November 2019

Kafka Documentation

Thomas Polzer
Hanau (Germany)

DXC.technology

1. Preface

- This presentation explains all required steps
    - to install Zookeeper and Kafka on a pre-installed Hadoop system
    - to test the functionality of the installed Kafka system
    - to write a producer based on the Java API
    - to write a consumer using the Java API
    - to do some basic configuration settings on the Kafka broker
- As a system we are using the Hadoop system with pre-installed HDFS and HBase (see presentations 01 and 02 in the existing tutorial series)
- For training purposes we are installing Kafka as a single node system (no Kafka cluster)
- As usual, Linux commands are written in cursive blue such as *ls -A*

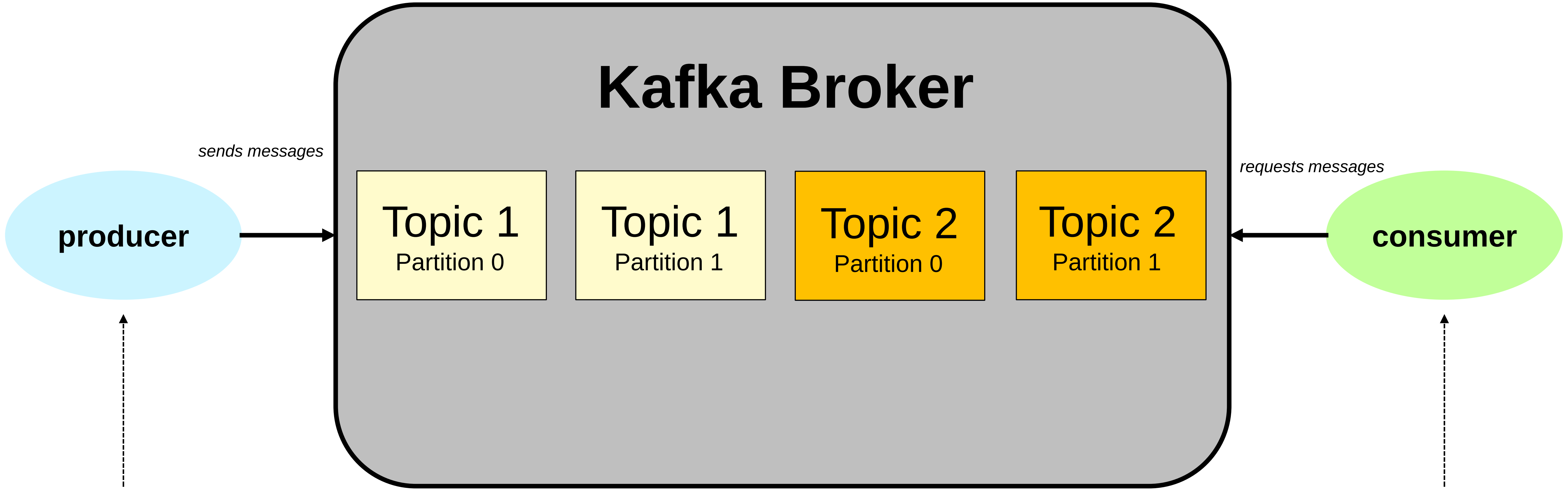
2. Overview

2.1. Fundamental terms

Term	Meaning
message	Unit of data within Kafka, **comparable to a row/record in a database system**, can have an optional key; for Kafka a message is just an array of bytes without a special meaning
topic	a container for messages that logically belong together => can be compared with a **table in a database** system
partition	A **topic** is usually divided into several **portions** (partitions) in order to achieve **scalability and redundancy**
broker	A single **Kafka server**. Receives messages from producers and responds to requests from consumers; two or more Kafka brokers make up a Kafka cluster
producer	Client that creates new **messages** and **sends** them to the Kafka broker
consumer	Client that **requests messages** from the Kafka broker
publish	Another term for **produce**
subscribe	Another term for **consume**
publish/subscripe system	**Kafka system**

2. Overview

2.2 Architecture (single server)



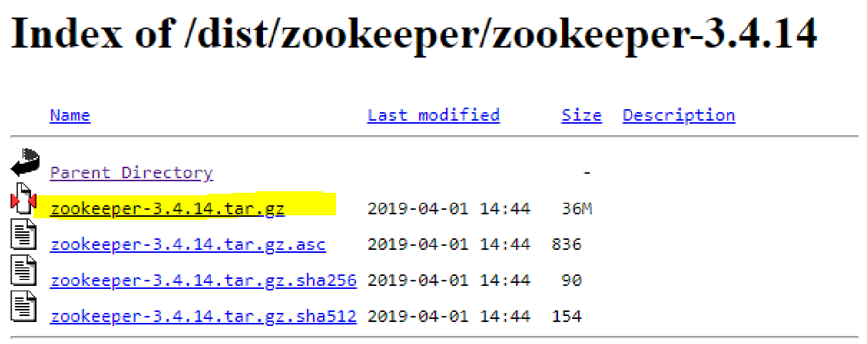
This can be Java or Python code (Kafka API) running on a web application sending out click information from users

This can be Java or Python code (Kafka API) running on an analytics platform (like Hadoop) with the aim of doing some analysis on that data

3. Installation

3.1 Zookeeper Part 1

- Kafka uses Zookeeper to store metadata about
  - Kafka cluster
  - Consumer client details
- Zookeeper requires Java 8 installation (see Hadoop presentation)
- In this installation we will use Zookeeper version 3.4.14
- Download Zookeeper on https://www-eu.apache.org/dist/zookeeper/zookeeper-3.4.14/



Index of /dist/zookeeper/zookeeper-3.4.14

```
Name                          Last modified      Size  Description

Parent Directory                                    -
zookeeper-3.4.14.tar.gz        2019-04-01 14:44   36M
zookeeper-3.4.14.tar.gz.asc    2019-04-01 14:44  836
zookeeper-3.4.14.tar.gz.sha256 2019-04-01 14:44   90
zookeeper-3.4.14.tar.gz.sha512 2019-04-01 14:44  154
```

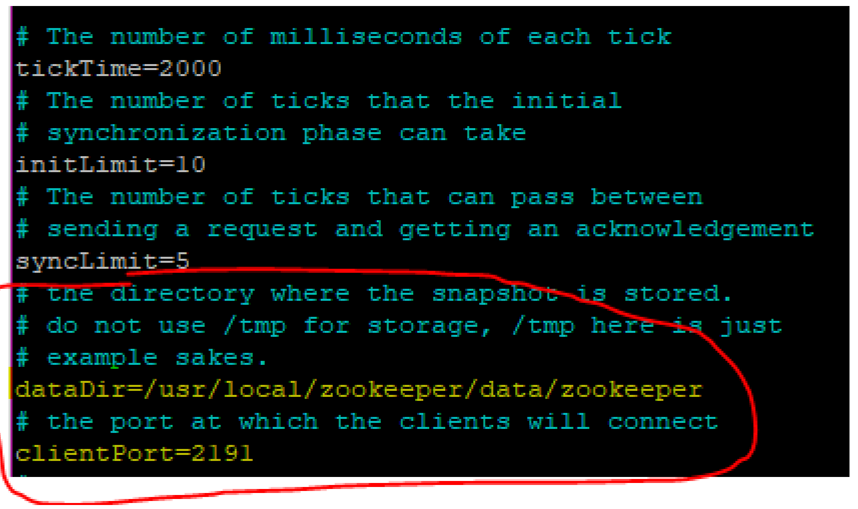
- Copy and paste the downloaded file to /usr/local

3. Installation
3.1 Zookeeper Part 2

- *sudo tar –zxf zookeeper-3.4.14.tar.gz*
- *sudo mv zookeeper-3.4.14 /usr/local/zookeeper*
- *sudo chown –R hdpuser:hdp /usr/local/zookeeper*
- *mkdir –p /usr/local/zookeeper/data/zookeeper*
- Since we are already running a zookeeper instance on our system that came as part of HBase we will need to change the client port of our new Kafka Zookeeper in order to avoid port conflicts
- Also, we ned to assign the zookeeper data directory
- *cp /usr/local/zookeeper/conf/zoo_sample.cfg /usr/local/zookeeper/conf/zoo.cfg*
- *sudo nano /usr/local/zookeeper/conf/zoo.cnf*

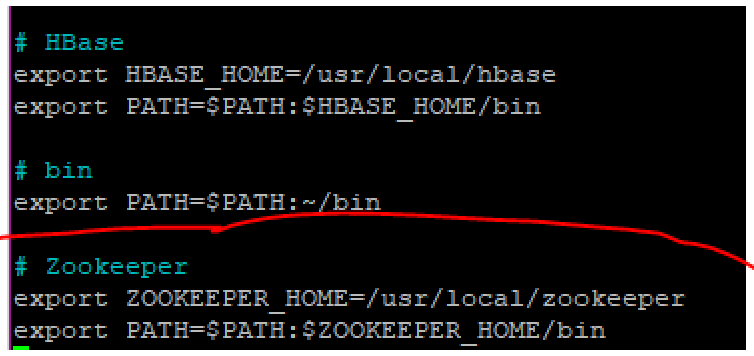
3. Installation
3.1 Zookeeper Part 3

- Change the settings as per below (dataDir, client port = 2191)

