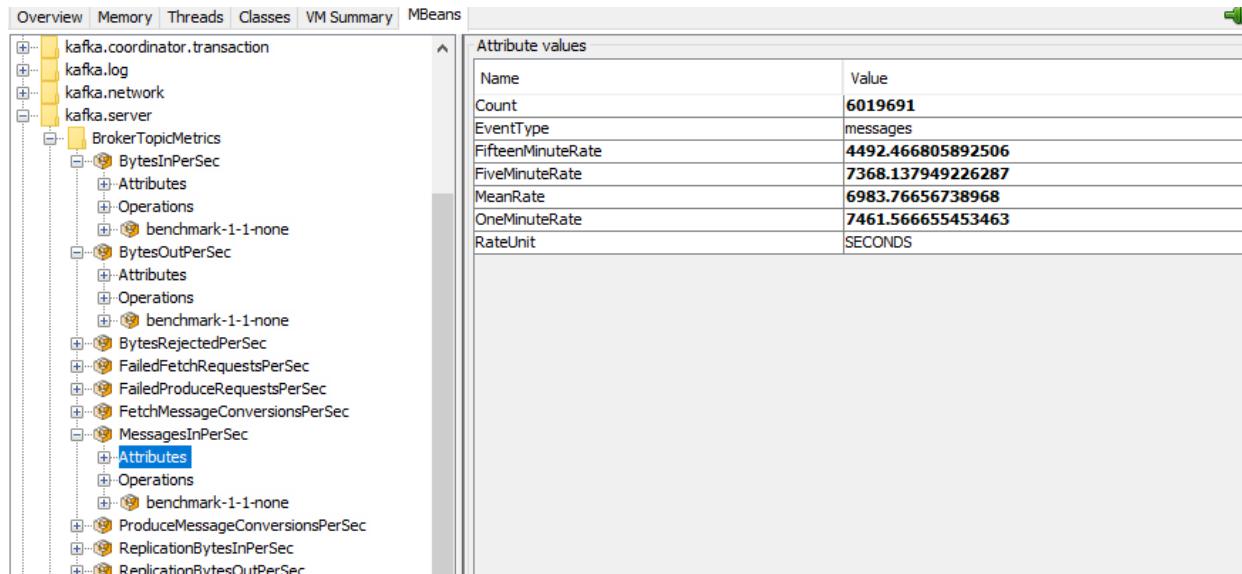
```
## Producer Throughput: 3 producer thread, no replication, no compression

/opt/kafka/bin/kafka-topics.sh --create \
--bootstrap-server localhost:9020 \
--partitions 1 \
--topic benchmark-3-0-none

/opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark-1-1-none \
--num-records 15000000 \
--record-size 100 \
--throughput 15000000 \
--producer-props \
acks=1 \
bootstrap.servers=tos.master.com:9092 \
buffer.memory=67108864 \
compression.type=none \
batch.size=8196
```

```
[root@tos scripts]# /opt/kafka/bin/kafka-producer-perf-test.sh --topic benchmark  
-l-l-none \  
> --num-records 15000000 \  
> --record-size 100 \  
> --throughput 15000000 \  
> --producer-props \  
> acks=1 \  
> bootstrap.servers=tos.master.com:9092 \  
> buffer.memory=67108864 \  
> compression.type=none \  
> batch.size=8196  
59346 records sent, 11859.7 records/sec (1.13 MB/sec), 2183.7 ms avg latency, 34  
19.0 max latency.  
171752 records sent, 34350.4 records/sec (3.28 MB/sec), 5475.9 ms avg latency, 6  
776.0 max latency.
```

```
19.0 max latency.  
171752 records sent, 34350.4 records/sec (3.28 MB/sec), 5475.9 ms avg latency, 6  
776.0 max latency.  
104858 records sent, 20959.0 records/sec (2.00 MB/sec), 8550.3 ms avg latency, 1  
1153.0 max latency.  
175602 records sent, 30312.8 records/sec (2.89 MB/sec), 13427.1 ms avg latency,  
15975.0 max latency.  
147926 records sent, 29514.4 records/sec (2.81 MB/sec), 18332.1 ms avg latency,  
20395.0 max latency.  
170938 records sent, 34167.1 records/sec (3.26 MB/sec), 20456.3 ms avg latency,  
20922.0 max latency.  
112176 records sent, 22435.2 records/sec (2.14 MB/sec), 21444.9 ms avg latency,  
22348.0 max latency.  
151989 records sent, 30391.7 records/sec (2.90 MB/sec), 20898.0 ms avg latency,  
21820.0 max latency.  
145851 records sent, 28326.1 records/sec (2.70 MB/sec), 20725.3 ms avg latency,  
21662.0 max latency.  
114107 records sent, 22803.2 records/sec (2.17 MB/sec), 21857.6 ms avg latency,  
22880.0 max latency.  
150657 records sent, 30131.4 records/sec (2.87 MB/sec), 22220.4 ms avg latency,  
22757.0 max latency.  
120324 records sent, 23751.3 records/sec (2.27 MB/sec), 22182.4 ms avg latency,  
22591.0 max latency.
```



