

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.	Kafka cluster – 90 Minutes.....	27
7.	Kafka Connector (File & JDBC) - 150 Minutes	53
8.	Schema Registry - Manage Schemas for Topics – 30 Minutes	82
9.	Performance & Tuning – 60 Minutes (ND).....	87
10.	Errors	101
I.	{test=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient).....	101
11.	Annexure Code:	103
II.	DumplogSegment.....	103
III.	Resources.....	104

Hardware:

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

Software Inventory:

- Zookeeper Version: apache-zookeeper-3.6.1-bin.tar
- Apache kafka : 2.13-3.1.0

- JDK 11.0.16
- Eclipse for Linux. (Any Latest version for JEE Development)
- Status : Color is Verified

Last Updated: Oct 24 - 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
```

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

Optional:

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

You can verify the container from a separate terminal:

```
(base) Henrys-MacBook-Air:~ henrypotsangbam$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
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   kafka0
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

Perform the installation after this as a normal server.

Update the server.properties.

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

```
advertised.listeners=PLAINTEXT://localhost:9092
```

After installation to start the zookeeper and kafka broker:

Open two separate terminals and execute the following

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

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

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

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

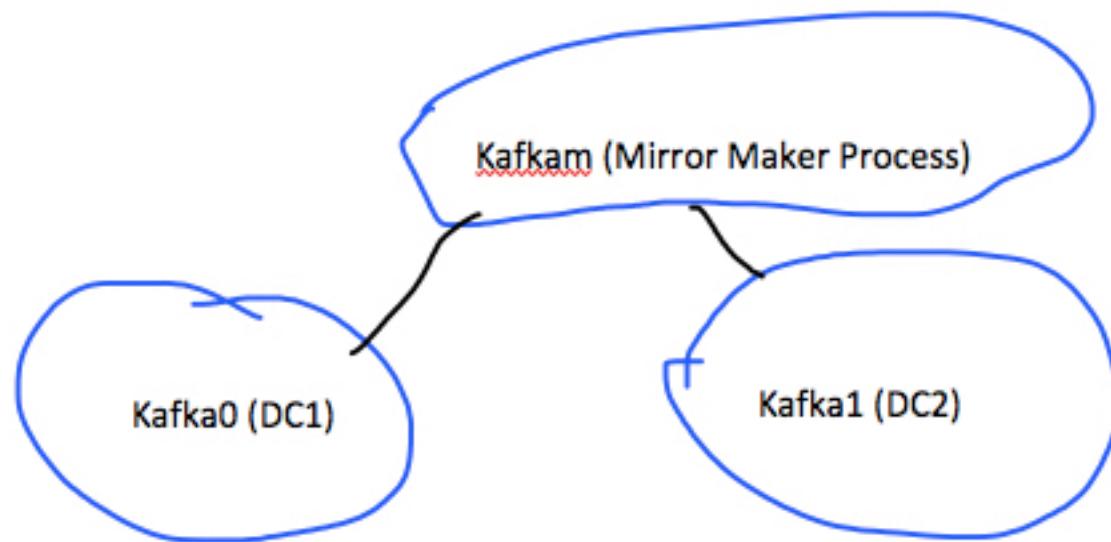
Kafka Mirror Maker: (It will execute the Kafka Mirroring process)

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

Kafka Mirror Architecture:

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.

[kafka_2.13-3.2.1.tgz](#)

jdk-11.0.16_linux-aarch64_bin.tar
apache-zookeeper-3.8.0-bin.tar.gz

2. Installation of Kafka – 90 Mins

You need to install java before installing zookeeper and Kafka.

Installation of Java

```
#tar -xvf jdk-11.0.16_linux* -C /opt  
#cd /opt  
#mv jdk* jdk
```

Set the above path in the PATH variable and JAVA_HOME

Update profile as follow:

```
#vi ~/.bashrc  
  
export JAVA_HOME=/opt/jdk  
export PATH=$PATH:$JAVA_HOME/bin
```

Update the shell scripts using the following command.

```
#bash
```

Installing Zookeeper

You can choose any of the options given below:

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

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

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

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

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

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

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

Start the zookeeper using the following scripts.