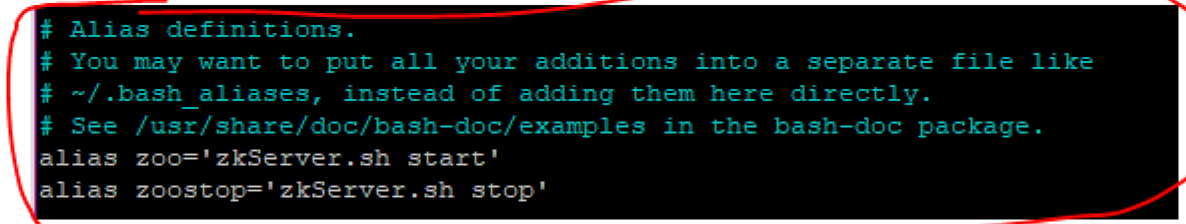


```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/usr/local/zookeeper/data/zookeeper
# the port at which the clients will connect
clientPort=2191
```

- Adjust /home/hdpuser/.bashrc file with as per below (activate afterwards with *source ~/.bashrc*:



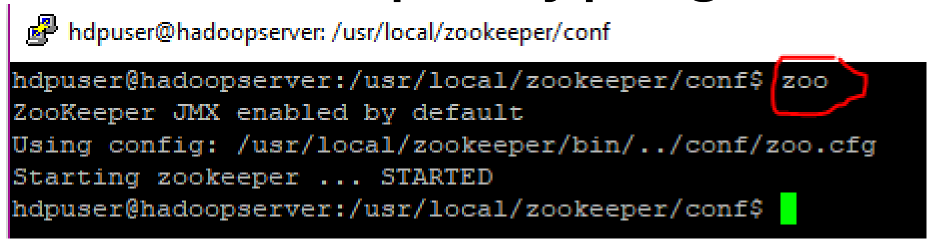
```
# HBase
export HBASE_HOME=/usr/local/hbase
export PATH=$PATH:$HBASE_HOME/bin

# bin
export PATH=$PATH:~/bin

# Zookeeper
export ZOOKEEPER_HOME=/usr/local/zookeeper
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```



```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
alias zoo='zkServer.sh start'
alias zoostop='zkServer.sh stop'
```

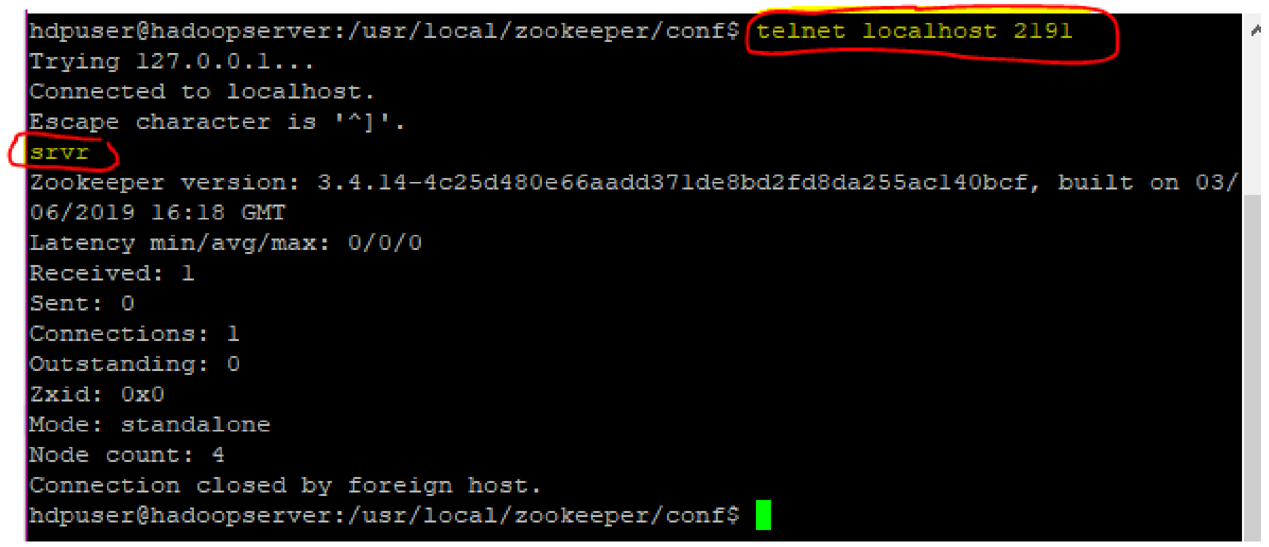
3. Installation
3.1 Zookeeper Part 4

- Start zookeper typing *zoo*



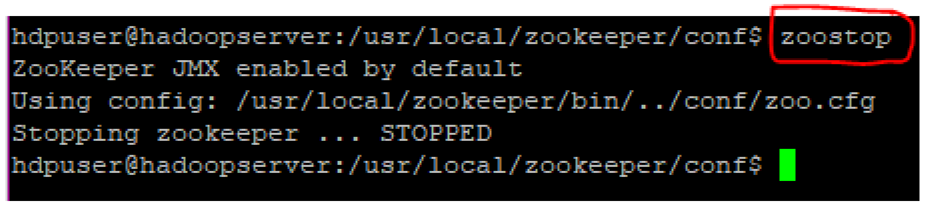
```
hdpuser@hadoopserver: /usr/local/zookeeper/conf
hdpuser@hadoopserver:/usr/local/zookeeper/conf$ zoo
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
hdpuser@hadoopserver:/usr/local/zookeeper/conf$
```

- Send the command *telnet localhost 2191* and then type *srvr*



```
hdpuser@hadoopserver:/usr/local/zookeeper/conf$ telnet localhost 2191
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.14-4c25d480e66aadd371de8bd2fd8da255ac140bcf, built on 03/
06/2019 16:18 GMT
Latency min/avg/max: 0/0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x0
Mode: standalone
Node count: 4
Connection closed by foreign host.
hdpuser@hadoopserver:/usr/local/zookeeper/conf$
```

- Stop zookeeper typing *zoostop* →

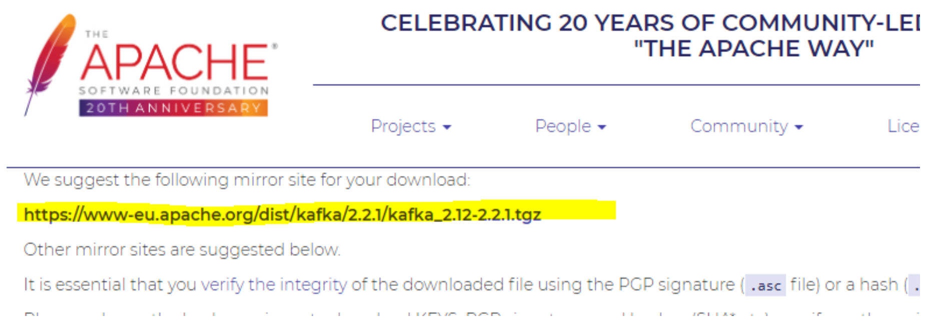


```
hdpuser@hadoopserver:/usr/local/zookeeper/conf$ zoostop
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
hdpuser@hadoopserver:/usr/local/zookeeper/conf$
```

3. Installation

3.2 Kafka Broker Part 1

- In this installation we will use Kafka version 2.12-2.2.1
- Download Kafka on https://www.apache.org/dyn/closer.cgi?path=/kafka/2.2.1/kafka_2.12-2.2.1.tgz
- Choose one of the presented mirror sites, e.g.


THE APACHE SOFTWARE FOUNDATION
20TH ANNIVERSARY
CELEBRATING 20 YEARS OF COMMUNITY-LEI
"THE APACHE WAY"
Projects
People
Community
Lice

We suggest the following mirror site for your download:

https://www-eu.apache.org/dist/kafka/2.2.1/kafka_2.12-2.2.1.tgz

Other mirror sites are suggested below.

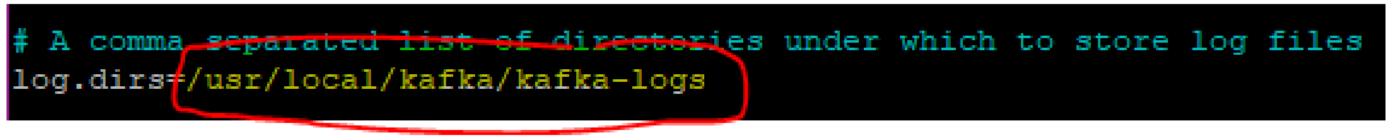
It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (.

- Once clicked the software package gets downloaded automatically
- Copy and paste the downloaded file to /usr/local

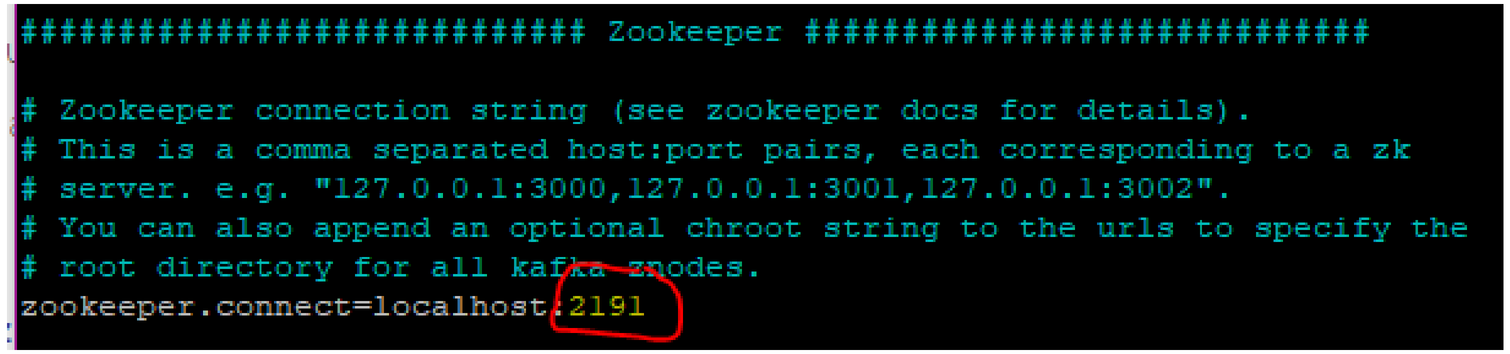
3. Installation

3.2 Kafka Broker Part 2

- *sudo tar –zxf kafka_2.12-2.2.1.tgz*
- *sudo mv kafka_2.12-2.2.1 /usr/local/kafka*
- *sudo chown –R hdpuser:hdp /usr/local/kafka*
- *mkdir –p /usr/local/kafka/kafka-logs*
- Next we need to change some basic settings on the server configuration file
- nano /usr/local/kafka/config/server.properties



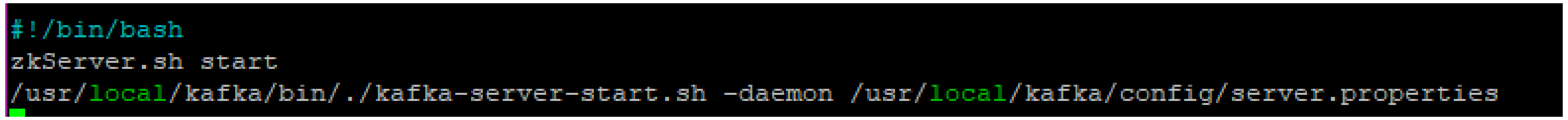
```
# A comma separated list of directories under which to store log files
log.dirs=/usr/local/kafka/kafka-logs
```



