Hardware:

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

Software Inventory:

- Zookeeper Version: apache-zookeeper-3.8.0-bin.tar
- Apache kafka : 2.13-3.2.1

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

Last Updated: June 29 - 2023.

1. Prerequisites:

Using docker:
Steps to be performed:

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

Execute the following command, it will perform the following:

- Create a network : spark-net
- download the image, centos:8
- start a container with the name, kafka0 and mount host folder in /opt

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

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

Note: /Users/henrypotsangbam/Documents/Software replace this directory with that of your system. Any folder to store software. E.x /tmp

You can verify the container from a separate terminal:

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

Perform the installation after this as a normal server.

Ends Here --------

Software:

- kafka_2.13-3.2.1.tgz
- jdk 11.0.1
- apache-zookeeper-3.6.3-bin.tar.gz
- confluent-7.2.2.tar

2. Installation of Kafka – 90 Mins

Start kafka0 node and perform the following.
You need to install java before installing zookeeper and Kafka.

Installation of Java.
Extract jdk in /opt folder. Required software should be in /Software folder. Ensure to enter the appropriate jdk file of your system. Rename the jdk folder to jdk

```
#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
```

Save the file and exit.
Update the shell scripts using the following command.

```
#bash
```

Install Zookeeper

You can choose any of the options given below:

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

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

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

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

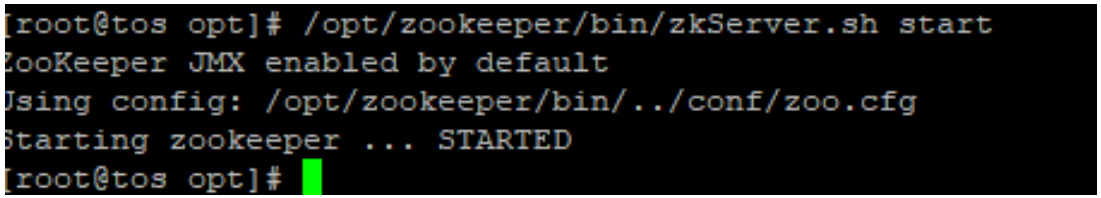
Create a zookeeper configuration file and update with the following values.

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

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

Start the zookeeper using the following scripts.

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



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

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

Option II Ends Here.

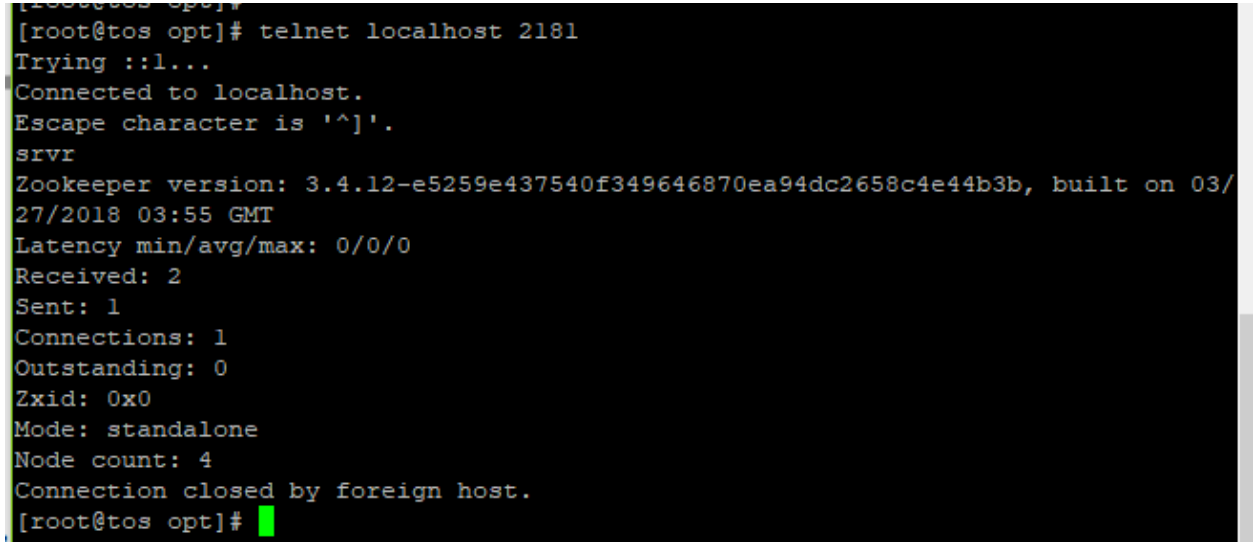
You can now validate that Zookeeper is running correctly in standalone mode by connecting to the client port and sending the four-letter command srvr:

```
#yum install telnet

#telnet localhost 2181
```

Enter the following in the zookeeper CLI.

```
srvr
```



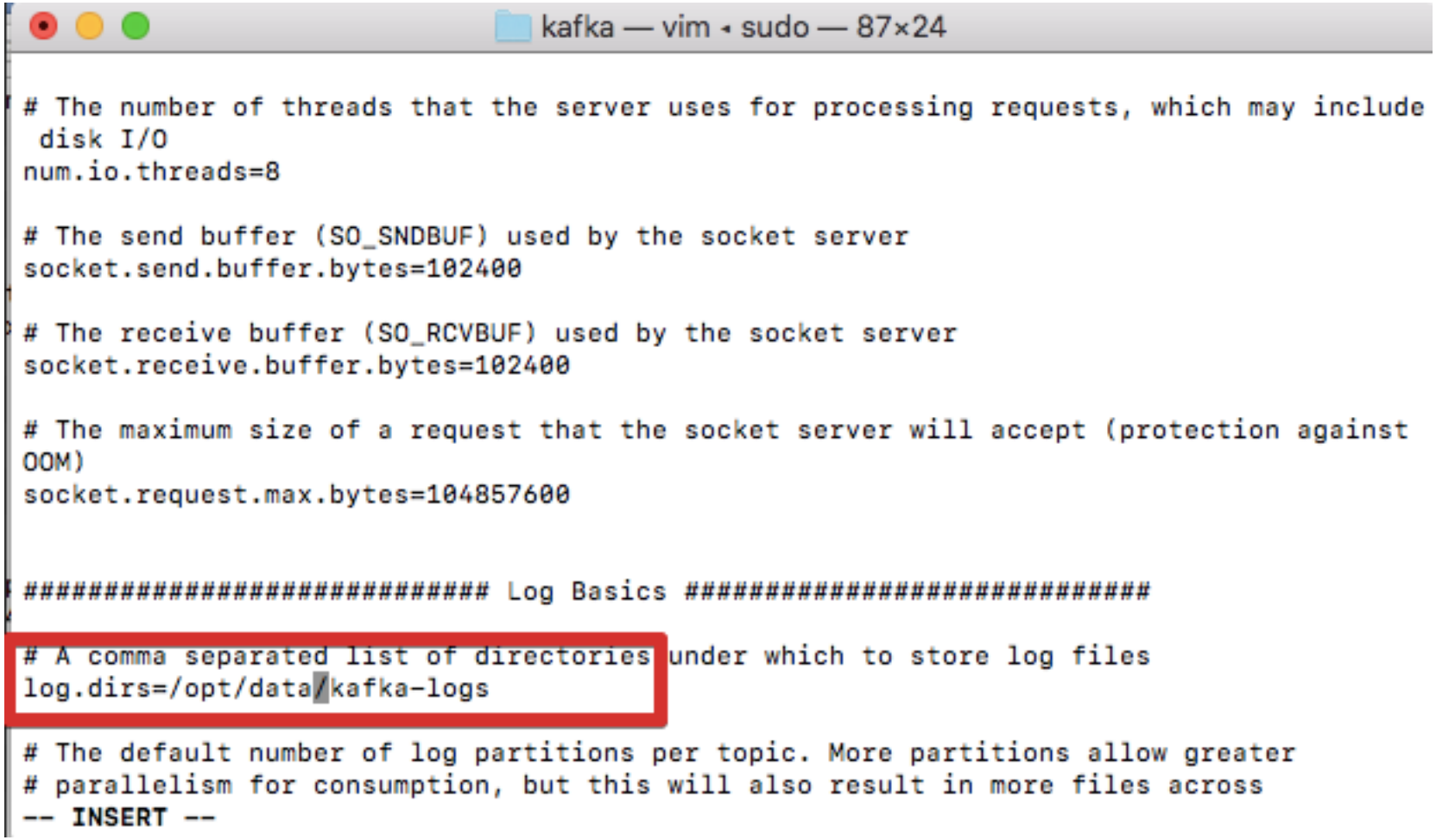
After zookeeper installation, let us install the Kafka Broker.