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

```
[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) – Skip.

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

Option II Ends Here.

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

```
#yum install telnet
```

```
#telnet localhost 2181
```

```
➤ srvr
```

11 Kafka – Administration

```
[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]#
```

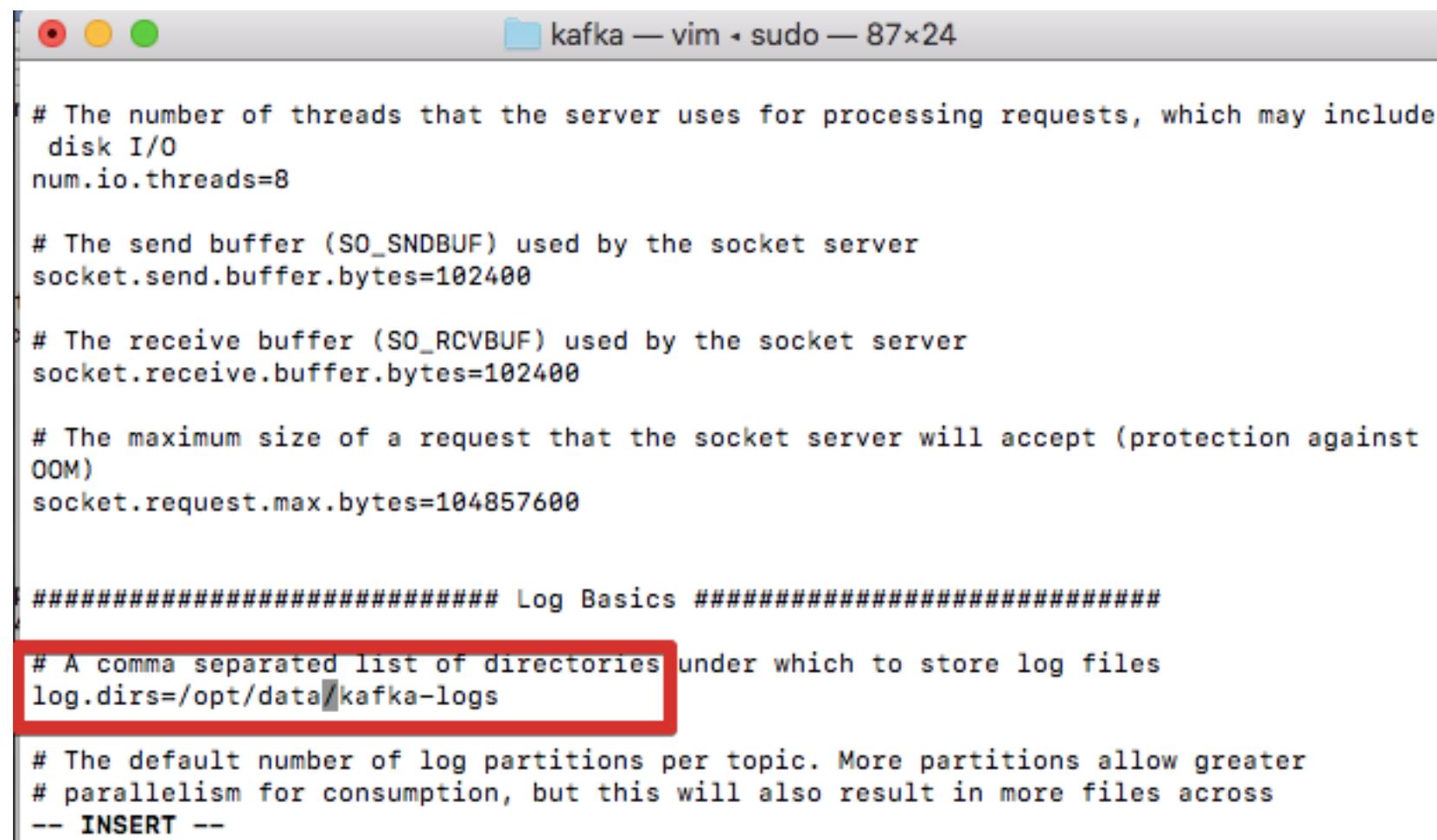
After zookeeper installation, let us install the Kafka Broker.

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

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



The screenshot shows a terminal window titled "kafka — vim - sudo — 87x24". The window displays the contents of the server.properties file. A red box highlights the line "log.dirs=/opt/data/kafka-logs", which is the line being modified to change the log storage directory. The rest of the configuration file includes settings for threads, send/receive buffers, request max bytes, and log basics.