```
############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2191
```

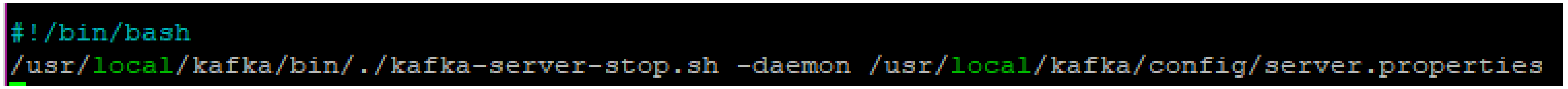
3. Installation
3.2 Kafka Broker Part 3

- Create a file named kafkastart in /home/hdpuser/bin
- *Edit the file as per below*



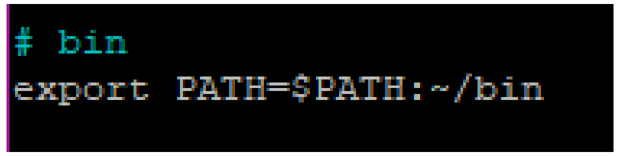
```
#!/bin/bash
zkServer.sh start
/usr/local/kafka/bin/./kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
```

- Make the file executable using *chmod 775 /home/hdpuser/bin/kafkastart*
- Create a file named kafkastop in /home/hdpuser/bin typing cp ~/bin/kafkastart ~/bin/kafkastop
- *Edit the file as per below*



```
#!/bin/bash
/usr/local/kafka/bin/./kafka-server-stop.sh -daemon /usr/local/kafka/config/server.properties
```

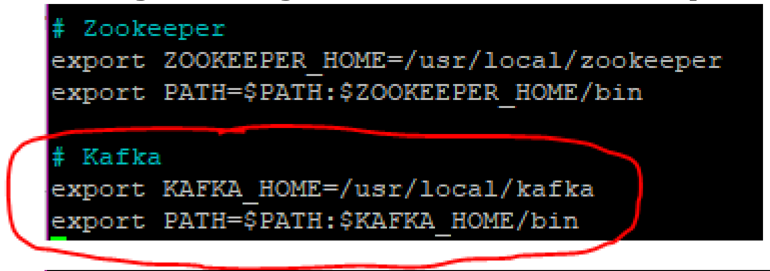
- Make sure you have the following section in your /home/hdpuser/.bashrc



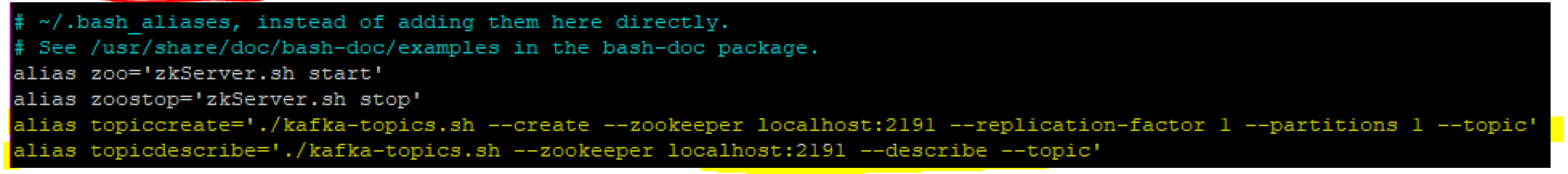
```
# bin
export PATH=$PATH:~/bin
```

3. Installation
3.2 Kafka Broker Part 4

- Adjust your /usr/hdpuser/.bashrc as per below

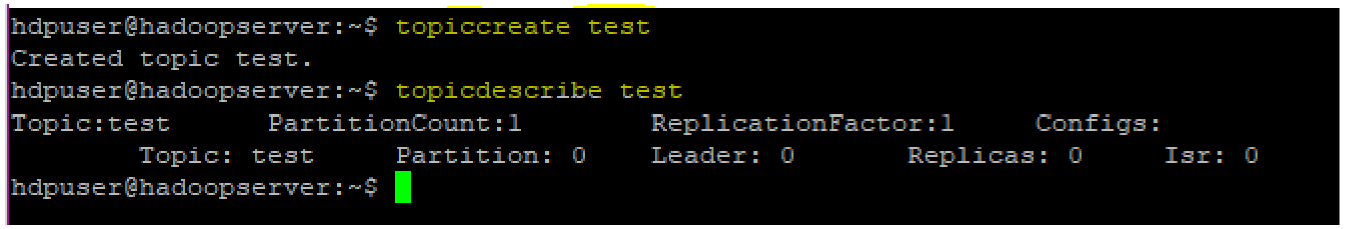


```
# Zookeeper
export ZOOKEEPER_HOME=/usr/local/zookeeper
export PATH=$PATH:$ZOOKEEPER_HOME/bin

# Kafka
export KAFKA_HOME=/usr/local/kafka
export PATH=$PATH:$KAFKA_HOME/bin
```



```
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
alias zoo='zkServer.sh start'
alias zoostop='zkServer.sh stop'
alias topiccreate='./kafka-topics.sh --create --zookeeper localhost:2191 --replication-factor 1 --partitions 1 --topic'
alias topicdescribe='./kafka-topics.sh --zookeeper localhost:2191 --describe --topic'
```

- Enter *~/source .bashrc* once finished and restart the shell
- Start Kafka typing *kafkastart*
- Create a topic typing topiccreate <topic name>, e.g. *topiccreate test*
- Enter *topicdesribe test*



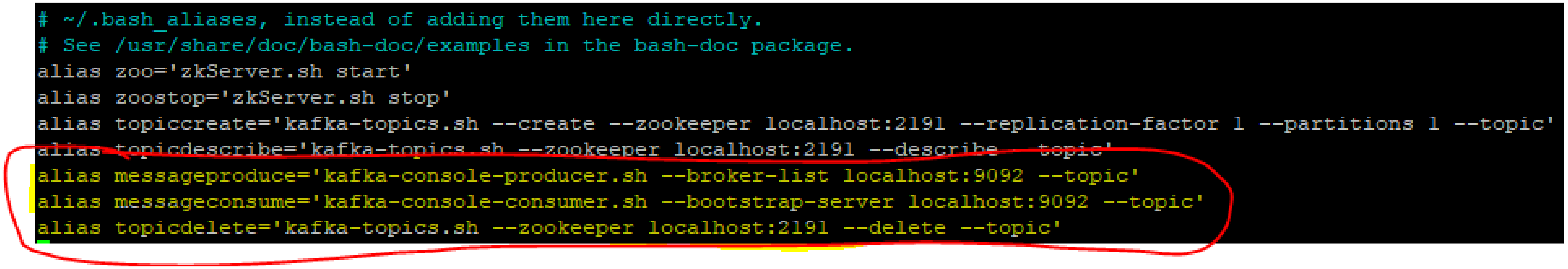
```
hdpuser@hadoopserver:~$ topiccreate test
Created topic test.
hdpuser@hadoopserver:~$ topicdescribe test
Topic:test      PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: test     Partition: 0    Leader: 0       Replicas: 0     Isr: 0
hdpuser@hadoopserver:~$
```

DXC.technology

3. Installation

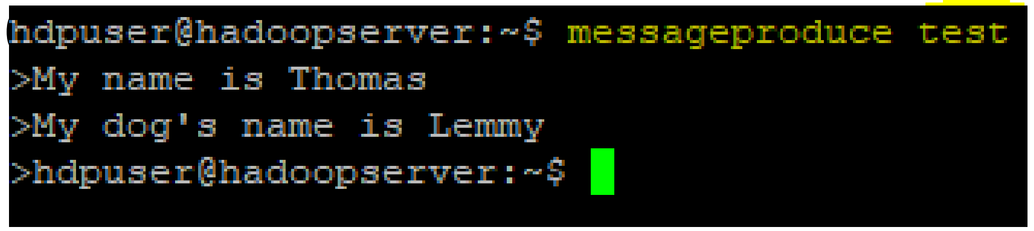
3.2 Kafka Broker Part 5

- Adjust your /usr/hdpuser/.bashrc as per below