Installation of Kafka Broker

The following example installs Kafka in /opt/kafka, configured to use the Zookeeper server started previously and to store the message log segments stored in /tmp/kafka-logs: You need to execute the following command from the folder where the software is located.

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

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

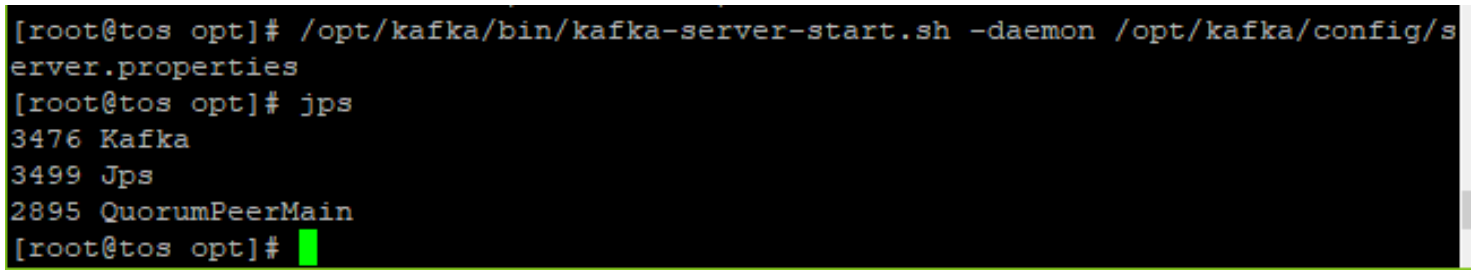


Back Up the configuration for later.

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

Start the broker with the following command

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




```
#mkdir /opt/scripts
```

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

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

Enter the following in the above file.

```
############################## 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 script:

```
#vi /opt/scripts/stopBroker.sh
```

with the following commands in it.

```
#!/usr/bin/env bash
/opt/kafka/bin/kafka-server-stop.sh
```

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

```
#vi /opt/scripts/stopZookeeper.sh
```

Update the following commands in the above script and save it.

```
#!/usr/bin/env bash
# Stop Zookeeper.
/opt/zookeeper/bin/zkServer.sh stop
```

```
echo "Stop zookeeper Successfully"
```

To reinitialize the Cluster.

```
#vi reinitializeCluster.sh

#!/usr/bin/env bash
rm -fr /opt/kafka/data/kafka-logs/*
rm -fr /opt/kafka/data/zookeeper/*
cp /opt/kafka/config/server.properties_plain /opt/kafka/config/server.properties
echo "Reinitialize Successfully"
```

Lab Installation completes End here.

3. Installation Confluent Kafka (Local) – 30 Minutes

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

Install Confluent kafka with the following step.

Inflate the confluent kafka compress file as shown below. Software should be in /Software

```
#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:

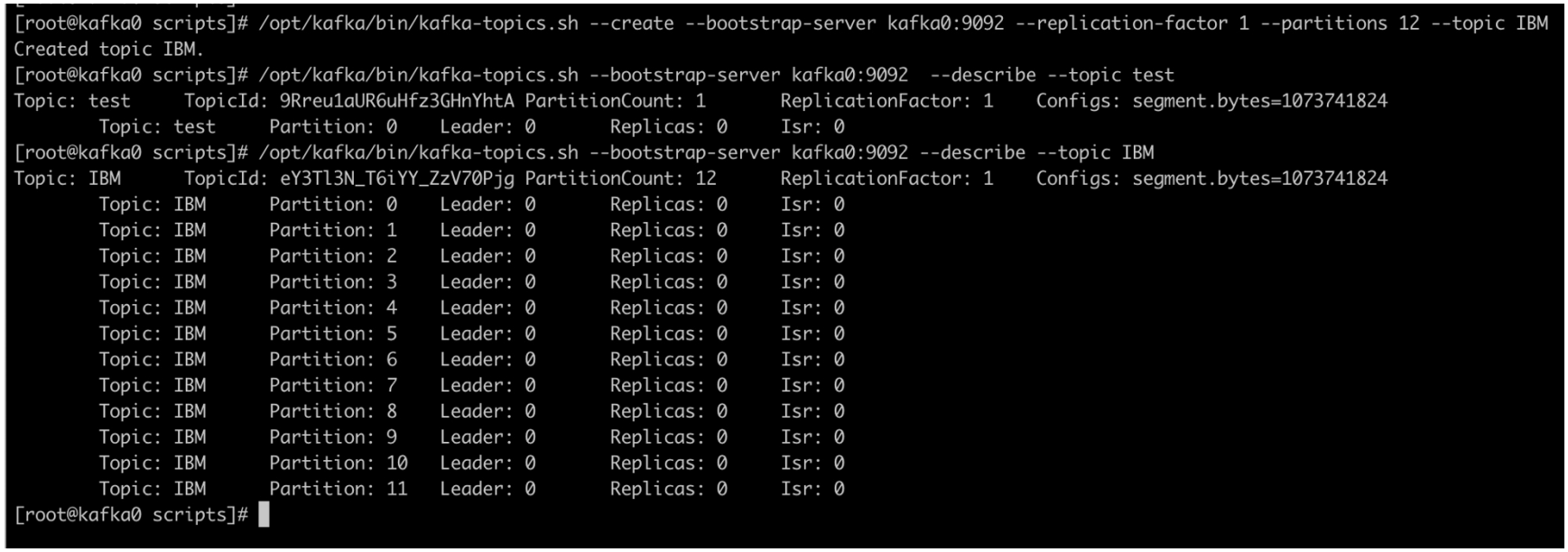
/opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic test

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 1 --partitions 1 --topic test
Created topic test.
```

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

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

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



Verify the no of partition in the output above.

list and describe topic.

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

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --bootstrap-server kafka0:9092  --list
IBM
test
[root@kafka0 scripts]#
```

List and describe Topics

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

How to use the tool?