```
# 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. Don't include the ---- line.

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

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

To reinitialize the Cluster.

```
#vi reinitializeCluster.sh
```

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

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

```
cp /opt/kafka/config/server.properties_plain /opt/kafka/config/server.properties  
echo "Reinitialize Successfully"
```

Lab Installation completes End here.

3. Installation Confluent Kafka (Local) – 30 Minutes

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

Install Confluent kafka with the following step.

Inflate the confluent kafka compress file as shown below:

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

Rename the folder.

```
#cd /opt
```

```
# mv confluent-7.2.2 confluent
```

Set the environment variable for the Confluent Platform directory.

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

Set your PATH variable:

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

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

type the following to activate the script.

```
#bash
```

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

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

In this lab you will be able to create a topic and perform some operations to understand the information about topic like partition and replication.

You need to start the broker using startABroker.sh. The script should be in /opt/scripts folder

```
#sh startABroker.sh  
#jps
```

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

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

Create and verify details about topic:

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

```
# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server 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. 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://kafkao:9092
advertised.listeners=PLAINTEXT://kafkao:9092
log.dirs=/opt/data/kafka-logs/kafka-0
```

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

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

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

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

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

```
broker.id=2
listeners=PLAINTEXT://kafkao:9094
advertised.listeners=PLAINTEXT://kafkao: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$ChangeEventProcessThread)
[2018-05-16 15:17:51,723] INFO Kafka version : 1.1.0 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:17:51,723] INFO Kafka commitId : fdcf75ea326b8e07 (org.apache.kafka.common.utils.AppInfoParser)
[2018-05-16 15:17:51,728] INFO [KafkaServer id=2] started (kafka.server.KafkaServer)
[root@tos:/opt/kafka-config] [2018-05-16 15:15:08,676] INFO [Transaction Marker Channel Manager 1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2018-05-16 15:15:09,267] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
Give thes[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)
Create[2018-05-16 15:15:09,285] INFO [KafkaServer id=1] started (kafka.server.KafkaServer)
```

Create a replicated topic my-failsafe-topic

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

Open a separate terminal and execute the following;

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

```
[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 kafka0:9094,kafka0:9092  
--topic my-failsafe-topic --from-beginning
```

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

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

```
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server 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 kafka0:9092,kafka0:9093 --topic my-failsafe-topic
```

Notice we start Kafka producer and pass it a list of Kafka Brokers to use via the parameter `--broker-list`.

Now use the start producer script to launch the producer as follows.