In this way you can perform benchmark of your broker.

Demo:

First let's create a topic for the test data. The following example will result in a topic named **my-perf-test** with 2 partitions, a replication factor of 1 and retention time of 24 hours. Replace broker0 as needed for the Kafka cluster in your environment:

```
#kafka-topics.sh --create --topic my-perf-test \
--partitions 2 --config retention.ms=86400000
\ --config min.insync.replicas=2 --bootstrap-server kafka0:9092
```

Then run the producer performance test script with different configuration settings. The following example will use the topic created above to store 1 lakh messages with a size of 1 KiB each. The -1 value for *--throughput* means that messages are produced as quickly as possible, with no throttling limit. Kafka producer related configuration properties like *acks* and *bootstrap.servers* are mentioned as part of *--producer-props* argument:

```
# kafka-producer-perf-test.sh --topic my-perf-test \
--num-records 100000 \
--record-size 1024 \
--throughput -1 \
--producer-props acks=1 \
bootstrap.servers=localhost:9092
```

```
[root@kafka0 code]# kafka-producer-perf-test.sh --topic my-perf-test \
> --num-records 100000 \
> --record-size 1024 \
> --throughput -1 \
> --producer-props acks=1 \
> bootstrap.servers=localhost:9092
7666 records sent, 1531.4 records/sec (1.50 MB/sec), 1013.3 ms avg latency, 1876.0 ms max latency.
12750 records sent, 2545.4 records/sec (2.49 MB/sec), 3116.2 ms avg latency, 5239.0 ms max latency.
8175 records sent, 1632.4 records/sec (1.59 MB/sec), 6957.0 ms avg latency, 9480.0 ms max latency.
8775 records sent, 1754.6 records/sec (1.71 MB/sec), 11696.2 ms avg latency, 13566.0 ms max latency.
14655 records sent, 2931.0 records/sec (2.86 MB/sec), 14802.3 ms avg latency, 15516.0 ms max latency.
14670 records sent, 2927.6 records/sec (2.86 MB/sec), 12559.2 ms avg latency, 14597.0 ms max latency.
17295 records sent, 3456.9 records/sec (3.38 MB/sec), 10034.4 ms avg latency, 10787.0 ms max latency.
12270 records sent, 2412.0 records/sec (2.36 MB/sec), 9613.4 ms avg latency, 10595.0 ms max latency.
100000 records sent, 2386.179250 records/sec (2.33 MB/sec), 9400.79 ms avg latency, 15516.00 ms max latency, 10062 ms 50th, 15252 ms
95th, 15469 ms 99th, 15502 ms 99.9th.
[root@kafka0 code]#
```

In this example, roughly 2.3k messages are produced per second on average, with a maximum latency of approx. 15 seconds.