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

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

List all topics (prints only topic names)

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

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092 --topic test
test
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --list --bootstrap-server kafka0:9092
IBM
test
[root@kafka0 scripts]#
```

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

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

Describe all topics (prints details about the topics)

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

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092 --topic test
Topic: test     TopicId: 9Rreu1aUR6uHfz3GHnYhtA PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: test     Partition: 0    Leader: 0       Replicas: 0     Isr: 0
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --describe --bootstrap-server kafka0:9092
Topic: test     TopicId: 9Rreu1aUR6uHfz3GHnYhtA PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: test     Partition: 0    Leader: 0       Replicas: 0     Isr: 0
Topic: IBM      TopicId: eY3Tl3N_T6iYY_ZzV70Pjg PartitionCount: 12      ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: IBM      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 7    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 8    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 9    Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 10   Leader: 0       Replicas: 0     Isr: 0
        Topic: IBM      Partition: 11   Leader: 0       Replicas: 0     Isr: 0
[root@kafka0 scripts]#
```

We will understand the output in details later.

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

How to use the tool?

create topic with default settings

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

```
[root@tos scripts]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic to
pic1 --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 kafka0:9092 --topic topic2 --partitions 2 --replication-factor 2
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --topic topic1 --partitions 2 --replication-factor 1
Created topic topic1.
[root@kafka0 scripts]# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --topic topic2 --partitions 2 --replication-factor 2
Error while executing topic command : Replication factor: 2 larger than available brokers: 1.
[2023-07-03 15:53:18,020] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 2 larger than available brokers: 1.
 (kafka.admin.TopicCommand$)
[root@kafka0 scripts]#
```

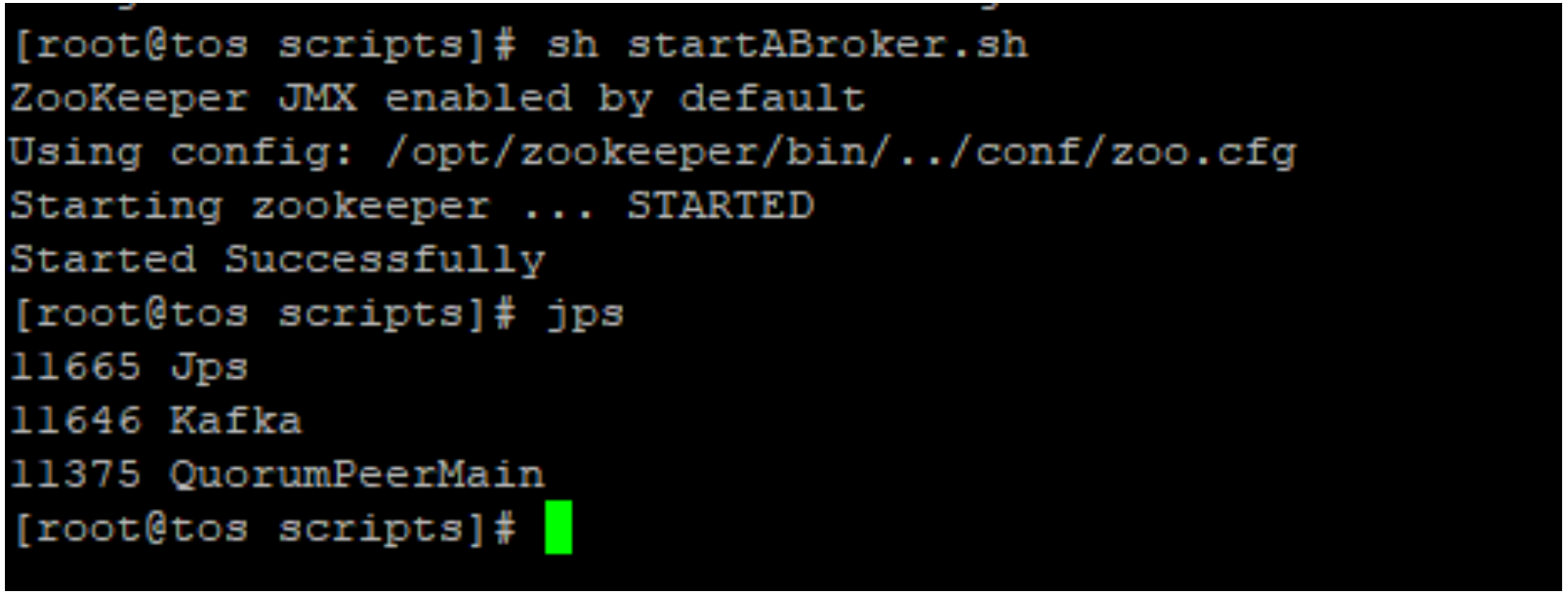
As shown above, it generates an error. Since there is only a single broker. It will fix later.

----------Lab CLI completes End here----------

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

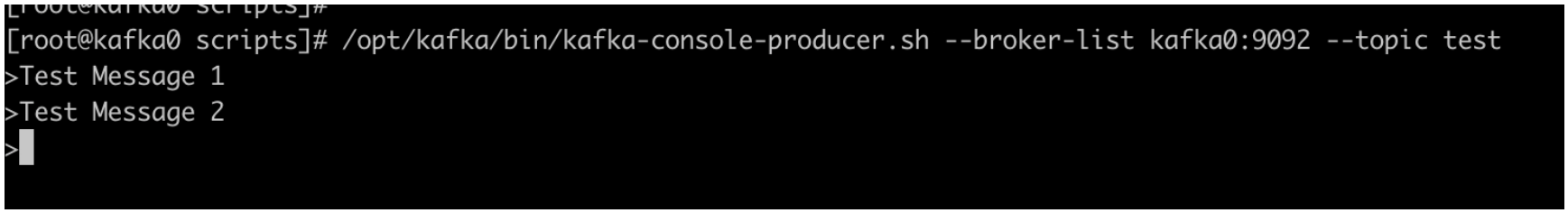
In this lab we will send message to the broker and consumer message using the kafka inbuilt commands.
You need to complete the previous lab before proceeding ahead.
You need to start the broker using startABroker.sh if not done earlier. The script should be in /opt/scripts folder