Now send messages

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

Producer Console.

```
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)

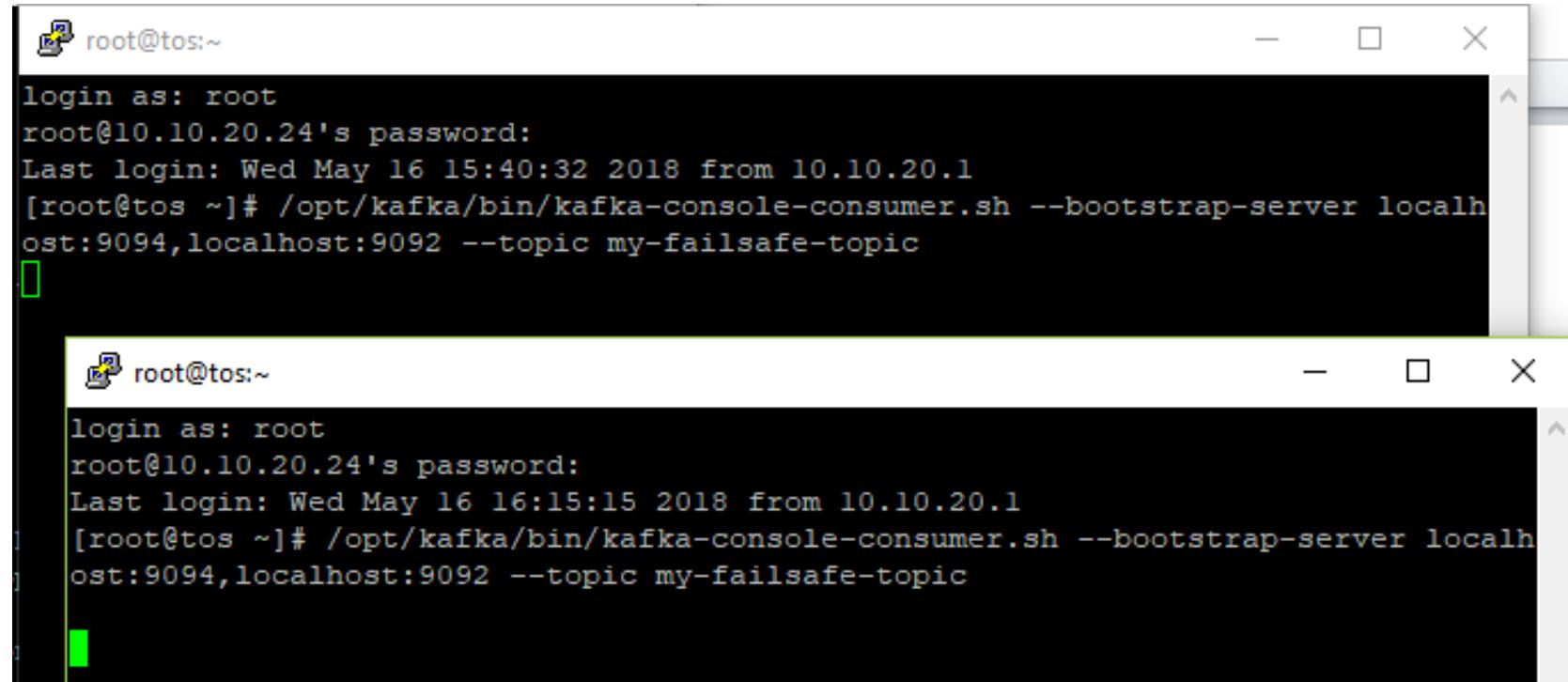
Consumer 1.

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

Consumer 2.

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafkao:9094,kafkao: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 command-line interface for a Kafka consumer. Both windows are titled 'root@tos:~' and show the same command being run:

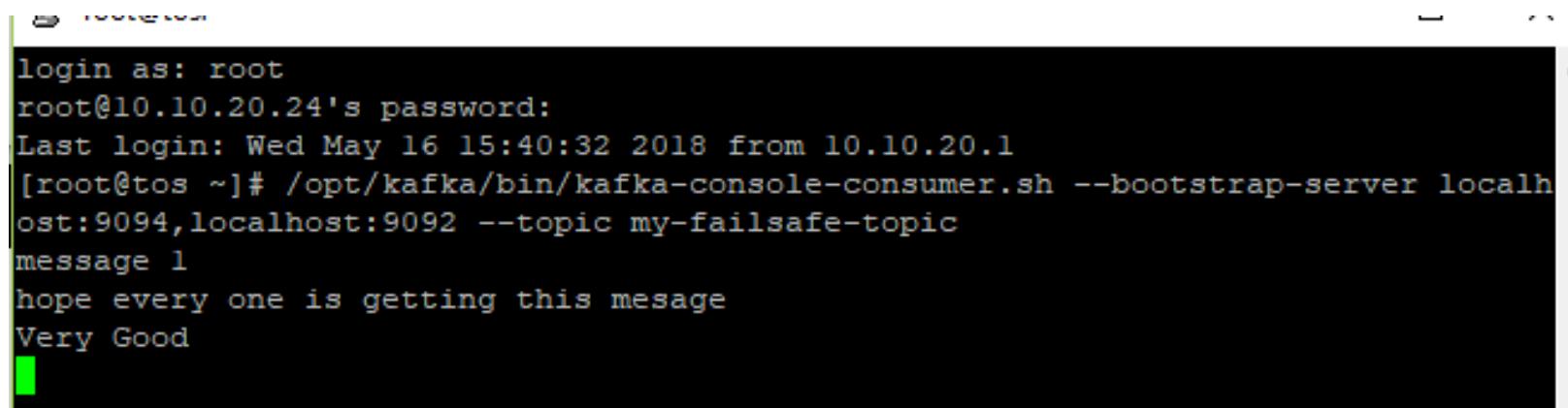
```
root@tos:~  
login as: root  
root@10.10.20.24's password:  
Last login: Wed May 16 15:40:32 2018 from 10.10.20.1  
[root@tos ~]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic
```

The terminal windows have standard window controls (minimize, maximize, close) at the top right. The background of the windows is black, and the text is white.

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
>
```