```
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
alias zoo='zkServer.sh start'
alias zoostop='zkServer.sh stop'
alias topiccreate='kafka-topics.sh --create --zookeeper localhost:2191 --replication-factor 1 --partitions 1 --topic'
alias topicdescribe='kafka-topics.sh --zookeeper localhost:2191 --describe --topic'
alias messageproduce='kafka-console-producer.sh --broker-list localhost:9092 --topic'
alias messageconsume='kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic'
alias topicdelete='kafka-topics.sh --zookeeper localhost:2191 --delete --topic'
```

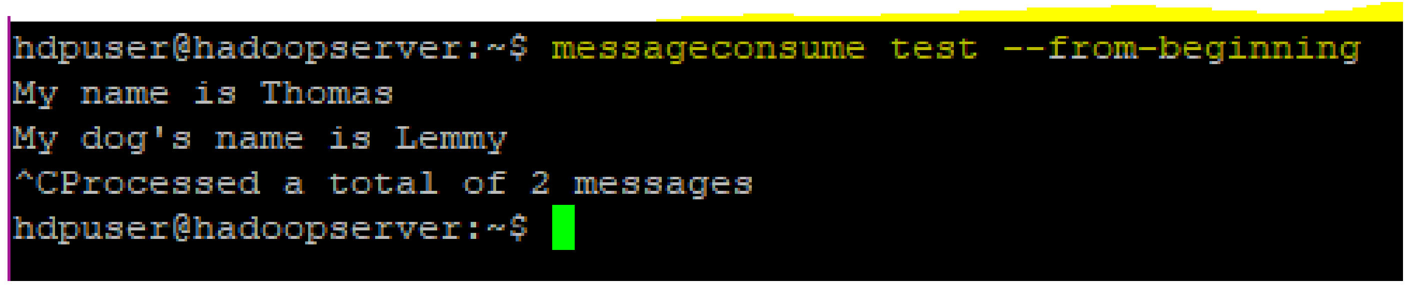
- *Another one: alias topiclist='kafka-topics.sh --zookeeper localhost:2191 --list'*
- Enter *~/source .bashrc* once finished and restart the shell
- Enter *messageproduce <topic name>* to be followed by some messages and



```
hdpuser@hadoopserver:~$ messageproduce test
>My name is Thomas
>My dog's name is Lemmy
>hdpuser@hadoopserver:~$
```

3. Installation
3.2 Kafka Broker Part 6

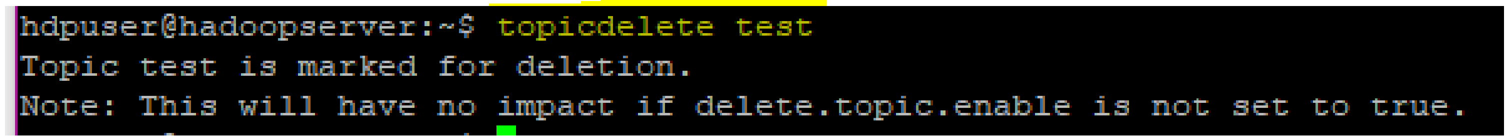
- In order to consume messages on a topic enter *messageconsume <topic name> --from-beginning* and quit typing STRG-C



```
hdpuser@hadoopserver:~$ messageconsume test --from-beginning
My name is Thomas
My dog's name is Lemmy
^CProcessed a total of 2 messages
hdpuser@hadoopserver:~$
```

→ **Kafka is running fine ☺**

- Enter *topicdelete <topic name>* to delete the dummy topic



```
hdpuser@hadoopserver:~$ topicdelete test
Topic test is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

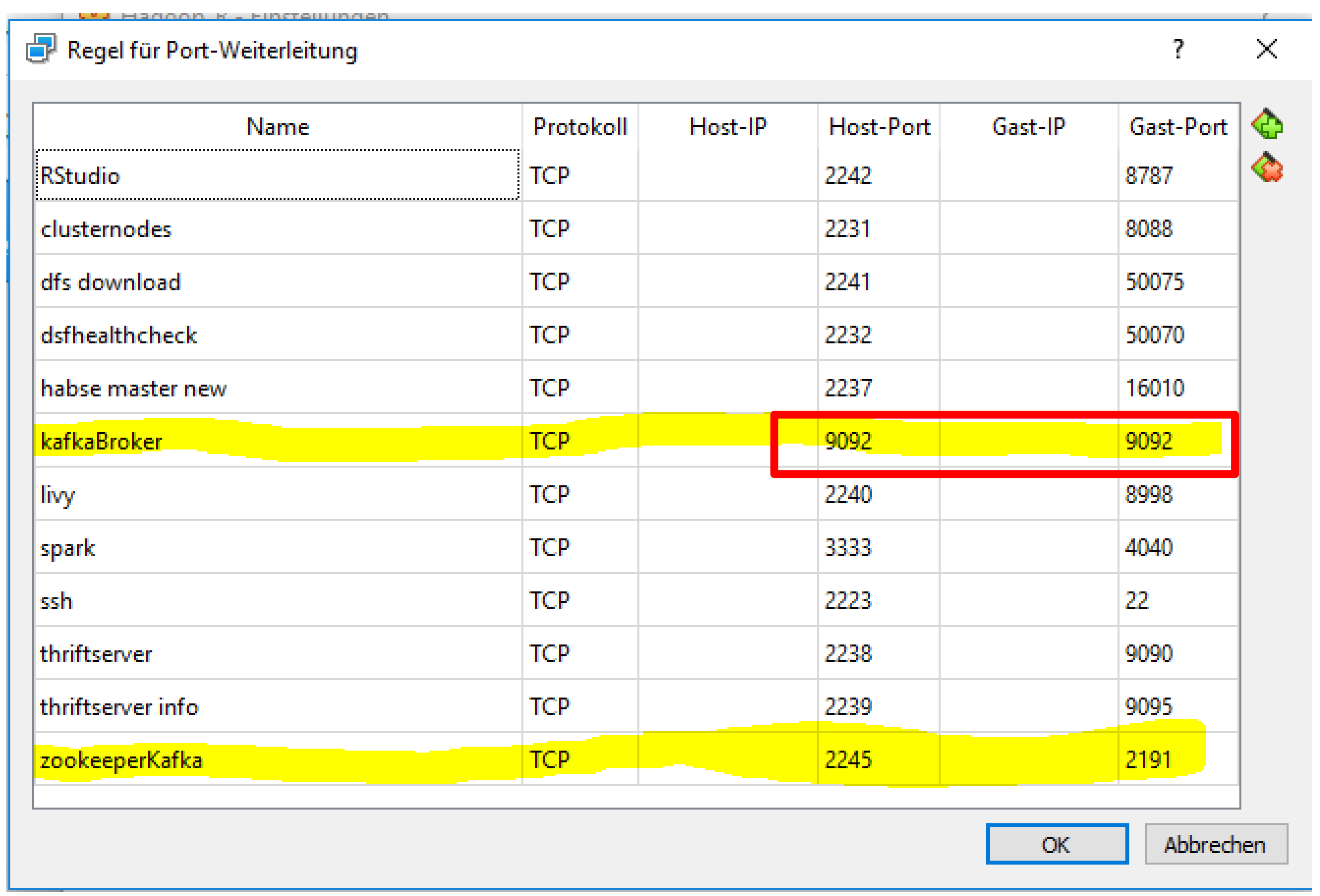
- By default delete.topic.enable is set to true, in case you want to change this enter *nano /usr/local/kafka/config/server.properties* and add at the end of the file *delete.topic.enable=false*

4. First Review: What have we achieved so far?

- We have installed Zookeeper (version 3.4.14) as it is required by Kafka
- We have installed Kafka (version 2.12-2.2.1) and changed some settings in the server.properties config file (dataDir and client port)
- We have created our first topic using the built in Kafka client
- We have produced messages for this topic using the built in Kafka client
- We have consumed the messages created before using the built in Kafka client
- We have deleted our dummy topic using the built in Kafka client
- So what comes next?
    - In real life creating topics and messages within Kafka broker actually does not make sense
    - So we will start writing a producer from an external system connected to the Kafka broker

5. Kafka Producers: Writing messages to Kafka

- We will use the Java API for Apache Kafka
- Example: Java web application
- Connect to Kafka Broker and send a message to the topic „gude“
- Add port-forwardings: **CAUTION: leave kafka-broker port as it is!**