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



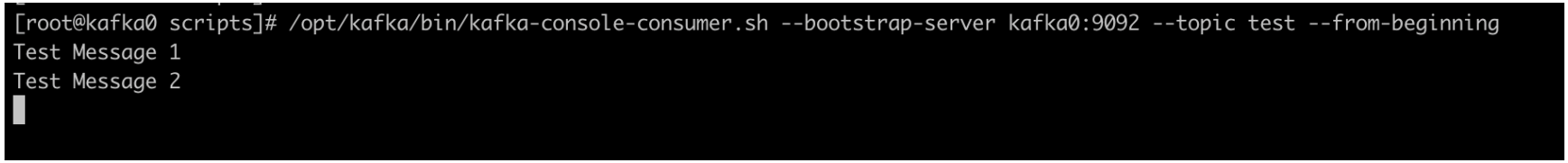
Sent message to **test** topic: Open a console to send message to the topic, test. Enter some text as shown below.

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



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

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



DumplogSegment

You can also view the message publish to a topic from a log file.

```
/opt/kafka/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files \
/opt/data/kafka-logs/test-0/00000000000000000000.log | head -n 12
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files /opt/data/kafka-logs/test-0/00
00000000000000000000.log | head -n 12
Dumping /opt/data/kafka-logs/test-0/00000000000000000000.log
Starting offset: 0
baseOffset: 0 lastOffset: 0 count: 1 baseSequence: 0 lastSequence: 0 producerId: 0 producerEpoch: 0 partitionLeaderEpoch: 0 isTransactional: false isControl:
 false deleteHorizonMs: OptionalLong.empty position: 0 CreateTime: 1688379972311 size: 82 magic: 2 compresscodec: none crc: 1391971847 isvalid: true
| offset: 0 CreateTime: 1688379972311 keySize: -1 valueSize: 14 sequence: 0 headerKeys: [] payload: Test Message 1
baseOffset: 1 lastOffset: 1 count: 1 baseSequence: 1 lastSequence: 1 producerId: 0 producerEpoch: 0 partitionLeaderEpoch: 0 isTransactional: false isControl:
 false deleteHorizonMs: OptionalLong.empty position: 82 CreateTime: 1688379975484 size: 82 magic: 2 compresscodec: none crc: 2251353974 isvalid: true
| offset: 1 CreateTime: 1688379975484 keySize: -1 valueSize: 14 sequence: 1 headerKeys: [] payload: Test Message 2
[root@kafka0 scripts]#
```

Determine the offset of the messages.
CLI to list offset number.

```
#/opt/kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list kafka0:9092 --topic test
```

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list kafka0:9092 --topic test
test:0:2
[root@kafka0 scripts]#
```

Partition 0 offset 2.

Sending Message with Key and Value

```
#/opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic IBM --property "parse.key=true" --property "key.separator=:"
```

Enter
WAS:600
DB2:700
ESB:900

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic IBM --property "parse.key=true" --property "key.separator=:"
>WAS:600
>DB2:700
>ESB:900
>
```

```
#/opt/kafka/bin/kafka-console-consumer.sh --topic IBM --bootstrap-server kafka0:9092 --from-beginning \
 --property print.key=true \
 --property key.separator=":"
```

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --topic IBM --bootstrap-server kafka0:9092 --from-beginning \
>  --property print.key=true \
>  --property key.separator=":"
WAS:600
ESB:900
DB2:700
```

Let us understand consumer group load balancing concepts.

- Create a topic with 2 partitions – city_info.
- Initiate 2 consumers with same consumer Group
- Send 6 messages to the topic
- Verify load get distributed among the 2 consumers.

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

Initiate 2 consumers with same consumer Group
Open in a separate 2 terminals.

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic
city_info   --consumer-property group.id=citygroup
```

```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9092 --topic
city_info   --consumer-property group.id=citygroup
```

- Produce about 6 messages to the topic

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

```
[root@kafka0 scripts]# /opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092 --topic city_info
>Imphal is scenic
>Delhi is classic
>Mumbai is cosmopolitian
>Bangalore is cool
>Dubai is warm
>Bangkok is wow
>
```

- load get distributed among the 2 consumers.

```
@kafka0:/ (com.docker.cli)
Last login: Tue Jul  4 08:42:24 on ttys000
henrypotsangbam@Henrys-MacBook-Pro ~ % docker exec -it kafka0 bash
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-
server kafka0:9092 --topic city_info    --consumer-property group.id=ci
tygroup
Delhi is classic
Bangalore is cool
Bangkok is wow
```

```
@kafka0:/ (com.docker.cli)
Last login: Tue Jul  4 08:54:34 on ttys001
henrypotsangbam@Henrys-MacBook-Pro ~ % docker exec -it kafka0 bash
[root@kafka0 /]# clear
[root@kafka0 /]#
[root@kafka0 /]#
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstra
p-server kafka0:9092 --topic city_info    --consumer-property group.i
d=citygroup
Imphal is scenic
Mumbai is cosmopolitian
Dubai is warm
```

----------Lab CLI ends here----------

6. Kafka cluster – 90 Minutes

In this tutorial, we are going to run many Kafka Nodes on our system to Understand Kafka Failover in a cluster environment. So, you will need at least 8 GB of RAM in your system. 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 group cordination along with broker failover. We will 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 perform the following steps:

- copy the existing Kafka server.properties to server-0.properties, server-1.properties, and server-2.properties.
- Update server-0.properties to set log.dirs to "/opt/data/kafka-logs/kafka-0.
- 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 - kafka-config/config.

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

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

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

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

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

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

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

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

Create Start-up scripts for these three Kafka servers

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

```
#vi /opt/scripts/startB0.sh
```

Copy the following in the above file.