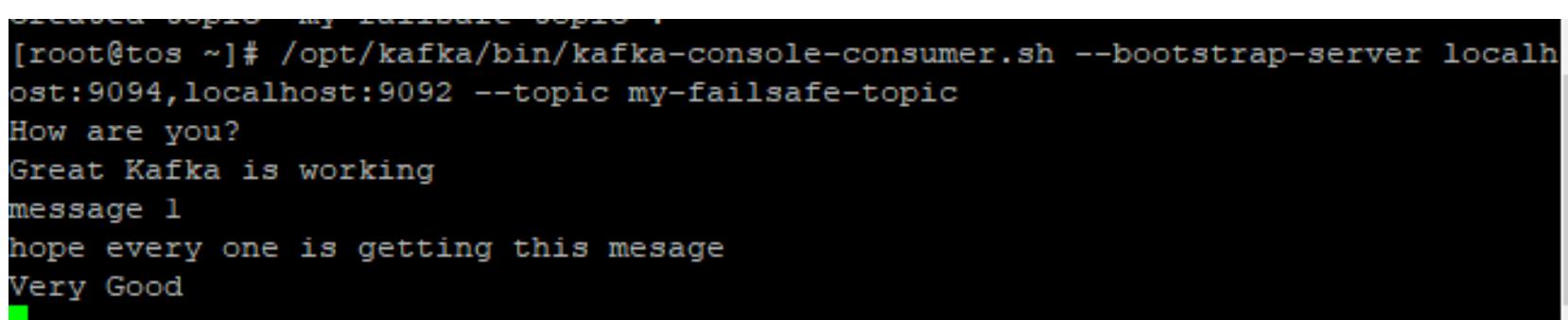
Consumer Console 1st, you should be able to view the messages whatever you type on the producer console after the new consumer console was started.

A screenshot of a terminal window titled "root@tos". The window shows a root shell session. The user has typed the command "/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic". The output shows the message "message 1 hope every one is getting this mesage Very Good".

```
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 screenshot of a terminal window titled "root@tos". The window shows a root shell session. The user has typed the command "/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9094,localhost:9092 --topic my-failsafe-topic". The output shows the message "How are you? Great Kafka is working message 1 hope every one is getting this mesage Very Good".

```
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 kafka0:9094,kafka0:9092  
--topic my-failsafe-topic --consumer-property group.id=mygroup
```

Notice that the script is the same as before except we added **--consumer-property group.id=mygroup** which will put every consumer that runs with this script into the **mygroup** consumer group.

Now we just run the producer and three consumers.

Run Producer Console

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

Now send eight messages from the Kafka producer console.

Producer Console

```
[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 local
host:9094,localhost:9092 --topic my-failsafe-topic --consumer-property group.
id=mygroup
message 5
message 6
message 7
```

Notice the first consumer gets messages m3 and m5.

2nd Kafka Consumer gets m2, m6

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

Notice the second consumer gets messages m2 and m6.

3rd Kafka Consumer gets m1, m4, m7

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

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

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

Kafka Consumer Failover

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

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

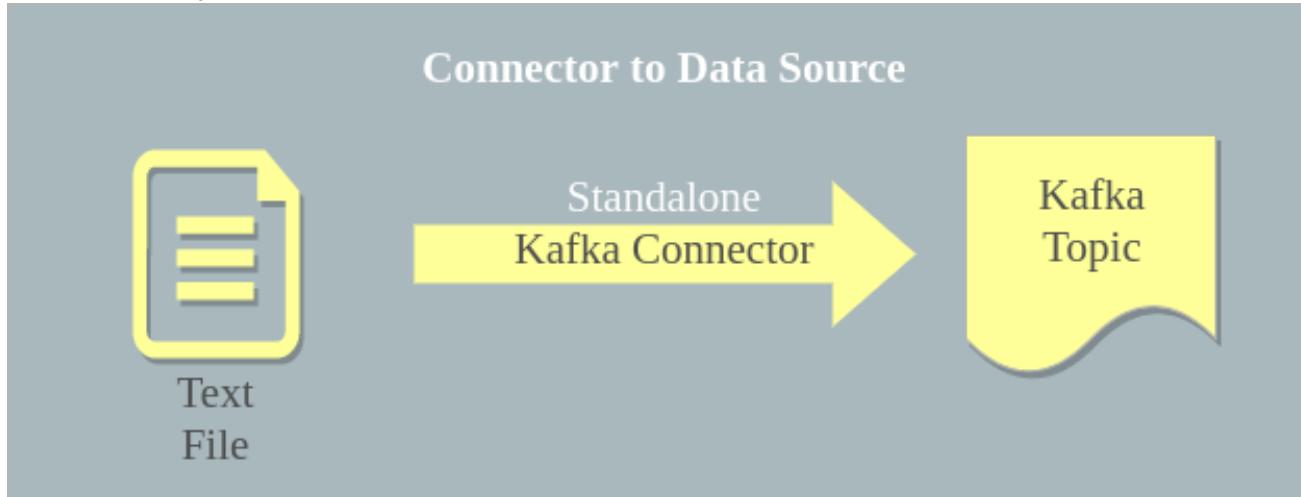
to Stop All 3 Brokers.