To run the consumer performance test script as an example, will read 1 lakh messages from our same topic *my-perf-test*.

```
#kafka-consumer-perf-test.sh --topic my-perf-test --broker-list kafka0:9092 --messages 100000
```

```
--version                                     Display Kafka version.
[root@kafka0 code]# kafka-consumer-perf-test.sh --topic my-perf-test --broker-list kafka0:9092 --messages 100000
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec,
fetch.nMsg.sec
2022-02-28 15:43:59:610, 2022-02-28 15:44:06:745, 98.1396, 13.7547, 100495, 14084.7933, 1933, 5202, 18.8658, 19318.5313
[root@kafka0 code]#
```

The throughput (MB.sec and nMsg.sec) which are of interest : 13.75 and 14084.

Increase the no of partitions:

```
# kafka-topics.sh --create --topic my-perf-test4 \ --partitions 12 --config  
retention.ms=86400000 \ --config min.insync.replicas=2 --bootstrap-server kafka0:9092
```

```
# kafka-producer-perf-test.sh --topic my-perf-test4 \  
--num-records 100000 \  
--record-size 1024 \  
--throughput -1 \  
--producer-props acks=1 \  
bootstrap.servers=kafka0:9092
```

```
[root@kafka0 code]# kafka-producer-perf-test.sh --topic my-perf-test4 --num-records 100000 --record-size 1024 --throughput -1 --producer-props acks=1 bootstrap.servers=kafka0:9092  
11946 records sent, 2373.1 records/sec (2.32 MB/sec), 521.9 ms avg latency, 1748.0 ms max latency.  
19260 records sent, 3843.5 records/sec (3.75 MB/sec), 2231.3 ms avg latency, 4188.0 ms max latency.  
20556 records sent, 4111.2 records/sec (4.01 MB/sec), 5283.8 ms avg latency, 6877.0 ms max latency.  
24264 records sent, 4848.9 records/sec (4.74 MB/sec), 6761.5 ms avg latency, 8210.0 ms max latency.  
100000 records sent, 4004.324671 records/sec (3.91 MB/sec), 4737.75 ms avg latency, 8210.00 ms max latency, 5609 ms 50th, 7438 ms 95th, 7888 ms 99th, 8152 ms 99.9th.  
[root@kafka0 code]#
```

Verify the nos. after increasing the partition. The differences will be observed if you have multiple nodes.

Tuning Brokers

Topics are divided into partitions. Each partition has a leader. Topics that are properly configured for reliability will consist of a leader partition and 2 or more follower partitions. When the leaders are not balanced properly, one might be overworked, compared to others.

Depending on your system and how critical your data is, you want to be sure that you have sufficient replication sets to preserve your data. For each topic, It is recommends starting with one partition per physical storage disk and one consumer per partition.

```
#kafka-topics.sh --bootstrap-server kafka0:9092 --describe --topic my-perf-test4
```

```
[root@kafka0 code]# kafka-topics.sh bootstrap-server kafka0:9092 --describe --topic my-perf-test4
Topic: my-perf-test4    TopicId: xBpQBcWQfaDtHf4ZXxDQ PartitionCount: 12    ReplicationFactor: 1    Configs: segment.bytes=1073741824,retention.ms=86400000
Topic: my-perf-test4    Partition: 0    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 1    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 2    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 3    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 4    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 5    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 6    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 7    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 8    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 9    Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 10   Leader: 0    Replicas: 0    Isr: 0
Topic: my-perf-test4    Partition: 11   Leader: 0    Replicas: 0    Isr: 0
[root@kafka0 code]#
```

Tuning Producers

When you use Producer.send(), you fill up buffers on the producer. When a buffer is full, the producer sends the buffer to the Kafka broker and begins to refill the buffer.

batch.size measures batch size in total bytes instead of the number of messages.

```
#kafka-producer-perf-test.sh --topic my-perf-test4 --num-records 100000 --record-size 1024 --throughput -1 --producer-props acks=1 batch.size=86384 bootstrap.servers=kafka0:9092
```

```
[root@kafka0 code]# kafka-producer-perf-test.sh --topic my-perf-test4 --num-records 100000 --record-size 1024 --throughput -1 --producer-props acks=1 batch.size=96384 bootstrap.servers=kafka0:9092
38240 records sent, 7648.0 records/sec (7.47 MB/sec), 69.3 ms avg latency, 1565.0 ms max latency.
100000 records sent, 10260.619741 records/sec (10.02 MB/sec), 74.23 ms avg latency, 1565.00 ms max latency, 68 ms 50th, 140 ms 95th,
175 ms 99th, 204 ms 99.9th.
[root@kafka0 code]#
```

No of throughput increases and latency max is about 1.5 seconds only.