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

```
#vi /opt/scripts/startB1.sh
#!/usr/bin/env bash
## Run Kafka
/opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-
1.properties"
```

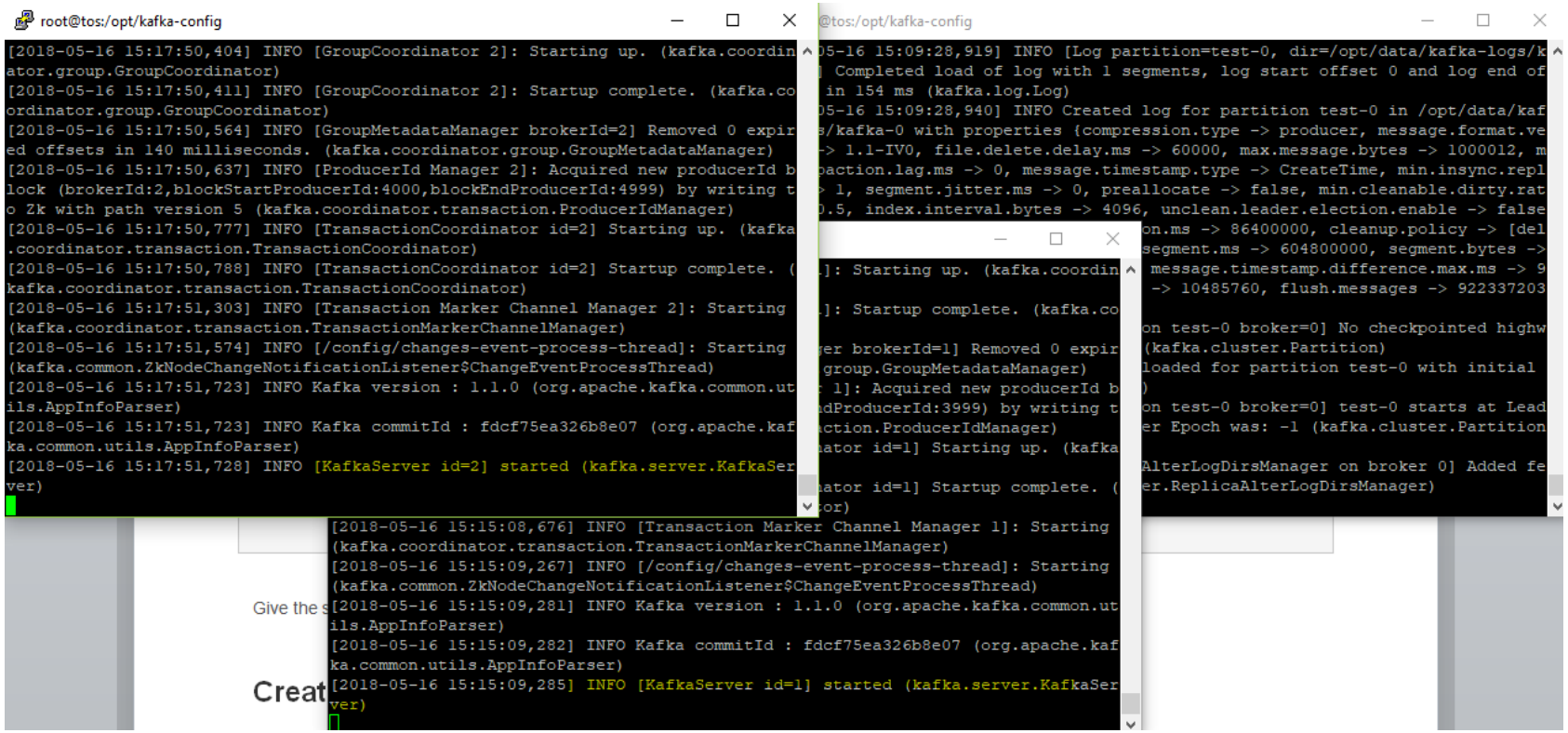
```
#vi /opt/scripts/startB2.sh
      #!/usr/bin/env bash
      ## Run Kafka
      /opt/kafka/bin/kafka-server-start.sh   "/opt/kafka-config/config/server-
      2.properties"
```

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 seperate terminal from /opt/kafka-config

```
#cd /opt/scripts
// to make the scripts executable.
#chmod 755 startB0.sh  startB1.sh startB2.sh
$ ./startB0.sh
$ ./startB1.sh
$ ./startB2.sh
```

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



You should have as many terminal as shown above at this point.

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 3  --partitions 13 --topic my-failsafe-topic
```

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --create --bootstrap-server kafka0:9092 --replication-factor 3  --partitions 13 --topic m
y-failsafe-topic
Created topic my-failsafe-topic.
[root@kafka0 /]#
```

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
```

```
[root@kafka0 /]# /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.
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.

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-producer.sh --broker-list kafka0:9092,kafka0:9093 --topic my-failsafe-topic
>How are you, kafka
>Its' working
>
```

Consumer Console