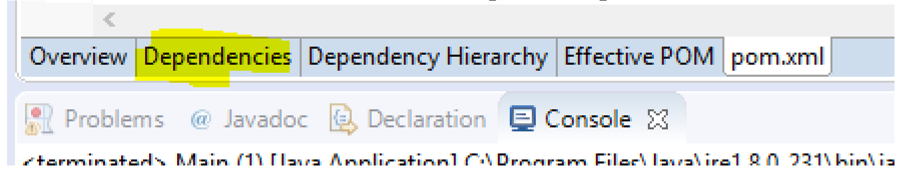


Regel für Port-Weiterleitung

Name	Protokoll	Host-IP	Host-Port	Gast-IP	Gast-Port
RStudio	TCP		2242		8787
clusternodes	TCP		2231		8088
dfs download	TCP		2241		50075
dsfhealthcheck	TCP		2232		50070
habse master new	TCP		2237		16010
kafkaBroker	TCP		9092		9092
livy	TCP		2240		8998
spark	TCP		3333		4040
ssh	TCP		2223		22
thriftserver	TCP		2238		9090
thriftserver info	TCP		2239		9095
zookeeperKafka	TCP		2245		2191

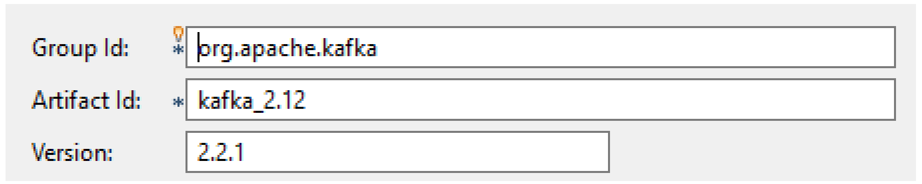
5. Kafka Producers: Prepare your JAVA project

- Create a java project called KafkaManager
- In the src folder add two packages, org.<name>.starter and org.<name>.kafka
- In kafka package add a class called Producerclient
- In starter package add a class called Main
- Right click on your project → Configure → Convert to Maven project
- Open your pom.xml → Dependencies



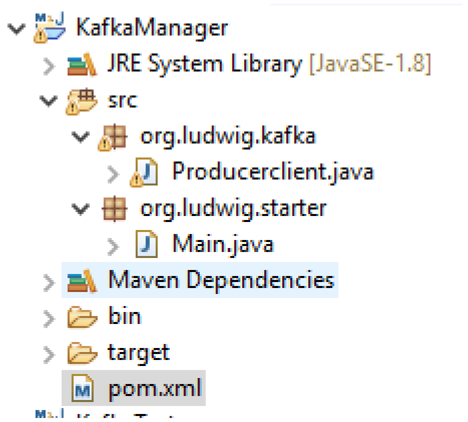
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\ja

- Click on Add, enter the following and save file:



Group Id:	org.apache.kafka
Artifact Id:	kafka_2.12
Version:	2.2.1

- Your project should look like this now:



```
KafkaManager
  JRE System Library [JavaSE-1.8]
  src
    org.ludwig.kafka
      Producerclient.java
    org.ludwig.starter
      Main.java
  Maven Dependencies
  bin
  target
  pom.xml
```

5. Kafka Producers: Producerclient.java

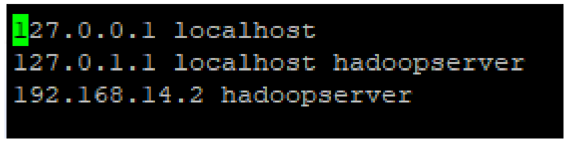
```
package org.ludwig.kafka;

import java.net.InetAddress;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class Producerclient {

        public void createProducer() {

                Properties kafkaProps = new Properties();
                kafkaProps.put("bootstrap.servers", “localhost:9092");
                kafkaProps.put("acks", "0");
                kafkaProps.put("retries", 0);
                kafkaProps.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
                kafkaProps.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

                Producer<String, String> producer = new KafkaProducer<String, String>(kafkaProps);

                ProducerRecord<String, String> record = new ProducerRecord<> ("gude","key","ai gude wie");




                try {
producer.send(record).get();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (ExecutionException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

                 System.out.println("Record sent");
                 producer.flush();
                 producer.close();

        }

}
```

5. Kafka Producers: Main.java

```
package org.ludwig.starter;

import org.ludwig.kafka.*;

public class Main {

    public static void main(String[] args) {

        Producerclient producerclient = new Producerclient();
        producerclient.createProducer();

    }

}
```

5. Kafka Producers: Network settings

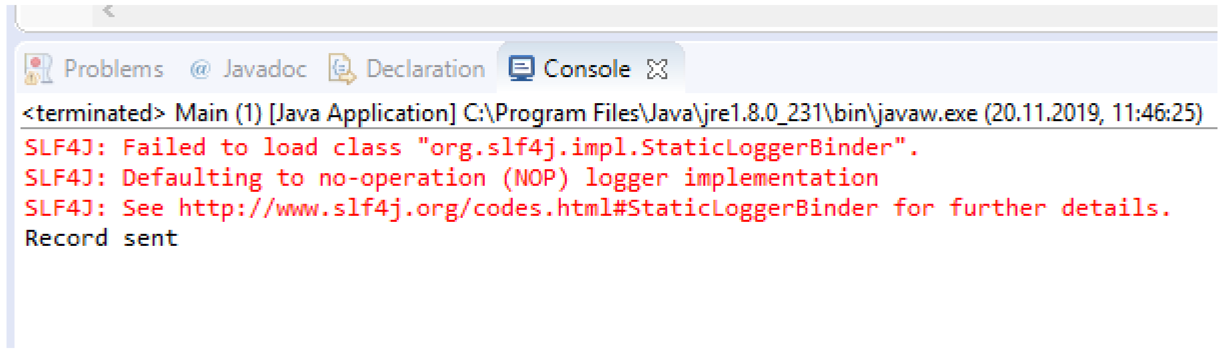
- On the linux VM: sudo nano /etc/hosts and enter the following values (192.168.14.2 is the IP assigned by virtual host adapter)



```
127.0.0.1 localhost
127.0.1.1 localhost hadoopserver
192.168.14.2 hadoopserver
```

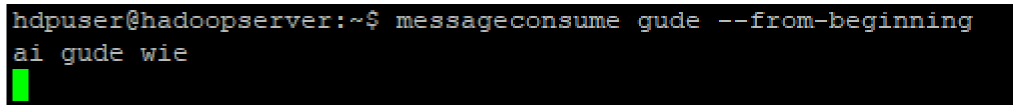
5. Kafka Producers: Verification

- When you run the Main-Class as java project you should see the following on your console:



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (20.11.2019, 11:46:25)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Record sent
```

- To check on the broker side if it worked you can do the following:



```
hdpuser@hadoopserver:~$ messageconsume gude --from-beginning
ai gude wie
```

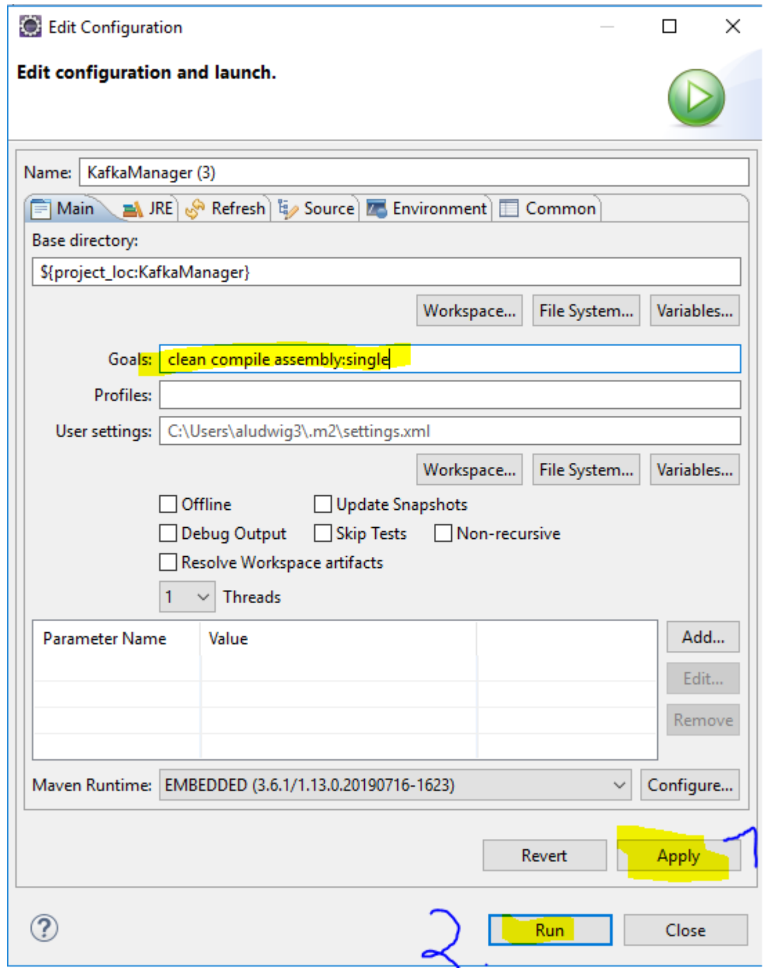
- You should the the message “ai gude wie” in the topic “gude”

5. Creating executable .jar out of maven

- Add to pom.xml following part (between </plugin> and </plugins>):