Tuning Consumers

Consumers can create throughput issues on the other side of the pipeline. The maximum number of consumers in a consumer group for a topic is equal to the number of partitions. You need enough partitions to handle all the consumers needed to keep up with the producers.

Consumers in the same consumer group split the partitions among them. Adding more consumers to a group can enhance performance (up to the number of partitions). Adding

more consumer groups does not affect performance. Execute the following command from 2 terminals.

```
#kafka-consumer-perf-test.sh --topic my-perf-test4 --broker-list kafka0:9092 --messages 100000 --group cg
```

```
[root@kafka0 my-kafka-0]# kafka-consumer-perf-test.sh --topic my-perf-test4 --broker-list kafka0:9092 --messages 100000 --group cg
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec,
fetch.nMsg.sec
2022-02-28 16:51:53:378, 2022-02-28 16:52:06:294, 97.6973, 7.5640, 100042, 7745.5869, 3336, 9580, 10.1980, 10442.7975
[root@kafka0 my-kafka-0]#
```

- To check the ISR set for topic partitions, run the following command:

```
kafka-topics.sh --bootstrap-server kafka0:9092 --describe --topic my-perf-test4
```

```
[root@kafka0 kafka-logs]# kafka-topics.sh --bootstrap-server kafka0:9092 --describe --topic my-perf-test4
Topic: my-perf-test4    TopicId: xBrpQBcWQfaDtHf4ZXoxDQ PartitionCount: 12    ReplicationFactor: 1    Configs: se
gment.bytes=1073741824,retention.ms=86400000
      Topic: my-perf-test4    Partition: 0    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 1    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 2    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 3    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 4    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 5    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 6    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 7    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 8    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 9    Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 10   Leader: 0    Replicas: 0    Isr: 0
      Topic: my-perf-test4    Partition: 11   Leader: 0    Replicas: 0    Isr: 0
[root@kafka0 kafka-logs]# ]
```

- If a partition leader dies, new leader is selected from the ISR set. There will be no data loss. If there is no ISR, unclean leader election can be used with the risk of data-loss.
- Unclean leader election occurs if `unclean.leader.election.enable` is set to true. By default, this is set to false.

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

16. Using REST Proxy

Pre-requisites: Install Kafka Broker.

Refer confluent Lab – Installation of Confluent Broker.

Update the broker information in the confluent config:

Start zookeeper and Kafka Single Node – broker.

Navigate to the Schema Registry properties file (`/opt/confluent/etc/schema-registry/schema-registry.properties`) and specify the following properties:

```
# Specify the address the socket server listens on, e.g. listeners = PLAINTEXT://your.host.name:9092
listeners=http://0.0.0.0:8081

# The host name advertised in ZooKeeper.

# List of Kafka brokers to connect to, e.g. PLAINTEXT://hostname:9092,SSL://hostname2:9092
kafkastore.bootstrap.servers=PLAINTEXT://localhost:9092
```

Start Schema Registry. Run this command in its own terminal.

```
/opt/confluent/bin/schema-registry-start /opt/confluent/etc/schema-registry/schema-registry.properties
```


Confluent REST Proxy

By default the server starts bound to port 8082, does not specify a unique instance ID (required to safely run multiple proxies concurrently), and expects ZooKeeper to be available at `localhost:2181` and Schema Registry at <http://localhost:8081>.