```
[root@kafka0 /]# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server kafka0:9094,kafka0:9092 --topic my-failsafe-topic   --from-beginning
How are you, kafka
Its' working
```

Now Start two more consumers and send more messages.

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

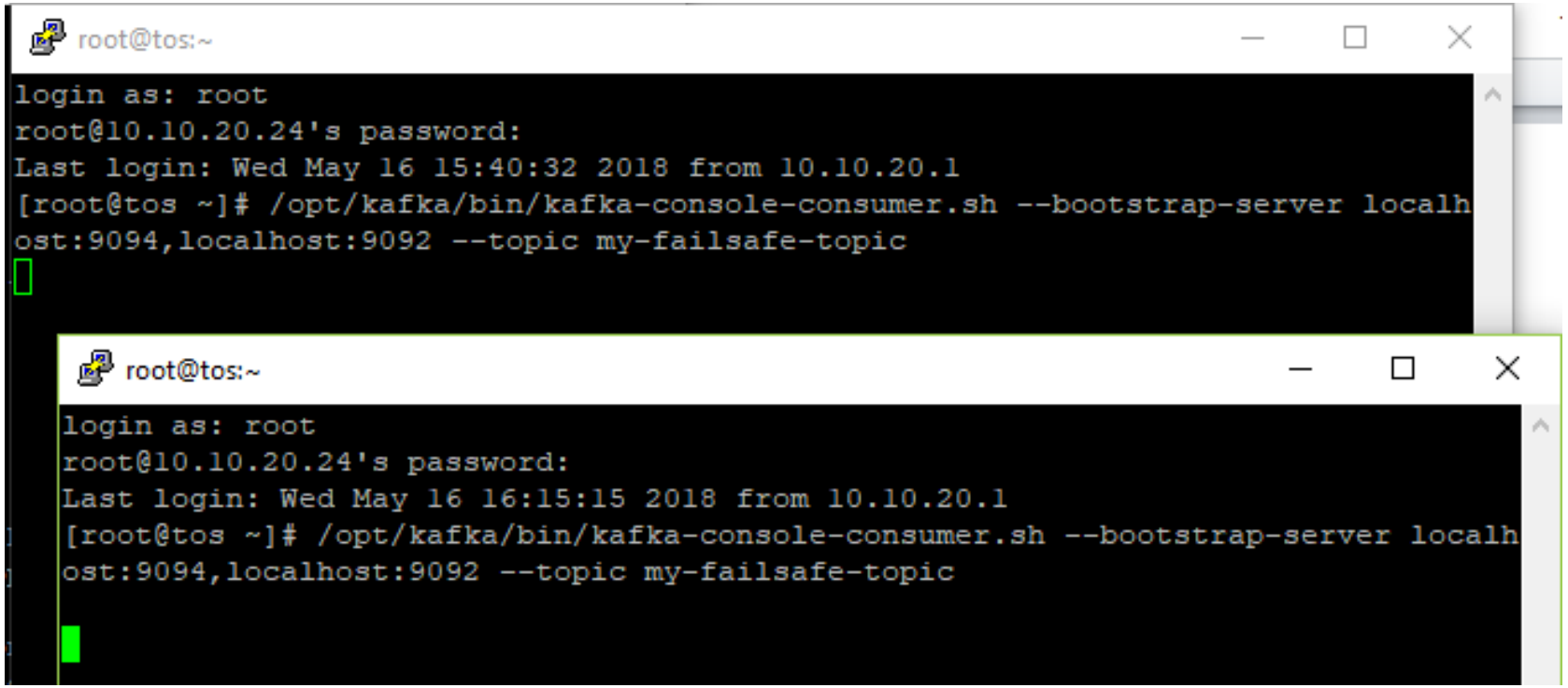
Consumer 1.

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

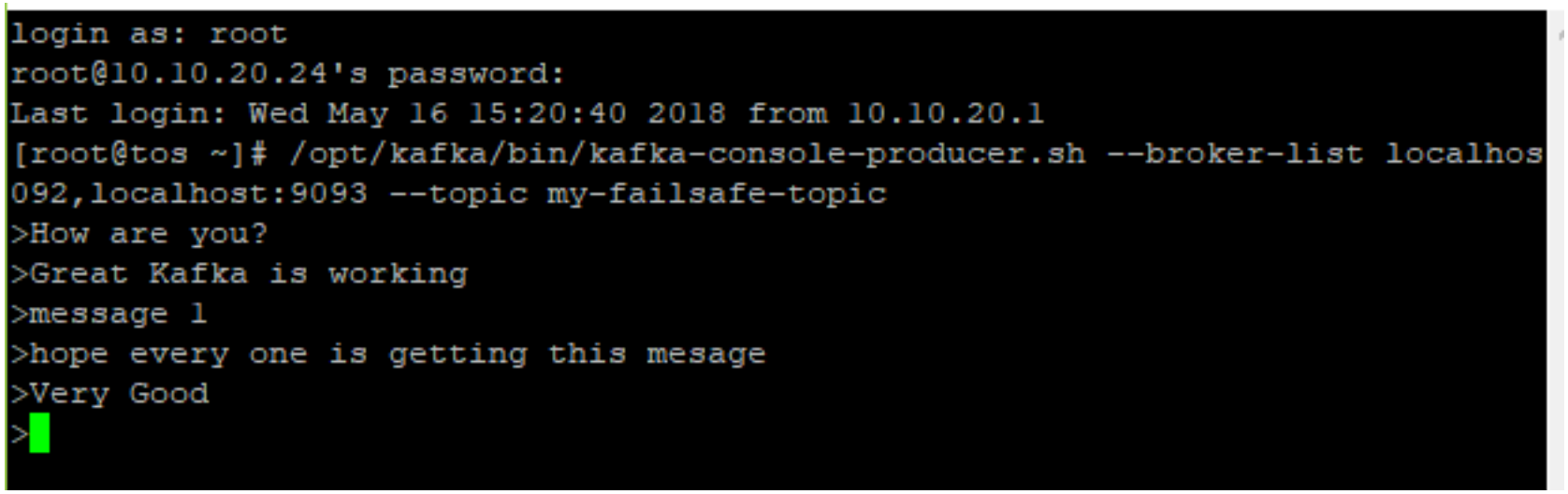
Consumer 2.

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

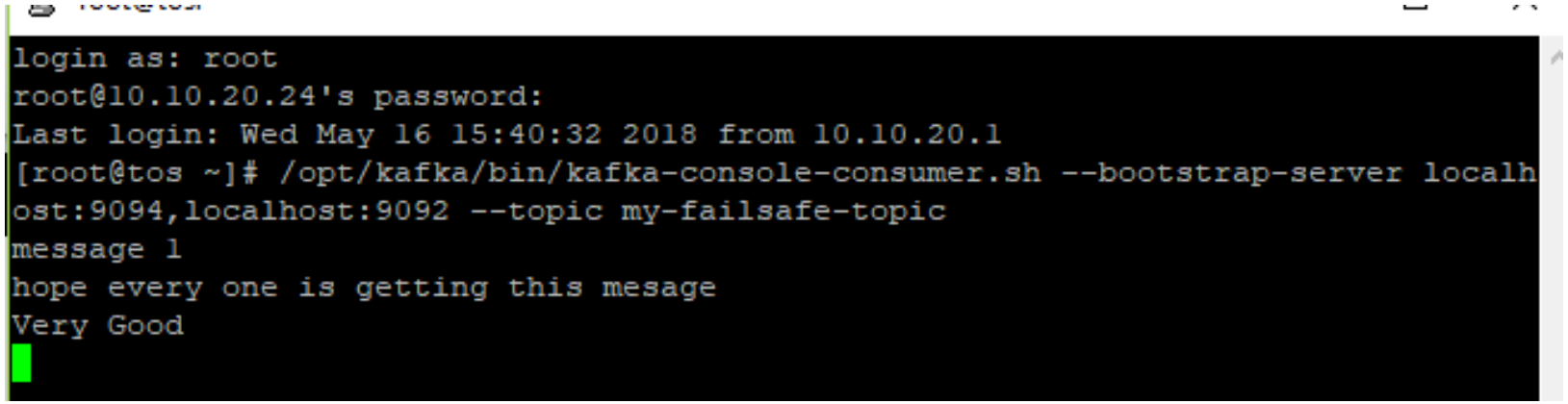
The two consumer consoles will be as shown below



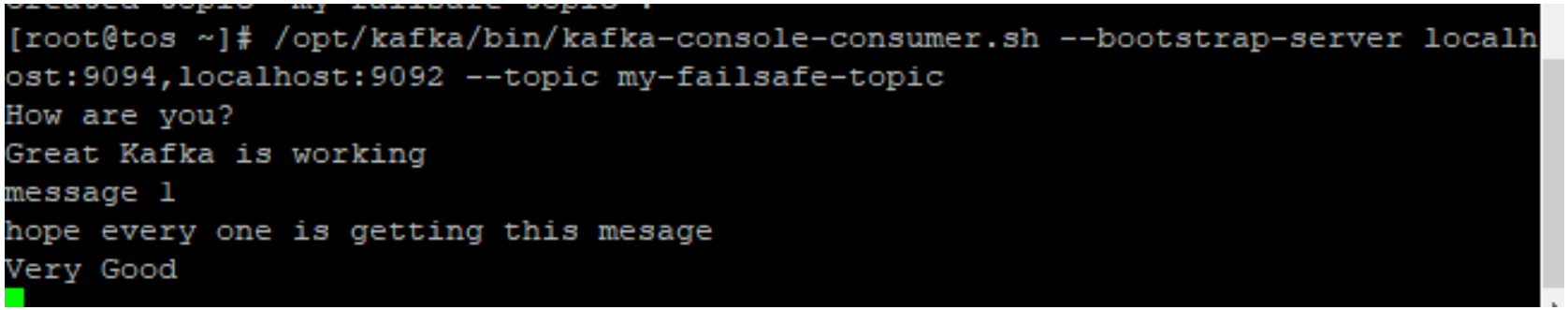
Producer Console will be as shown below:



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



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



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

Change consumer to be in their own consumer group.

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

Run the following scripts three times – from different console.

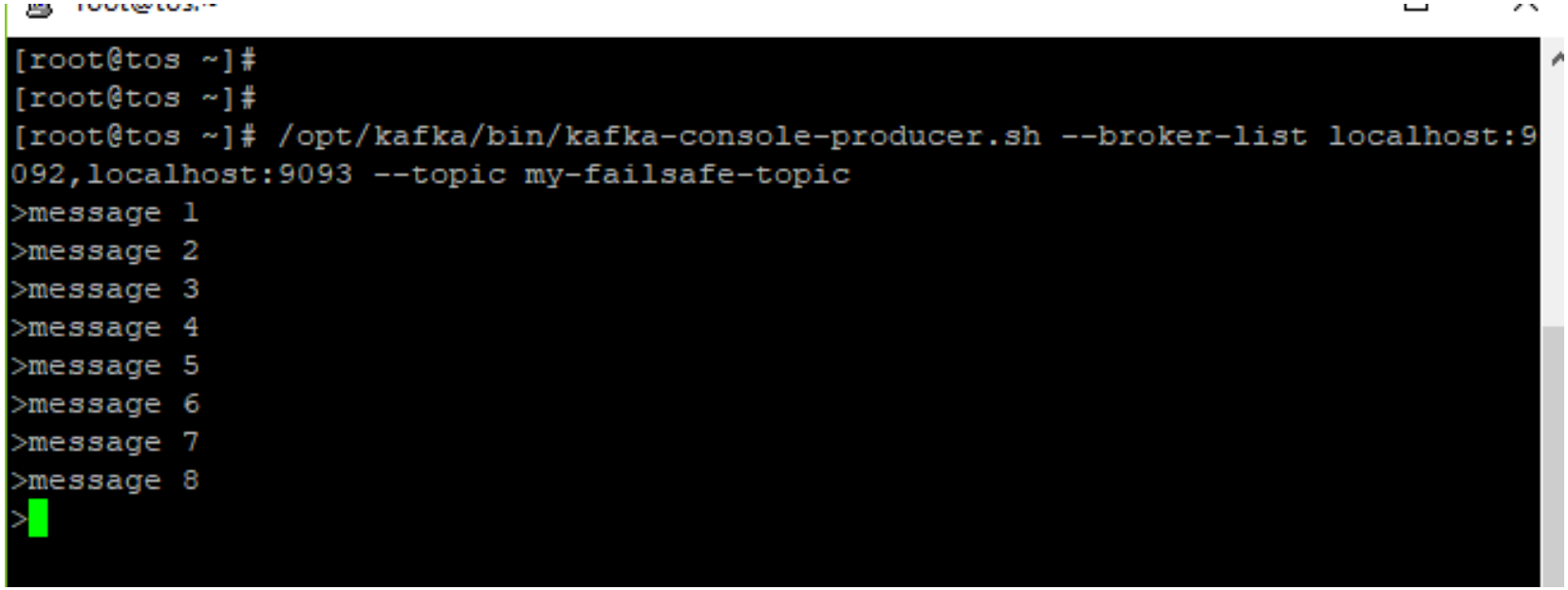
```
#/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server
kafka0:9094,kafka0:9092 --topic my-failsafe-topic   --consumer-property
group.id=mygroup
```

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

Run Producer Console

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

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



Notice that the messages are spread evenly among the consumers.

1st Kafka Consumer gets m3, m5

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

Notice the first consumer gets messages m3 and m5.

2nd Kafka Consumer gets m2, m6

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

Notice the second consumer gets messages m2 and m6.

3rd Kafka Consumer gets m1, m4, m7

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

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

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

Kafka Consumer Failover

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

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

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

Producer Console - send seven more messages m8 through m14

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

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

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

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

message 9
m 12
m 13
m 14
```

