Hardware:

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

Software Inventory:

- Zookeeper Version: apache-zookeeper-3.6.1-bin.tar

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

Last Updated: Feb 26 - 2022.

1. Prerequisites:

Option I

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

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

Option II.

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

Execute the following command, it will perform the following:

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

#docker network create --driver bridge spark-net

#docker run --name kafka0 --hostname kafka0 -p 9092:9092 -p 8081:8081 -p 2181:2181 –p 9999:9999 -i -t --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt centos:7 /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 -i -t --privileged --network spark-net -v /Volumes/Samsung_T5/software/:/Software -v /Volumes/Samsung_T5/software/install/:/opt -v /Volumes/Samsung_T5/software/data/:/data centos:7 /usr/sbin/init

You can verify the container from a separate terminal:

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

Perform the installation after this as a normal server.

Update the server.properties.

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

advertised.listeners=PLAINTEXT://localhost:9092

After installation to start the zookeeper and kafka broker:

Open two separate terminals and execute the following

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

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

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

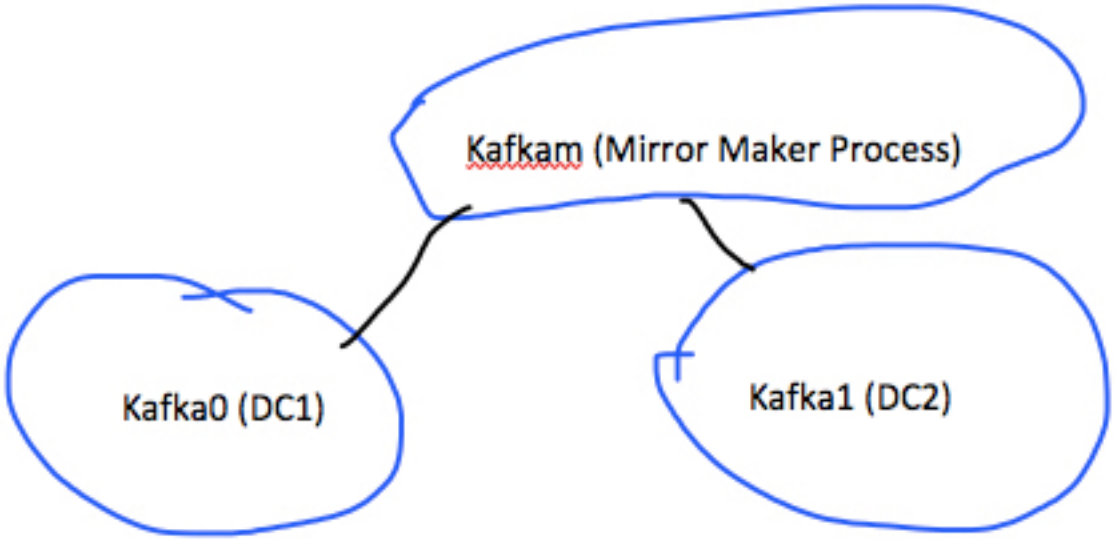
Install the kafka as done before if required or copy the kafka folder in different name to persevere the libraries.

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

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

Kafka Mirror Architecture:

12:46 PM



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

// Changes Begin

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

// Changes End

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

2. Installation of Kafka – 90 Mins

You need to install java before installing zookeeper and Kafka.

Installation of Java
#tar -xvf jdk-8u45-linux-x64.tar.gz -C /opt

Set the above path in the PATH variable and JAVA_HOME
export JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin
Include in the profile as follow:
#vi ~/.bashrc

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

# User specific aliases and functions

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

export JAVA_HOME=/opt/jdk1.8.0_45

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

Update the shell scripts using the following command.
#bash

Installing Zookeeper

You can choose any of the options given below:

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

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

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

```
# tar -xvf apache-zookeeper-3.6.1-bin.tar -C /opt
#mv a*zookeeper-3.6.1* /opt/zookeeper
#mkdir -p /opt/data/zookeeper
```

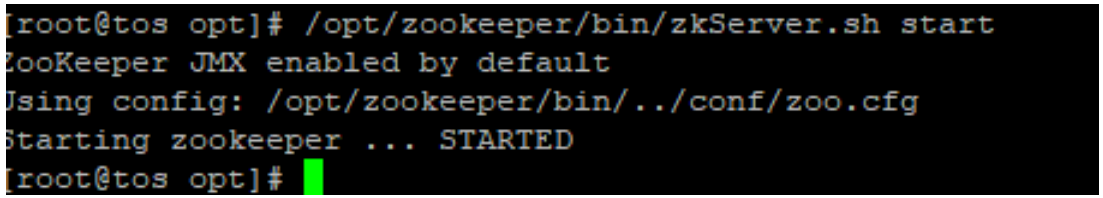
Create a zookeeper configuration file and update with the following values.

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

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

Start the zookeeper using the following scripts.