```
<plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
                <archive>
                        <manifest>
                                <mainClass>org.ludwig.starter.Main</mainClass>
                        </manifest>
                </archive>
                <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
        </configuration>
</plugin>
```

- In eclipse: Right click on your project → Run as → maven build…


Edit Configuration

Edit configuration and launch.

Name: KafkaManager (3)

Base directory: ${project_loc:KafkaManager}

Goals: clean compile assembly:single

Profiles:

User settings: C:\Users\aludwig3\.m2\settings.xml

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts

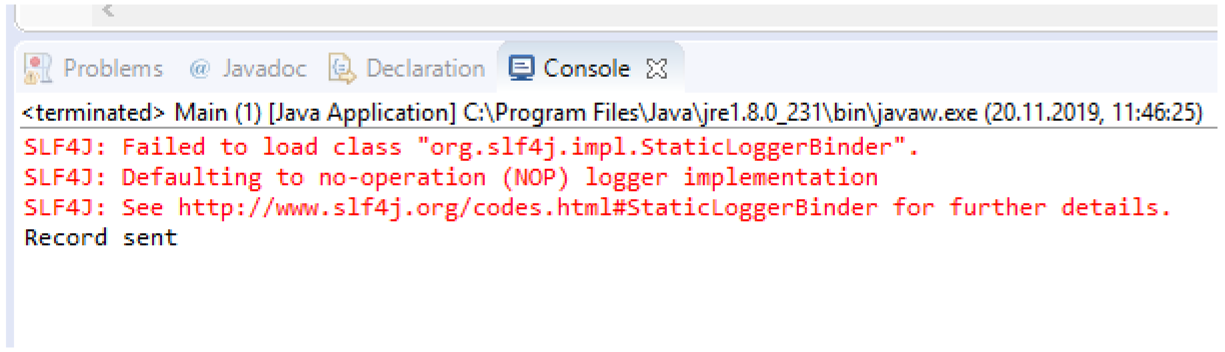
1 Threads

Parameter Name	Value

Maven Runtime: EMBEDDED (3.6.1/1.13.0.20190716-1623)

DXC.technology

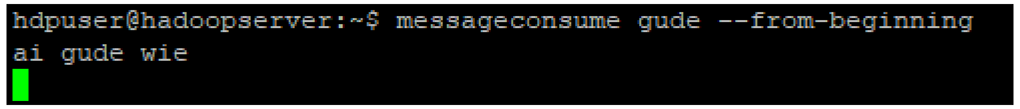
5. Kafka Producers: Verification



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (20.11.2019, 11:46:25)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Record sent
```

- To check on the broker side if it worked you can do the following:

- You should the the message “ai gude wie” in the topic “gude”



```
hdpuser@hadoopserver:~$ messageconsume gude --from-beginning
ai gude wie
```

6. Kafka Connect

- Kafka Connect allows to easily transfer entire files
- Connect is used if you are not interested in how the connection is configured, you are interested in the transfered data only
- Will be used to pull data from the external datastore into Kafka or push data from Kafka to an external store
- No separate installation required, Connect ships with Apache Kafka

6.1 Running Connect Part 1

As an example

- we will sending the content of an entire file to a Kafka topic
- and have it saved in a new file
- This procedure is called „file source and file sink“

Create a folder /usr/local/kafka/files and store the below content as /usr/local/kafka/files/airpollution_1.txt

```
"Date","sulfate","nitrate","ID"
"2010-01-01",NA,NA,"L1000488"
"2010-01-02",NA,NA,"L1000488"
"2010-01-03",NA,NA,"L1000488"
"2010-01-04",NA,NA,"L1000488"
"2010-01-05",NA,NA,"L1000488"
```

6.2 Running Connect Part 2: Source configuration

Next step:

- Create a file named /usr/local/kafka/connect-file-source-pollution.properties inserting the following content:

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=files/airpollution_1.txt
topic=connect-airpollution1
```

6.3 Running Connect Part 3: Sink configuration

Next step:

- Create a file named /usr/local/kafka/connect-file-sink-pollution.properties inserting the following content:

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=files/pollutionsink
topics=connect-airpollution1
```

„s“ is important!

6.4 Running Connect Part 4: Run

Start the below command from
/usr/local/kafka/

```
bin/connect-standalone.sh config/connect-standalone.properties connect-file-source-pollution.properties connect-file-sink-pollution.properties
```

We are using the standalone mode, see
https://docs.confluent.io/3.2.2/connect/quickstart.html

7. Connect Kafka to Relational database (source) and automatically replicate data into HDFS and Hive (sink)

This approach requires the following steps:

1. Download and install relational database (here: MySQL database)
2. Configure hive metastore as remote hive metastore hosted by MySQL database
3. Download Kafka jdbc source connector and Kafks hdfs sink connector and copy relevant jar-files into Kafka classpath (/usr/local/kafka/lib)
4. Download kyro-2.21-shaded.jar and copy into Kafka classpath
5. Create jdbc-source.properties file
6. Create hive-sink.properties file
7. Execute Kafka Connect using configured jdbc source properties file and hive sink properties file

7.1 Install MySQL database

Enter the following commands for installation

```
sudo apt update
sudo apt install mysql-server
sudo mysql_secure_installation
```

Setting a password for root (from mysql shell)

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH auth_socket BY 'password’;
mysql> FLUSH PRIVILEGES;
```

Create new user:

```
mysqld> CREATE DATABASE testuser;
```

3. Create user:

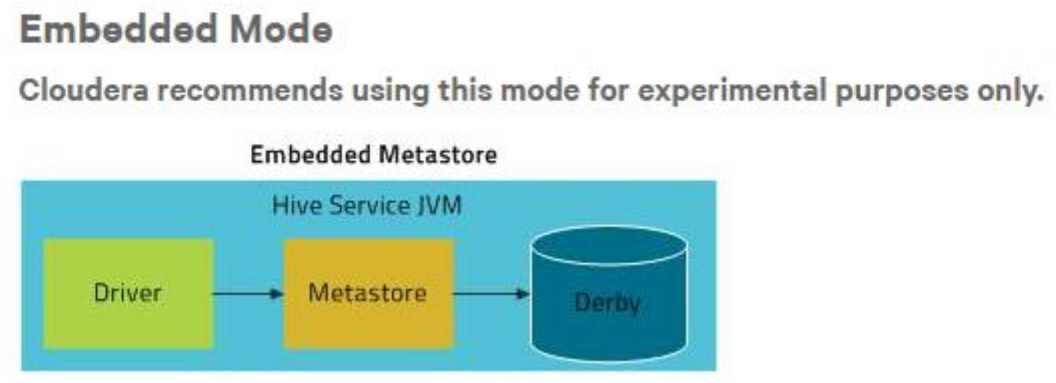
```
mysqld> CREATE USER 'testuser'@'localhost' IDENTIFIED WITH mysql_native_password BY 'testuser';
```

4. Grant privileges:

```
mysqld> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, INDEX, DROP, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES ON testuser.* TO 'testuser'@'localhost';
```

7.2 Hive metastore (part 1)

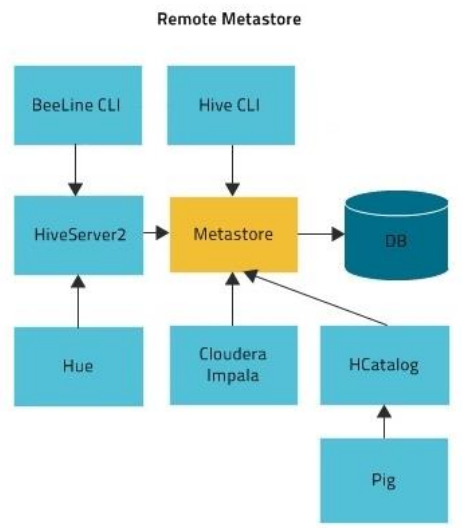
On a local hive installation the following embedded mode is pre-configured:



Embedded Mode

Cloudera recommends using this mode for experimental purposes only.

In order to give Kafka access to the metastore „remote Mode“ is required:



In Remote mode, the Hive metastore service runs in its own JVM process. HiveServer2, HCatalog, Impala, and other processes communicate with it using the Thrift network API (configured using the hive.metastore.uris property). The metastore service communicates with the metastore database over JDBC (configured using the javax.jdo.option.ConnectionURL property). The database, the HiveServer2 process, and the metastore service can all be on the same host, but running the HiveServer2 process on a separate host provides better availability and scalability.
The main advantage of Remote mode over Local mode is that Remote mode does not require the administrator to share JDBC login information for the metastore database with each Hive user. HCatalog requires this mode.

7.2 Hive metastore (part 2): MySQL settings

Create metastore database and metastore user

```
$ mysql –u root –p

mysql> CREATE DATABASE metastore;
mysql> USE metastore;
mysql> SOURCE /usr/local/hive/scripts/metastore/upgrade/mysql/hive-schema-0.14.0.mysql.sql;

mysql> CREATE USER ‘hiveuser‘@‘%‘ IDENTIFIED BY ‘hivepassword‘;
mysql> GRANT all on *.* to ‘hiveuser‘@localhost IDENTIFIED BY ‘hivepassword‘;
mysql> flush privileges;
```

See:
https://dzone.com/articles/how-configure-mysql-metastore

7.2 Hive metastore (part 3): Adjust hive-site.xml

Adjust the following parts of /usr/local/hive/conf/hive-site.xml

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://myhost/metastore</value>
<description>the URL of the MySQL database</description>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hive</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>mypassword</value>
</property>
<property>
<name>datanucleus.autoCreateSchema</name>
<value>false</value>
</property>
<property>
<name>datanucleus.fixedDatastore</name>
<value>true</value>
</property>
<property>
<name>datanucleus.autoStartMechanism</name>
<value>SchemaTable</value>
</property>
<property>
<name>hive.metastore.uris</name>
<value>thrift://<n.n.n.n>:9083</value>
<description>IP address (or fully-qualified domain name) and port of the metastore
host</description>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>true</value>
</property>
```