2nd Kafka Consumer gets m10, m12, m13

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

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

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

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

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

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

Run describe-topics again to see leadership change

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

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

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

7. 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.

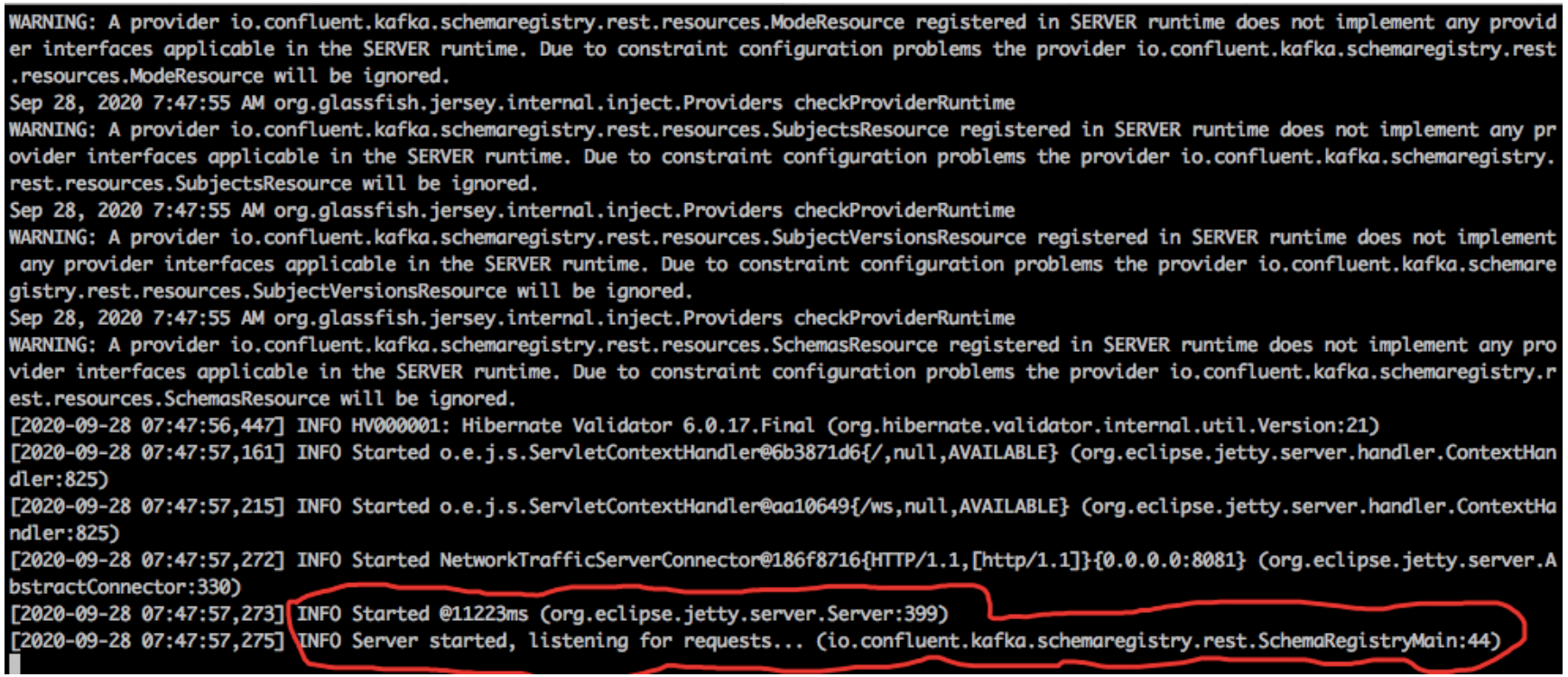
Update the following in

```
#vi /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
```