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



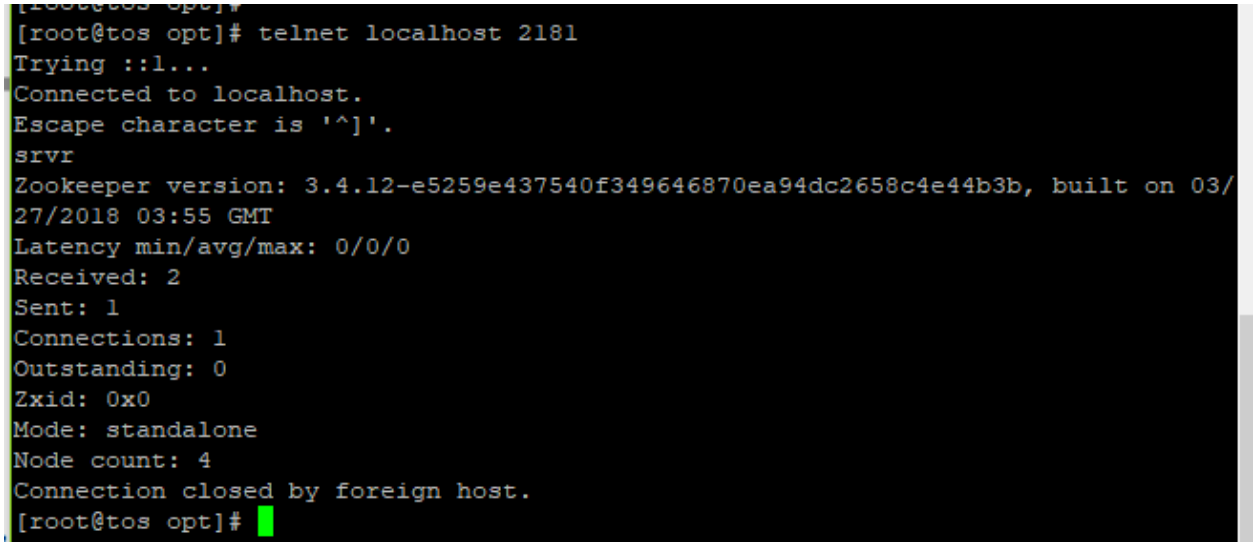
Option II (Part of the Apache Kafka)

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

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

#telnet localhost 2181

- srvr



After zookeeper installation, let us install the Kafka Broker.

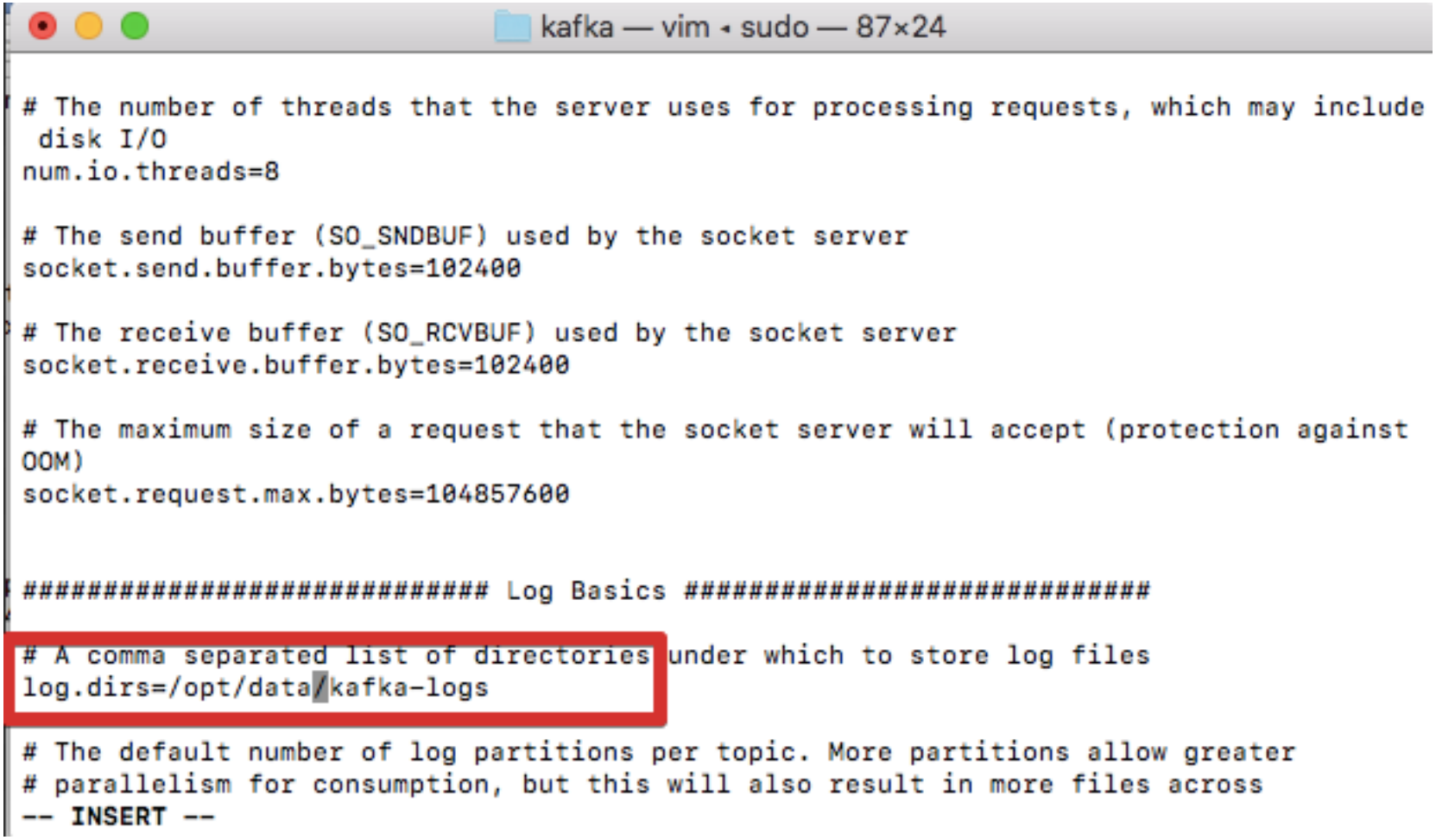
Installation of Kafka Broker

The following example installs Kafka in /opt/kafka, configured to use the

Zookeeper server started previously and to store the message log segments stored in /tmp/kafka-logs:

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

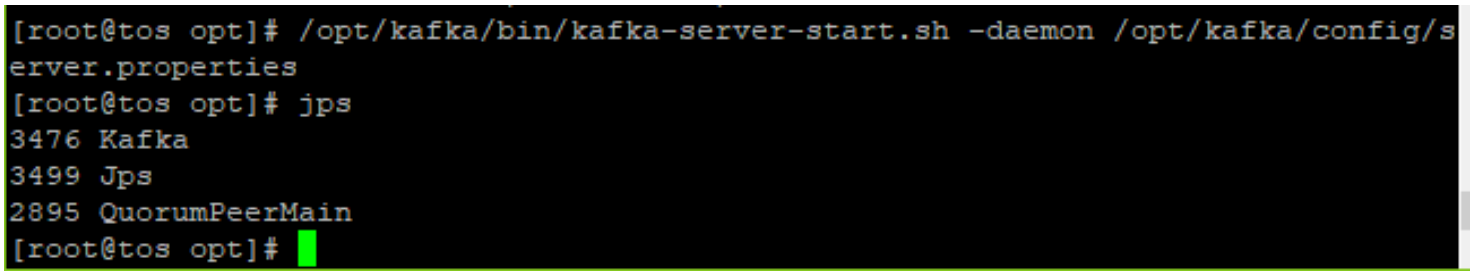
Update the /opt/kafka/config/server.properties to store the Kafka Log in the above mention folder.



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

Start the broker with the following command

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



#mkdir /opt/scripts

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

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

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

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

echo "Started Successfully"

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

To Stop Zookeeper create the following script.
#vi /opt/scripts/stopZookeeper.sh
Update the following commands in the above script and save it.

#!/usr/bin/env bash
Stop Zookeeper.
/opt/zookeeper/bin/zkServer.sh stop
echo "Stop zookeeper Successfully"
--- ---

Lab Installation completes End here.

3. Installation Confluent Kafka (Local) – 30 Minutes

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

Installing Confluent kafka

Inflate the confluent kafka compress file as shown below:

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

Set the environment variable for the Confluent Platform directory.

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

Set your PATH variable:

vi ~/.bashrc

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

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

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

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

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

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

Create and verify details about topic:

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

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

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

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

```
[root@tos opt]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:21
81 --replication-factor 1 --partitions 1 --topic test
Created topic "test".
[root@tos opt]# /opt/kafka/bin/kafka-topics.sh --zookeeper localhost:2181 --desc
ribe --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-topicl
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.b
ytes=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 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 localhost:9092 --topic topic2 --partitions 2 --replication-factor 2

```
[root@kafka0 /]# /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic topic2 --partition
s 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 /]#
```

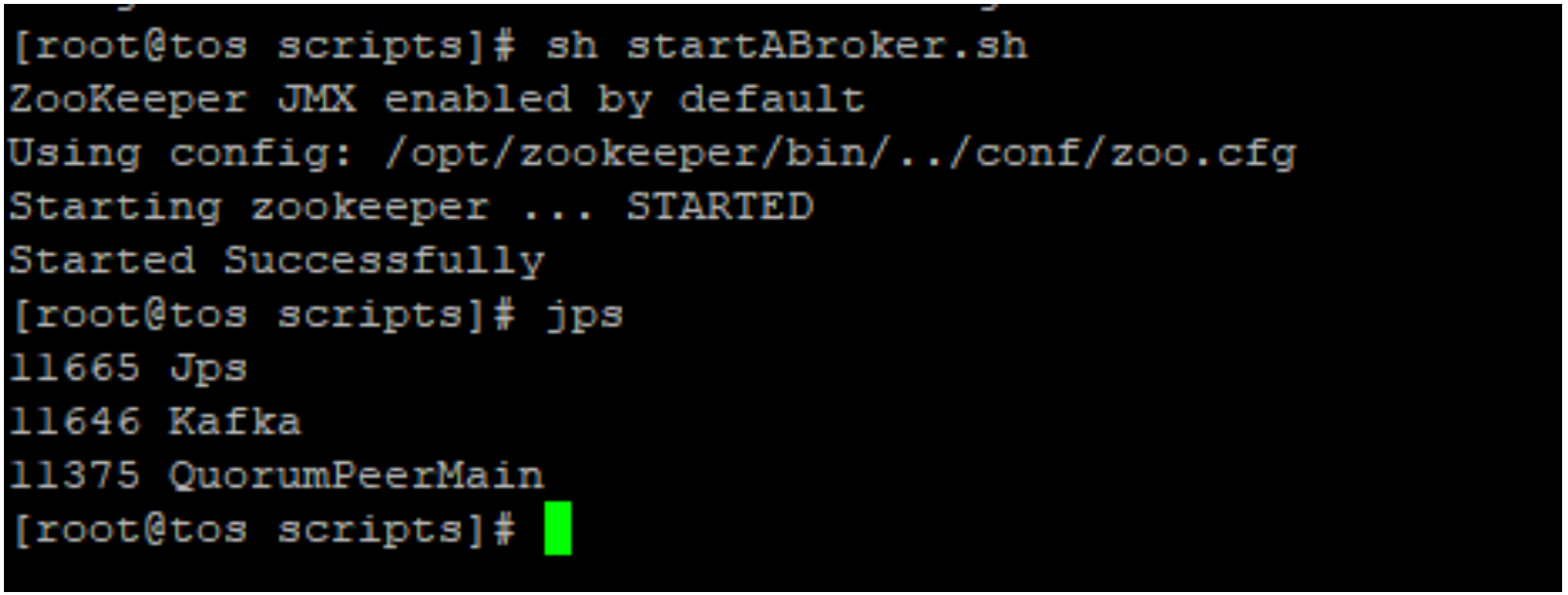
As shown above, it generates an error. Since there is only a single broker. It will fix later.

Lab CLI completes End here.

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

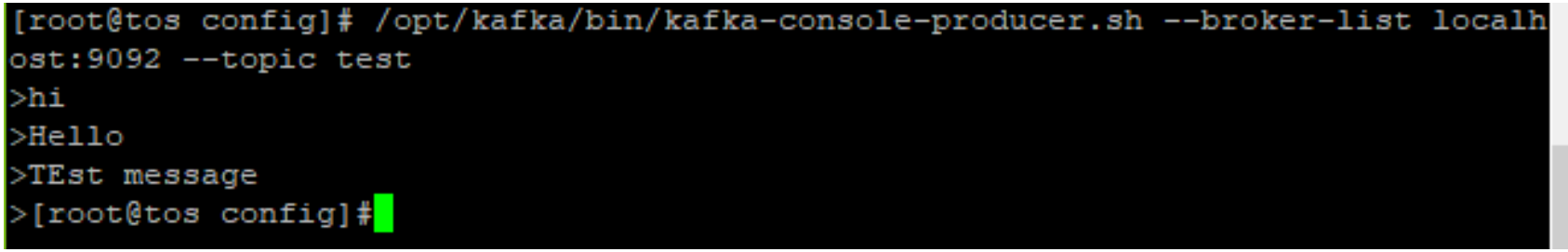
In this lab we will send message to the broker and consumer message using the kafka inbuilt commands.
You need to complete the previous lab before proceeding ahead.
You need to start the broker using startABroker.sh if not done earlier. The script should be in /opt/scripts folder

#sh startABroker.sh
#jps



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
#
```