```
# 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 -----

7. 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 kafka:9092 \
--replication-factor 1 \
--partitions 1 \
--topic my-kafka-connect
```

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

Creating a Source Config File

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

Open a new file.

```
#vi my-file-source.properties
```

Paste the following instruction in the above file.

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

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

Creating a Worker Config File.

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

Create a file.

```
#vi my-standalone.properties
```

Update with the following contents.

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

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

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

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

Download the file connector.

<https://www.confluent.io/hub/mmolimar/kafka-connect-fs>

Installation

Confluent Hub CLI installation

Use the [Confluent Hub client](#) to install this connector with:

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

 [Copy](#)

Download installation

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

[Download](#)

By downloading you agree to the [terms of use](#) and [software license agreement](#).

Download the above file.

```
#mkdir -p /software/plugins/
```

Rename the folder:

```
#mv mmolimar-kafka-connect-fs-1.3.0/ /software/plugins/kafka-file
```

Open a terminal.

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

Create a file and paste the following text.

```
#vi /tmp/my-test.txt
```

This Message is from Test File.

It will be consumed by the Kafka Connector.

Running Kafka Connect.

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

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

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

```
[2018-06-17 12:56:53,991] WARN Couldn't find file /tmp/my-test.txt for 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.
```

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 postgres db. We will configure postgresdb 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 PostgresDB:
#docker run -it --name postgresdb --privileged -p 5432:5432 centos:7 /usr/sbin/init
docker exec -it postgresdb bash
Add both the container in same network.
#docker network connect spark-net postgresdb
]

The following command will install and initialize the postgres DB

#yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

Or

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

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

Enable and start the service:

```
systemctl enable postgresql-13
systemctl start postgresql-13
```

```
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';
```

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

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

Pre-requisite: Install kafka server.

Download and install confluent kafka : Schema Registry only. (Refer the Confluent Installation.)

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 -----

9. Performance & Tuning – 60 Minutes (ND)

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
```

```
[root@tos scripts]#  
[root@tos scripts]#  
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper tos.master.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 150000000 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
```