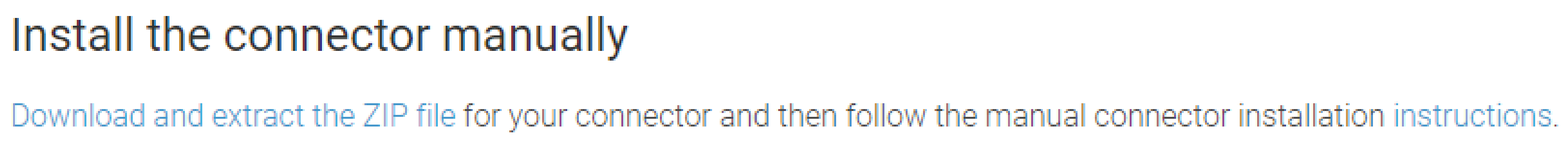
See:

https://docs.cloudera.com/documentation/enterprise/5-8-x/topics/cdh_ig_hive_metastore_configure.html

You can get a proper hive-site.xml file under git@github.com:thpolzer/KafkaConnect.git

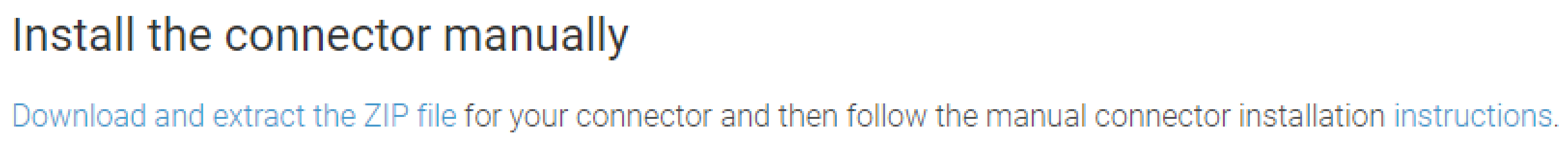
7.3 Kafka connectors

Download Kafka Jdbc connector (jar-files):
https://docs.confluent.io/current/connect/kafka-connect-jdbc/index.html


Install the connector manually

Download and extract the ZIP file for your connector and then follow the manual connector installation instructions.

Download Kafka hdfs sink connector (jar-files):
https://docs.confluent.io/current/connect/kafka-connect-jdbc/index.html


Install the connector manually

Download and extract the ZIP file for your connector and then follow the manual connector installation instructions.

Download kyro-2.21-shaded.jar

7.4 Kafka classpath

1. Kafka classpath is /usr/local/kafka/libs
2. Copy and paste the jar files downloaded in 8.3 into Kafka classpath
3. Important: Make sure that the classpath does not contain different versions of the same jar-file. In doubt, choose the latest one

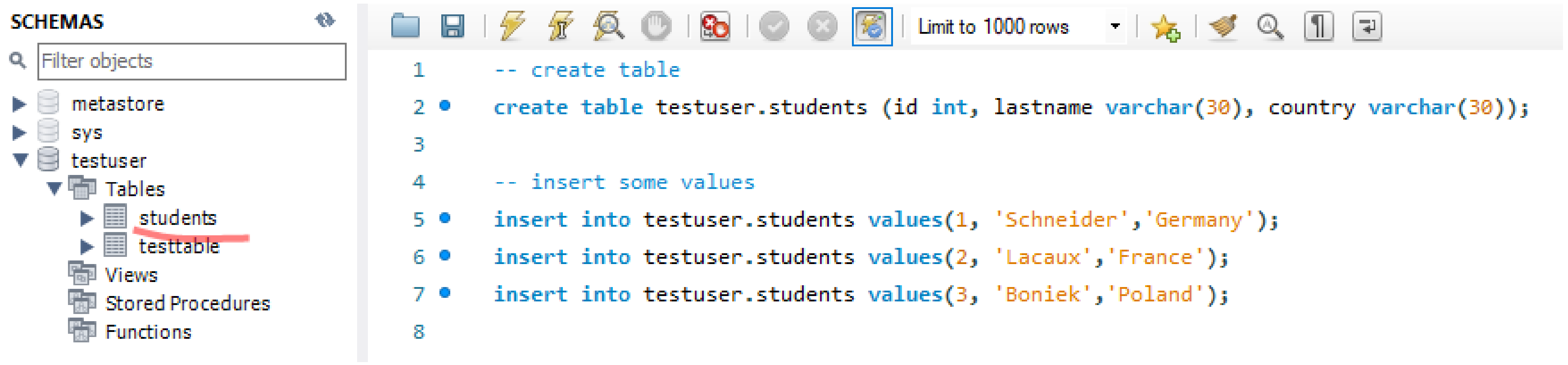
You can get a proper lib folder under
git@github.com:thpolzer/KafkaConnect.git

7.5 End-To-End-Example

This example covers the following steps:

1. Create and populate table in a database
2. Configure Kafka Connect
3. Run Kafka Connect
4. Check replication status on HDFS and Hive
5. Increment table on database
6. Check delta load on Hive

7.5.1 Create and populate table in a database



```
-- create table
create table testuser.students (id int, lastname varchar(30), country varchar(30));

-- insert some values
insert into testuser.students values(1, 'Schneider','Germany');
insert into testuser.students values(2, 'Lacaux','France');
insert into testuser.students values(3, 'Boniek','Poland');
```

7.5.2 Configure Kafka Connect part 1

- Create folder /usr/local/kafka/connect_properties
- Inside the above folder: Enter nano connect-jdbc-source-mysql.properties
- Put the below content into this file and save:

```
name=mysql-source-connector
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector
connection.url=jdbc:mysql://127.0.0.1:3306/testuser?user=testuser&password=Ich.bin.51
mode=incrementing
incrementing.column.name=id
whitelist=boys
validate.non.null=false
topic.prefix=mysqlhive.
```

- Close the file connect-jdbc-source-mysql.properties

Take your own password

You can get a proper connect-jdbc-source-mysql.properties file under git@github.com:thpolzer/KafkaConnect.git

7.5.2 Configure Kafka Connect part 2

- Navigate to folder /usr/local/kafka/connect_properties
- Inside the above folder: Enter nano connect-hive-properties-sink.properties
- Put the below content into this file and save:

```
name=hive-sink
connector.class=io.confluent.connect.hdfs.HdfsSinkConnector
format.class=io.confluent.connect.hdfs.avro.AvroFormat
tasks.max=1
topics=mysqlhive.students
hdfs.url=hdfs://localhost:9000
hive.integration=true
hive.metastore.uris=thrift://localhost:9083
flush.size=1
schema.compatibility=BACKWARD
```

You can get a proper connect-hive-properties-sink.properties file under git@github.com:thpolzer/KafkaConnect.git

- Close the file connect-hive-properties-sink.properties

7.5.3 Run Kafka Connect

- Start HDFS using start-all.sh
- Start Kafka using zoo; kafkastart
- Start Hive Metastore using hive –service metastore in a separate terminal (2 minus characters in front of service)
- Start Hiveserver2 using hivestart in a separate terminal
- The two steps above take a little time, once the command sudo nano netstat tulpn | grep 9083 returns a result everything is up and running
- Start beeline client using beeline in a separate client
- Once you have a connection to hiveserver2 by typing !connect jdbc:hive2://localhost:10000 hdpuser hadoop
- navigate to /usr/local/kafka entering the below command and hit enter

```
bin/connect-standalone.sh config/connect-standalone.properties connect_properties/connect-jdbc-source-mysql.properties
connect_properties/connect-hive-properties-sink.properties
```

- Kafka Connect has started, give it 2 minutes (you see the progress in the terminal)

7.5.3 Run Kafka Connect

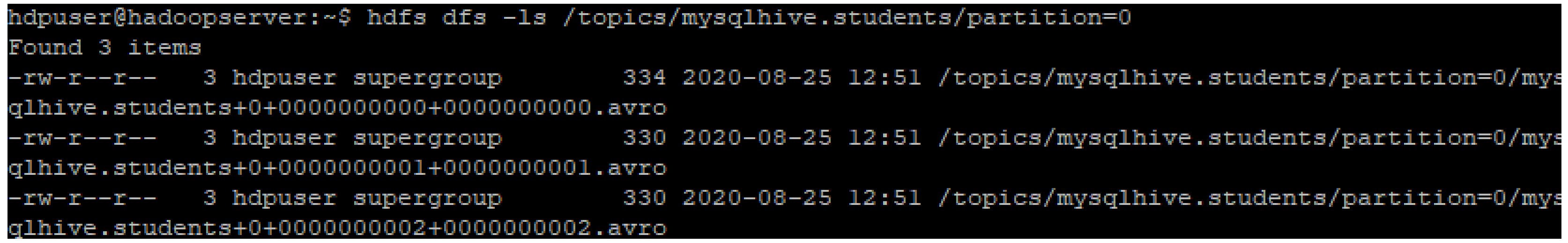
- Start HDFS using start-all.sh
- Start Kafka using zoo; kafkastart
- Start Hive Metastore using hive –service metastore in a separate terminal (2 minus characters in front of service)
- Start Hiveserver2 using hivestart in a separate terminal
- The two steps above take a little time, once the command sudo nano netstat tulpn | grep 9083 returns a result everything is up and running
- Start beeline client using beeline in a separate client
- Once you have a connection to hiveserver2 by typing !connect jdbc:hive2://localhost:10000 hdpuser hadoop
- navigate to /usr/local/kafka entering the below command and hit enter

```
bin/connect-standalone.sh config/connect-standalone.properties connect_properties/connect-jdbc-source-mysql.properties
connect_properties/connect-hive-properties-sink.properties
```