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.

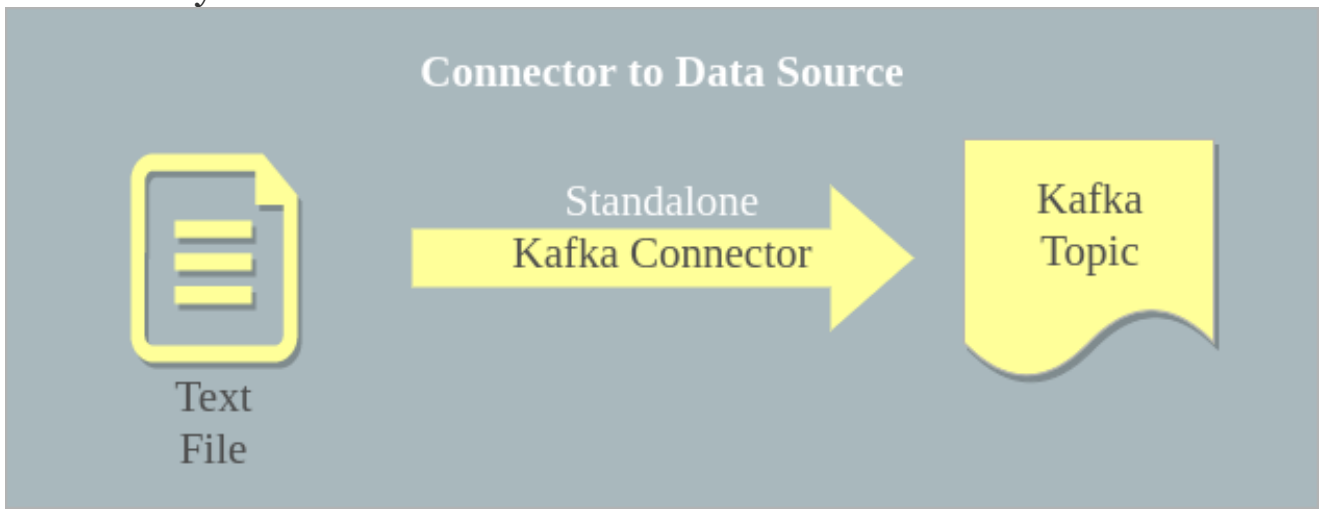
```
# /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test
--from-beginning
```

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

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

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

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

Creating a Source Config File

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

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

Paste the following instruction in the above file.

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

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

Creating a Worker Config File.

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

Create a file.
#vi my-standalone.properties

Update with the following contents.

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

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

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

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

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

Running Kafka Connect.

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

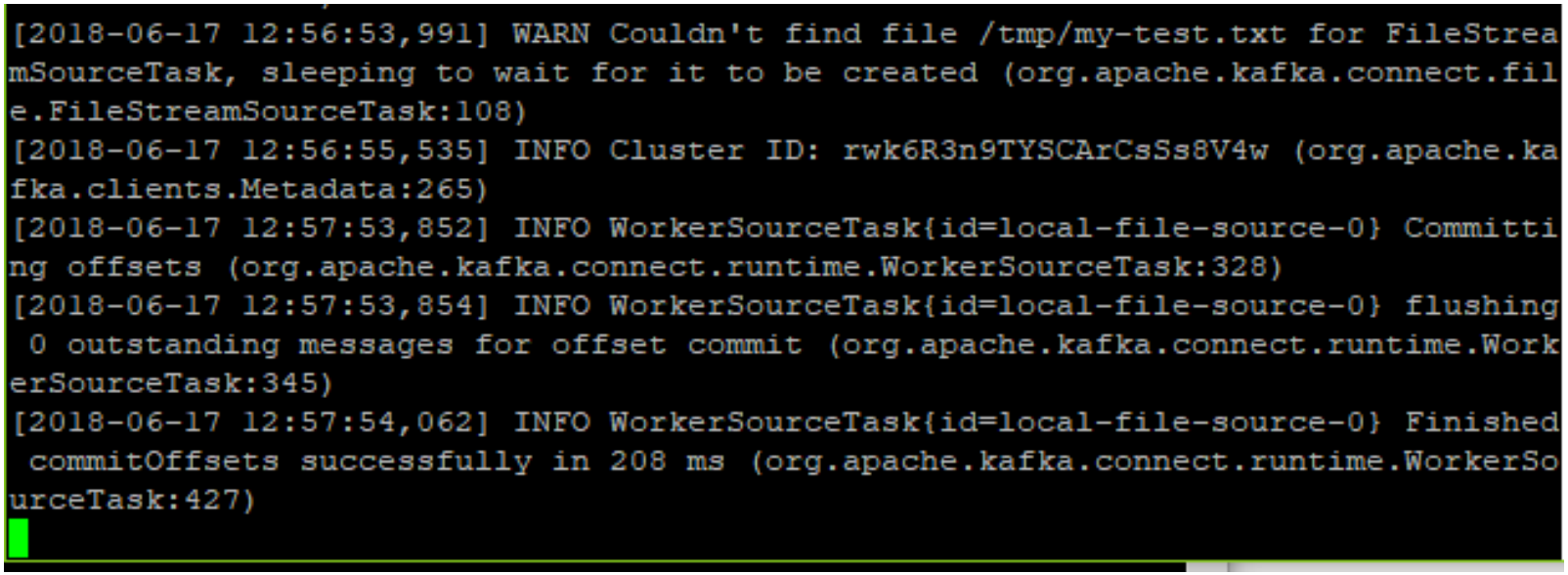
#/opt/kafka/bin/connect-standalone.sh my-standalone.properties my-file-source.properties