Create a topic and let us bind schema to this.

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

Use the Schema Registry API to add a schema for the topic my-kafka. Subject is my-kafka-value, means the value schema of 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\"},{\"na
me\":\"amount\",\"type\":\"double\"}]}"}[root@kafka0 code]#
```

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

Update: It will generate incompatibility schema issue, since there is no default value for the field count in the earlier messages.

#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\"},{\"name\":\"count\",\"type\":\"double\"}]}"}' http://localhost:8081/subjects/my-kafka-value/versions

```
[root@kafka0 scripts]# curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema": "{\"t
ype\":\"record\",\"name\":\"Payment\",\"namespace\":\"com.tos\",\"fields\":[{\"name\":\"id\",\"type\":\"string\"},{\"n
ame\":\"amount\",\"type\":\"double\"},{\"name\":\"count\",\"type\":\"double\"}]}"}' http://localhost:8081/subjects/my-
kafka-value/versions
{"error_code":409,"message":"Schema being registered is incompatible with an earlier schema for subject \"my-kafka-val
ue\", details: [Incompatibility{type:READER_FIELD_MISSING_DEFAULT_VALUE, location:/fields/2, message:count, reader:{\"
type\":\"record\",\"name\":\"Payment\",\"namespace\":\"com.tos\",\"fields\":[{\"name\":\"id\",\"type\":\"string\"},{\"
name\":\"amount\",\"type\":\"double\"},{\"name\":\"count\",\"type\":\"double\"}]}, writer:{\"type\":\"record\",\"name\
":\"Payment\",\"namespace\":\"com.tos\",\"fields\":[{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"amount\",\"type\
":\"double\"}]}}] io.confluent.kafka.schemaregistry.rest.exceptions.RestIncompatibleSchemaException: Schema being regi
stered is incompatible with an earlier schema for subject \"my-kafka-value\", details: [Incompatibility{type:READER_FI
ELD_MISSING_DEFAULT_VALUE, location:/fields/2, message:count, reader:{\"type\":\"record\",\"name\":\"Payment\",\"names
pace\":\"com.tos\",\"fields\":[{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"amount\",\"type\":\"double\"},{\"name
\":\"count\",\"type\":\"double\"}]}, writer:{\"type\":\"record\",\"name\":\"Payment\",\"namespace\":\"com.tos\",\"fiel
ds\":[{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"amount\",\"type\":\"double\"}]}}]\nio.confluent.kafka.schemare
gistry.rest.exceptions.RestIncompatibleSchemaException: Schema being registered is incompatible with an earlier schema
 for subject \"my-kafka-value\", details: [Incompatibility{type:READER_FIELD_MISSING_DEFAULT_VALUE, location:/fields/2
, message:count, reader:{\"type\":\"record\",\"name\":\"Payment\",\"namespace\":\"com.tos\",\"fields\":[{\"name\":\"id
```

Add one compatible version of schema.

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

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 ----------

8. Errors

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

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

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

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

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

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

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

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

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

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

9. LOG verification + segment Sizing

10. 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-0/00000000000000000000.log | head -n 4

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

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

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. Settings

Open two separate terminals and execute the following

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

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

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

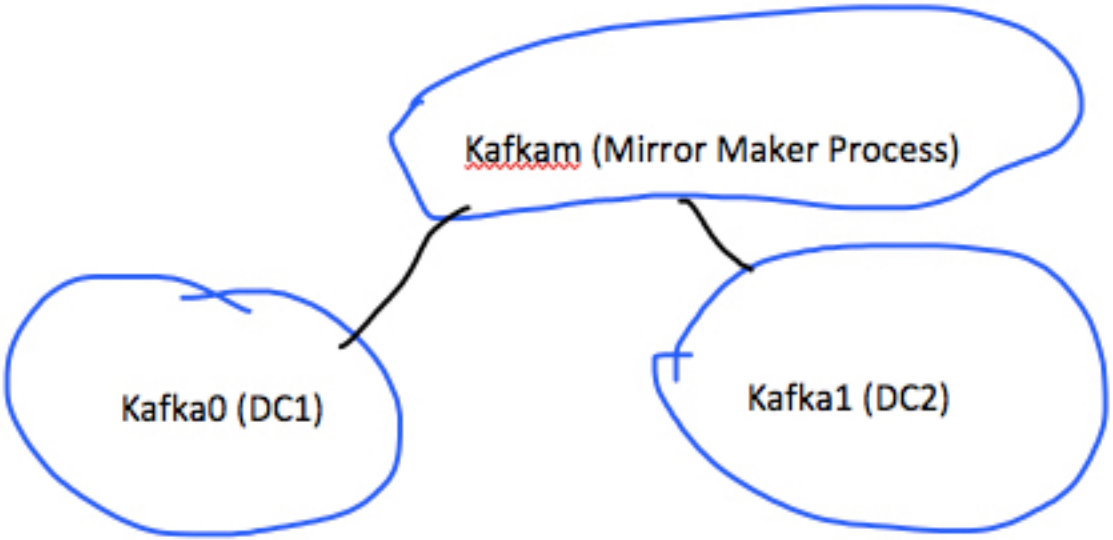
Install the kafka as done before if required or copy the kafka folder in different name to persevere the libraries.

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

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

Kafka Mirror Architecture:

12:46 PM



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

// Changes Begin

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

// Changes End

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

IV. 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