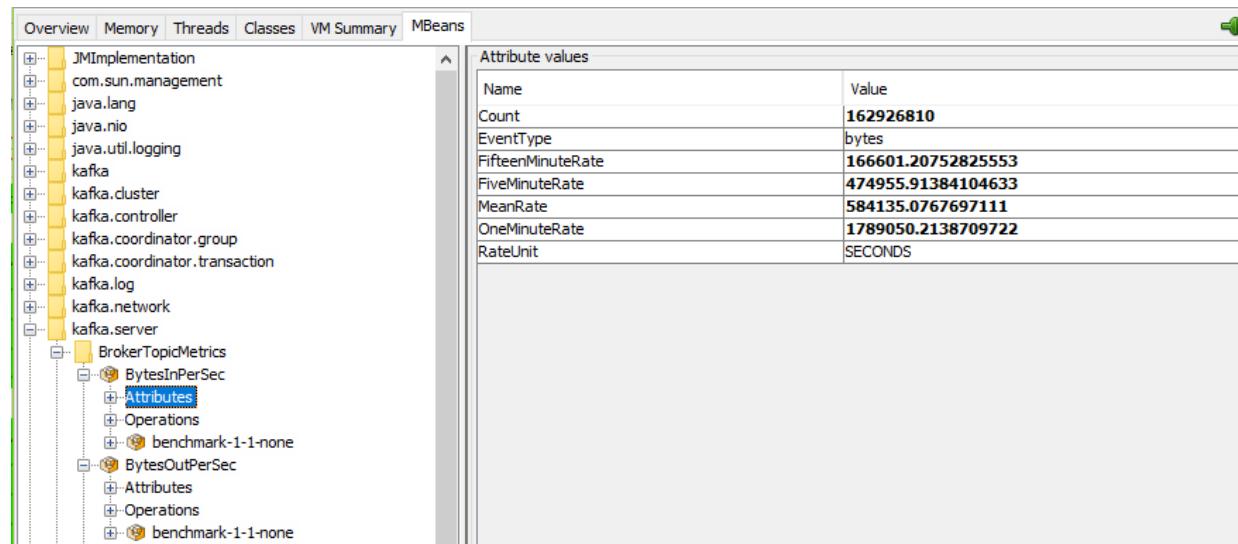
This is because in an effort to increase availability and durability, version 0.8 introduced intra-cluster replication support, and by default a producer waits for an acknowledgement

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:



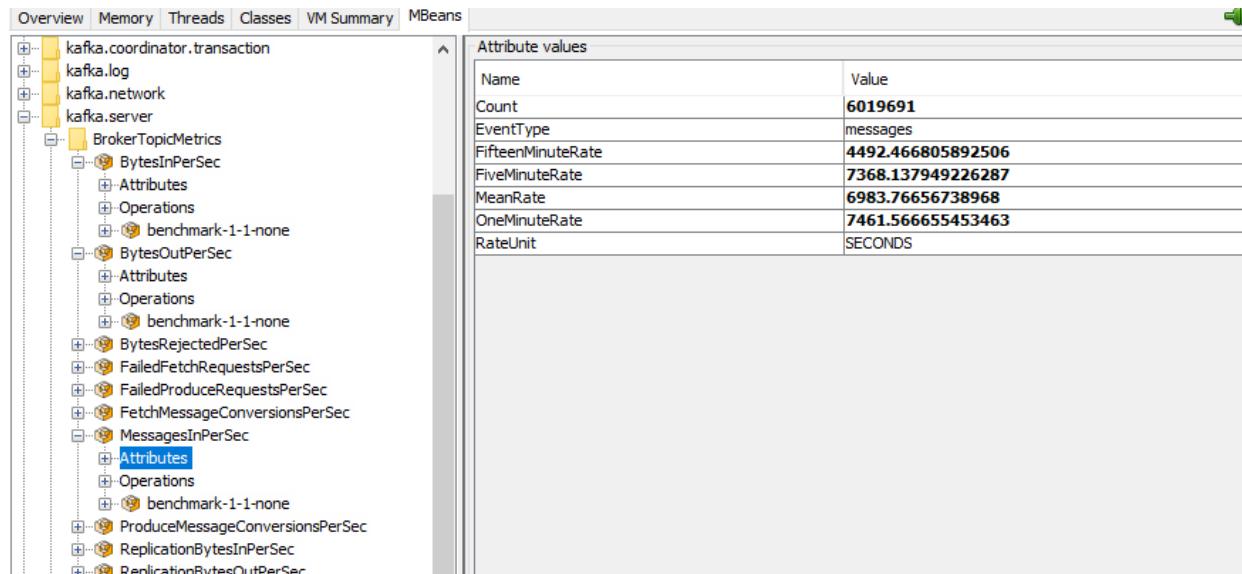
```
## Producer Throughput: 3 producer thread, no replication, no compression

/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-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  
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: xBpQBcWQfaDtHf4ZXoxDQ PartitionCount: 12    ReplicationFactor: 1    Configs: segment.bytes=10737
41824,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 -----

10. Errors

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

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

Solutions: /opt/kafka/config/server.properties

Update the following information.

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

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

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

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

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

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

11. 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>