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

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.



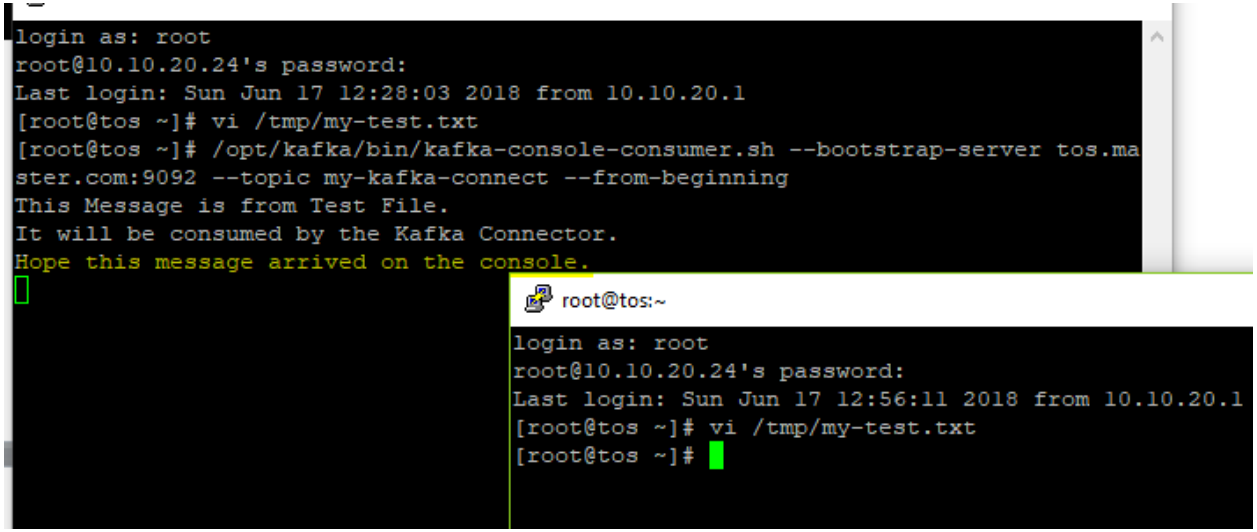
Open another terminal and execute the following consumer to consume the message.

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.ma
ster.com:9092 --topic my-kafka-connect --from-beginning
This Message is from Test File.
It will be consumed by the Kafka Connector.
```

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



Let us try with json file.
Stop the kafka connector – ctrl + C of the terminal in which we have started it.

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

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

Start the Kafka connector now:

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

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

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

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

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

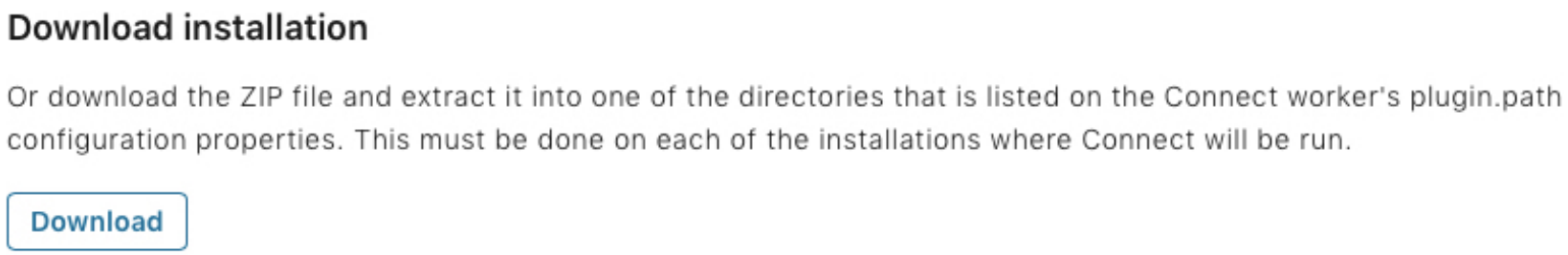
We have successfully configured, file connector in standalone mode.

Next, let us configure JDBC in a distributed mode.

Kafka Connect – JDBC Connector.

Download the connector from the following url.

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



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.

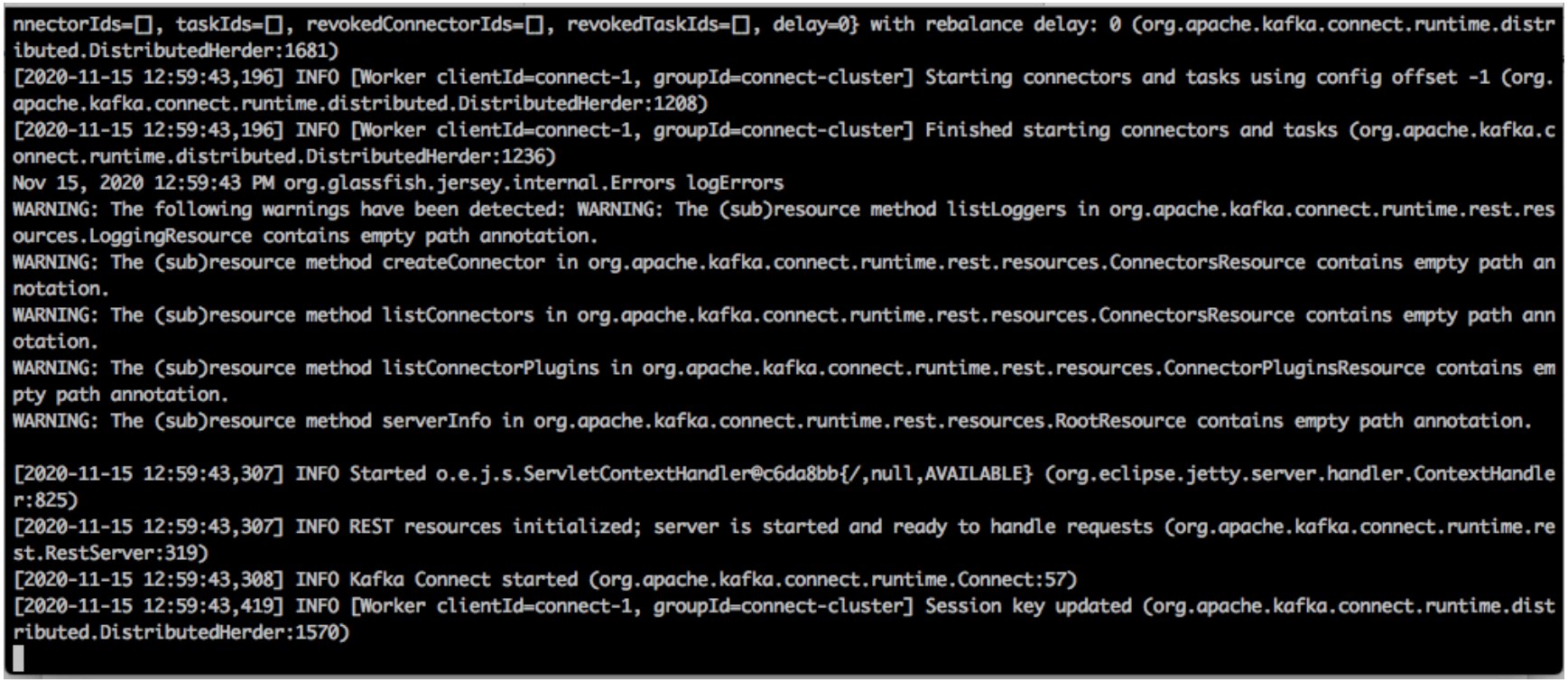
```
# A list of host/port pairs to use for establishing the initial connection to the Kafka
cluster.
bootstrap.servers=localhost:9092