Verify the `etc/kafka-rest/kafka-rest.properties`

Start by running the REST Proxy and the services it depends on: ZooKeeper, Kafka, and Schema Registry.

```
/opt/confluent/bin/kafka-rest-start /opt/confluent/etc/kafka-rest/kafka-rest.properties
```

```
# Get a list of topics
$ curl "http://localhost:8082/topics"
```

```
curl "http://localhost:8082/topics/test"
```

```
[root@893094f7bb92 /]# curl "http://localhost:8082/topics"
[{"_schemas", "log", "test"}][root@893094f7bb92 /]# curl "http://localhost:8082/topics/test"
{"name": "test", "configs": {"compression.type": "producer", "leader.replication.throttled.replicas": "", "message.downconversion.enable": "true", "min.insync.replicas": "1", "segment.jitter.ms": "0", "cleanup.policy": "delete", "flush.ms": "9223372036854775807", "follower.replication.throttled.replicas": "", "retention.ms": "604800000", "segment.bytes": "1073741824", "flush.messages": "9223372036854775807", "message.format.version": "2.6-IV0", "max.compaction.lag.ms": "9223372036854775807", "file.delete.delay.ms": "60000", "max.message.bytes": "1048588", "min.compaction.lag.ms": "0", "message.timestamp.type": "CreateTime", "preallocate": "false", "min.cleanable.dirty.ratio": "0.5", "index.interval.bytes": "4096", "unclean.leader.election.enable": "false", "retention.bytes": "-1", "delete.retention.ms": "86400000", "segment.ms": "604800000", "message.timestamp.difference.max.ms": "9223372036854775807", "segment.index.bytes": "10485760"}, "partitions": [{"partition": 0, "leader": 0, "replicas": [{"broker": 0, "leader": true, "in_sync": true}]}]}][root@893094f7bb92 /]#
```

```
# Produce a message with JSON data
$ curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
--data '{"records": [{"value": {"name": "testUser"} } ]}' \
"http://localhost:8082/topics/jsontest"

# Create a consumer for JSON data, starting at the beginning of the topic's
# log. The consumer group is called "my_json_consumer" and the instance is "my_consumer_instance".
$ curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" \
--data '{"name": "my_consumer_instance", "format": "json", "auto.offset.reset": "earliest"}' \
http://localhost:8082/consumers/my_json_consumer

# Subscribe the consumer to a topic
$ curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" --data '{"topics": ["jsontest"]}' \
http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/subscription
# No content in response
```

```
# Then consume some data from a topic using the base URL in the first response.

$ curl -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/records

# Finally, close the consumer with a DELETE to make it leave the group and clean up
# its resources.

$ curl -X DELETE -H "Accept: application/vnd.kafka.v2+json" \
http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance
# No content in response
```

```
[root@893094f7bb92 ~]# curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
> --data '[{"records":[{"value":{"name": "testUser"}}]}]' \
> "http://localhost:8082/topics/jsontest"
{"offsets":[{"partition":0,"offset":0,"error_code":null,"error":null}],"key_schema_id":null,"value_schema_id":null}[root@893094f7bb92 ~]# curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" -H "Accept: application/vnd.kafka.v2+json" \
> --data '{"name": "my_consumer_instance", "format": "json", "auto.offset.reset": "earliest"}' \
> "http://localhost:8082/consumers/my_json_consumer"
{"instance_id": "my_consumer_instance", "base_uri": "http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance"}[root@893094f7bb92 ~]# curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" -H "Accept: application/vnd.kafka.v2+json" \
> --data '["topics":[]]' \
> "http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/subscription"
[root@893094f7bb92 ~]# curl -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
> "http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/records"
[{"topic": "jsontest", "key": null, "value": {"name": "testUser"}, "partition": 0, "offset": 0}][root@893094f7bb92 ~]# curl -X DELETE -H "Accept: application/vnd.kafka.v2+json" \
> "http://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance"
[root@893094f7bb92 ~]#
```

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

17. Using Kubernetes

minikube

<https://docs.docker.com/docker-for-mac/kubernetes/>

19. 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.apa
che.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
# Many listeners expose security protocols, the default is for them to be the s
ame
```

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 details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

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

20. LOG verification + segment Sizing

21. 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/oooooooooooooooooooo.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
\kafka 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.DumpLogSegments --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 valuesize: 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 valuesize: 43 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence: -1 isTransactional: false headerKeys: [] payload: It will be consumed by the Kafka 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>