- Kafka Connect has started, give it 2 minutes (you see the progress in the terminal)

7.5.4 Check HDFS and Hive Status

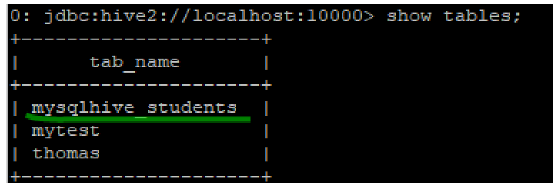
- **HDFS**
- Enter hdfs dfs -ls /topics/mysqlhive.students/partition=0



```
hdpuser@hadoopserver:~$ hdfs dfs -ls /topics/mysqlhive.students/partition=0
Found 3 items
-rw-r--r--   3 hdpuser supergroup        334 2020-08-25 12:51 /topics/mysqlhive.students/partition=0/mys
qlhive.students+0+0000000000+0000000000.avro
-rw-r--r--   3 hdpuser supergroup        330 2020-08-25 12:51 /topics/mysqlhive.students/partition=0/mys
qlhive.students+0+0000000001+0000000001.avro
-rw-r--r--   3 hdpuser supergroup        330 2020-08-25 12:51 /topics/mysqlhive.students/partition=0/mys
qlhive.students+0+0000000002+0000000002.avro
```

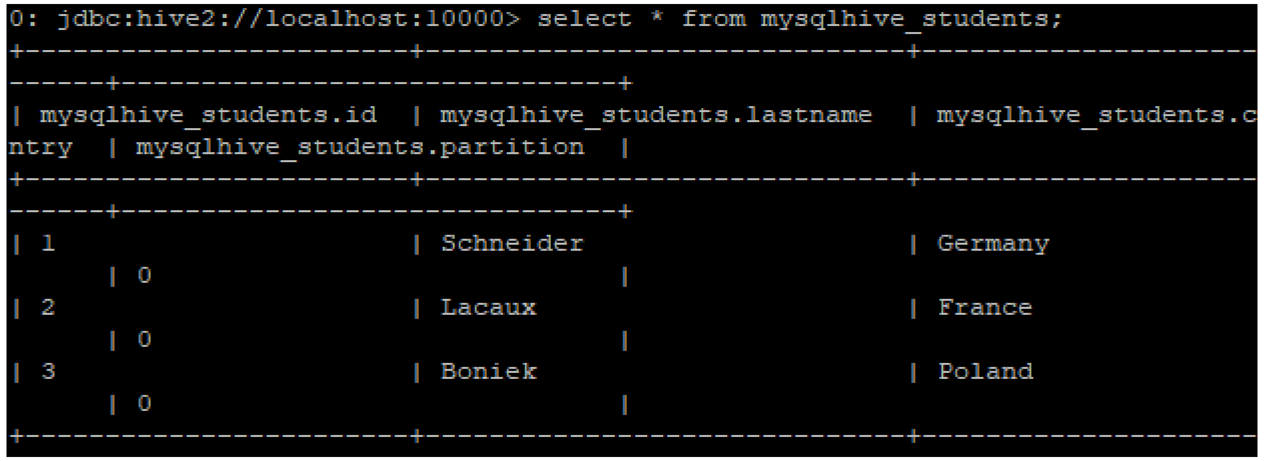
- Successful, file has been created ☺

- **Hive**
- Go to beeline client and enter use default;
- Enter show tables;



```
0: jdbc:hive2://localhost:10000> show tables;
+---------------------+
|      tab_name       |
+---------------------+
| mysqlhive_students  |
| mytest              |
| thomas              |
+---------------------+
```

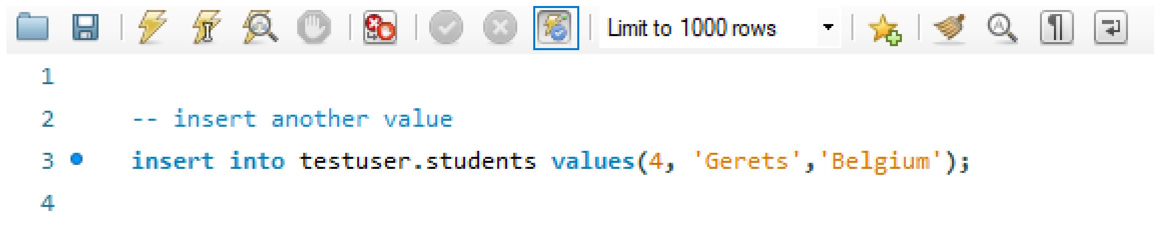
- Enter select * from mysqlhive_students;



```
0: jdbc:hive2://localhost:10000> select * from mysqlhive_students;
+------------------------+-------------------------------+---------------------
------+------------------------------+
| mysqlhive_students.id  | mysqlhive_students.lastname   | mysqlhive_students.c
ntry   | mysqlhive_students.partition  |
+------------------------+-------------------------------+---------------------
------+------------------------------+
| 1                      | Schneider                     | Germany
      | 0                            |
| 2                      | Lacaux                        | France
      | 0                            |
| 3                      | Boniek                        | Poland
      | 0                            |
+------------------------+-------------------------------+---------------------
```

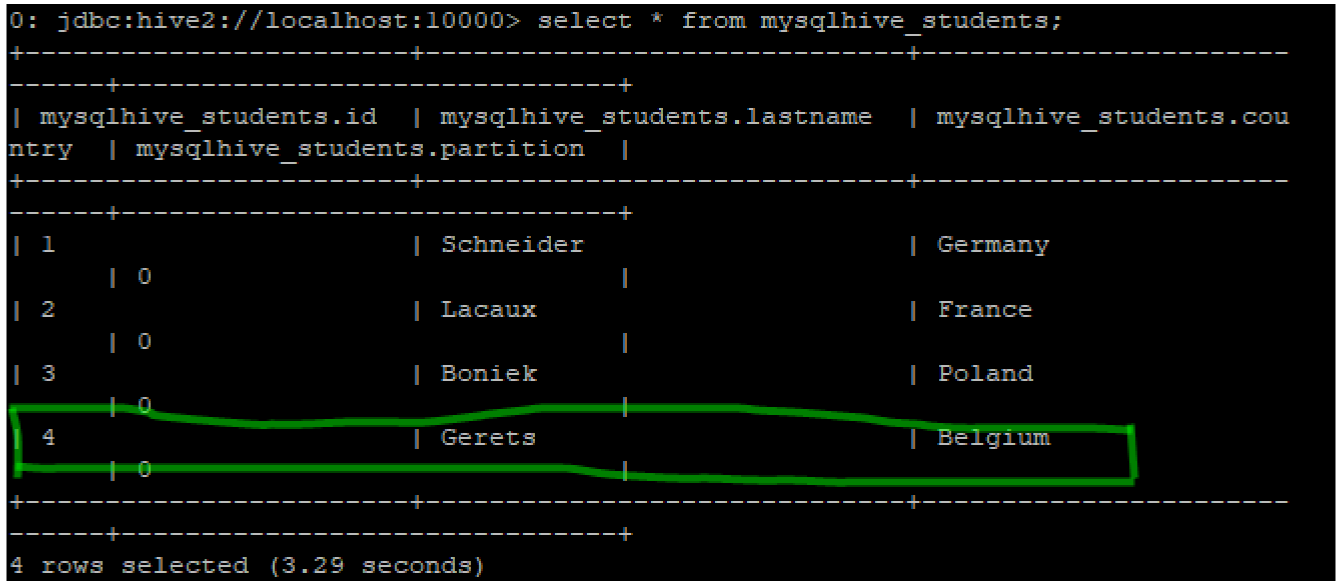
7.5.5 Increment data on database and check effect on Hive

Go to database and add a new row to the table:



```

-- insert another value
insert into testuser.students values(4, 'Gerets','Belgium');

```

Go to beeline client and check if the new row has automatically been replicated



```
0: jdbc:hive2://localhost:10000> select * from mysqlhive_students;
+-------------------------+-------------------------------+-----------------------
------+--------------------------------+
| mysqlhive_students.id  | mysqlhive_students.lastname  | mysqlhive_students.cou
ntry  | mysqlhive_students.partition  |
+-------------------------+-------------------------------+-----------------------
------+--------------------------------+
| 1                       | Schneider                     | Germany
      | 0                              |
| 2                       | Lacaux                        | France
      | 0                              |
| 3                       | Boniek                        | Poland
      | 0                              |
| 4                       | Gerets                        | Belgium
      | 0                              |
+-------------------------+-------------------------------+-----------------------
------+--------------------------------+
4 rows selected (3.29 seconds)
```

Everything was working fine ☺

Have fun.

DXC.technology