# 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

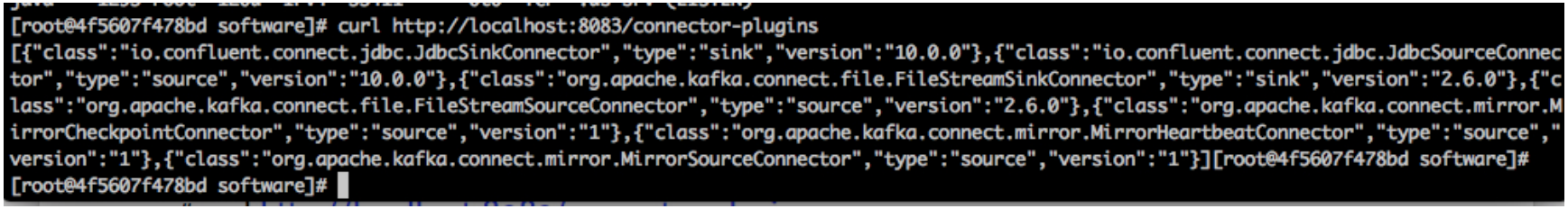
Go to the bin folder of the kafka installation or specify the full path.

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



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



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

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

For Centos 7:

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

The following command will install and initialize the postgres DB
#yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
#yum install -y postgresql13-server
#/usr/pgsql-13/bin/postgresql-13-setup initdb

Enable and start the service:

systemctl enable postgresql-13
systemctl start postgresql-13

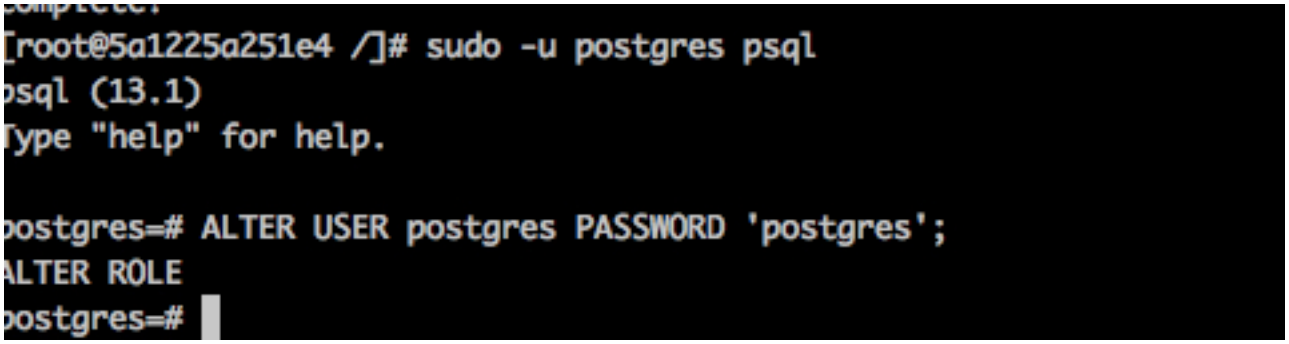
```
Complete!
[root@5a1225a251e4 /]# /usr/pgsql-13/bin/postgresql-13-setup initdb
Initializing database ... OK

[root@5a1225a251e4 /]# systemctl enable postgresql-13
Created symlink from /etc/systemd/system/multi-user.target.wants/postgresql-13.service to /usr/lib/systemd/system/postgresql-13.service.
[root@5a1225a251e4 /]# systemctl start postgresql-13
[root@5a1225a251e4 /]# systemctl status postgresql-13
● postgresql-13.service - PostgreSQL 13 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-13.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-11-15 13:21:23 UTC; 6s ago
     Docs: https://www.postgresql.org/docs/13/static/
  Process: 322 ExecStartPre=/usr/pgsql-13/bin/postgresql-13-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)
 Main PID: 327 (postmaster)
   CGroup: /docker/5a1225a251e4011bfe46ec657ce6ede8168c9c54e77cecc29df0c095aeebbd37/docker/5a1225a251e4011bfe46ec657ce6ede8168c9c54e77cecc29df
0c095aeebbd37/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.

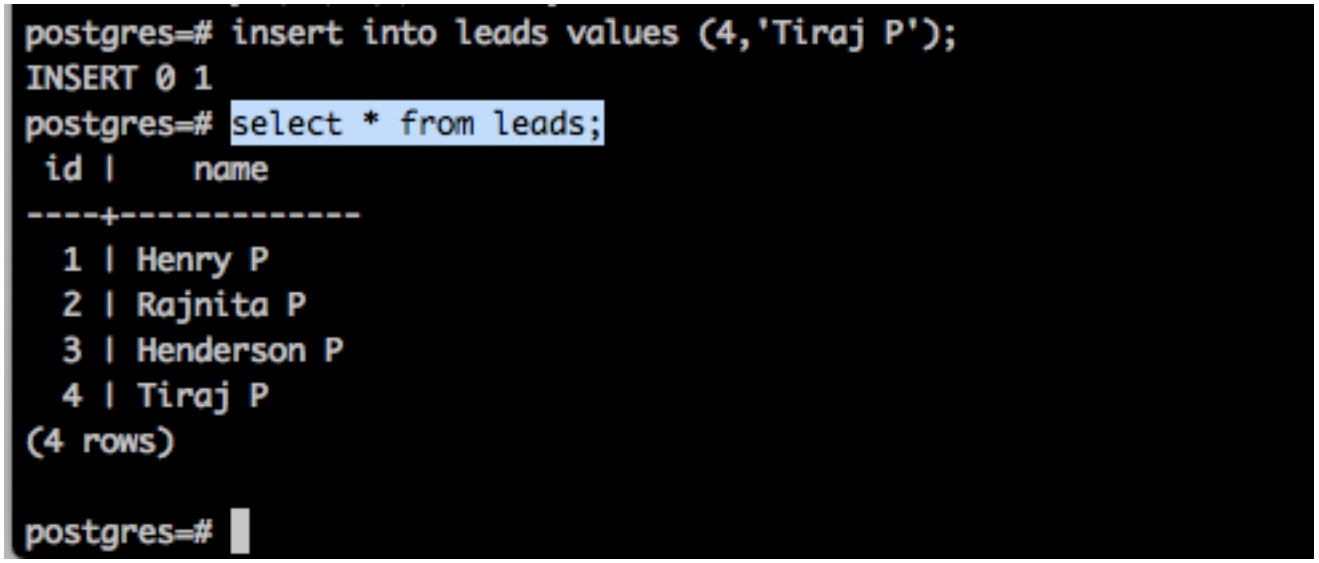
sudo -u postgres psql
ALTER USER postgres PASSWORD 'postgres';



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

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

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



To exit : \q

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

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

listen_addresses = '*'

Append in the last line of the following file.

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

host all all all trust

#systemctl restart postgresql-13

Starting the JDBC Connector – On the Kafka Node.

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

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

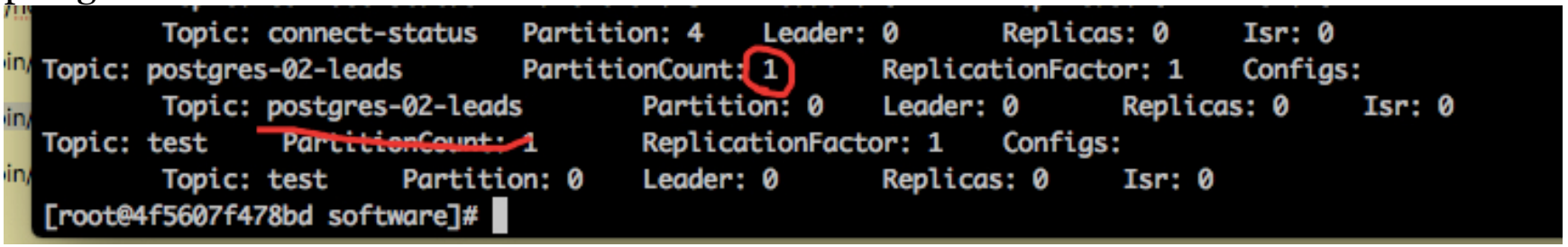
```
        timestamp.column.name = □
        timestamp.delay.interval.ms = 0
        timestamp.initial = null
        topic.prefix = postgres-02-
        validate.non.null = true
 (io.confluent.connect.jdbc.source.JdbcSourceTaskConfig:354)
[2020-11-15 16:05:29,313] INFO Using JDBC dialect PostgreSql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

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

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

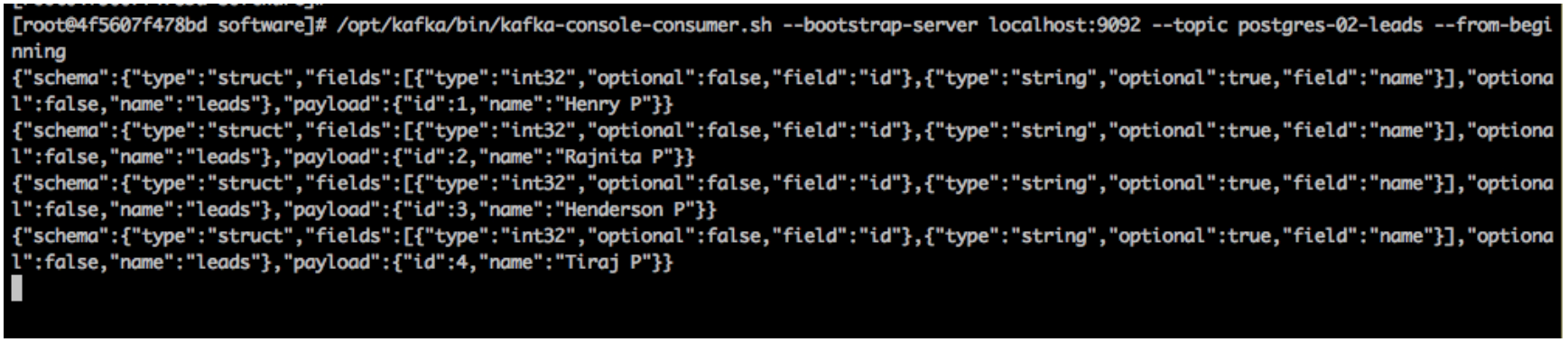
```
[root@4f5607f478bd software]# /opt/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
__consumer_offsets
connect-configs
connect-offsets
connect-status
postgres-02-leads
test
[root@4f5607f478bd software]#
```

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



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



Let us perform a Single Message Transformation.

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

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

Add another instance of connector.

Define the Single Message Tranformation parameters.

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

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

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

Start the instance

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

Verify the topic and check the message inside that topic.

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

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

Execute some Queries to get information about connectors:

Get a list of active connectors.

#curl http://localhost:8083/connectors

Get configuration info for a connector.

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

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

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

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

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

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

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

Gets the current status of the connector, including:

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

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

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

Retrieve details for specific tasks

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

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

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

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

Pre-requisite: Install kafka server.

Download and install confluent kafka : Schema Registry only.

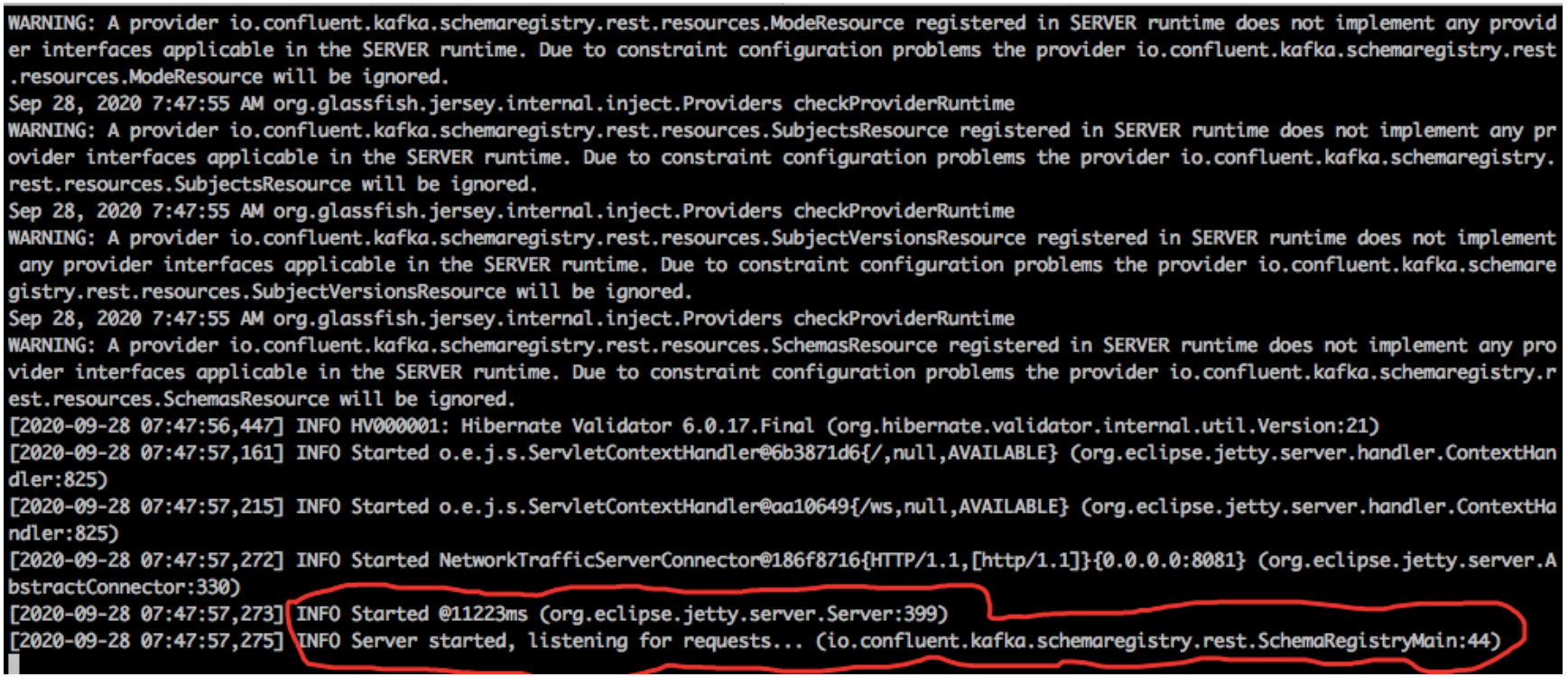
Start the Zookeeper and Broker.

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

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

Start the Registry:

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



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

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\"},{\"na
me\":\"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

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

9. 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
```

10. LOG verification + segment Sizing

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
\▒▒▒▒▒▒▒▒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. 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