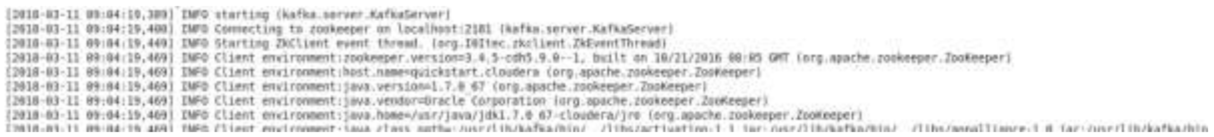


```
[2018-03-11 09:04:19,389] INFO starting (kafka.server.KafkaServer)
[2018-03-11 09:04:19,400] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2018-03-11 09:04:19,449] INFO Starting ZkClient event thread. (org.I0Itec.zkclient.ZkEventThread)
[2018-03-11 09:04:19,469] INFO Client environment:zookeeper.version=3.4.5-cdh5.9.0--1, built on 10/21/2016 08:05 GMT (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:host.name=quickstart.cloudera (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.version=1.7.0_67 (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.home=/usr/java/jdk1.7.0_67-cloudera/jre (org.apache.zookeeper.ZooKeeper)
[2018-03-11 09:04:19,469] INFO Client environment:java.class.path=:/usr/lib/kafka/bin/../libs/activation-1.1.jar:/usr/lib/kafka/bin/../libs/aopalliance-1.0.jar:/usr/lib/kafka/bin
```

Kafka broker is running on port 9042.

2. Open a new terminal window and create a Kafka topic named `weblogs` that will contain messages representing lines in `Loudacre's web server logs`.

Since your exercise environment is a single-node cluster running on a virtual machine, use a replication factor of 1 and a single partition.

Navigate to the location where Kafka is installed, which is "/usr/lib/kafka".

```
$ bin/kafka-topics.sh --create \ --zookeeper localhost:2181 \ --
replication-factor 1 \ --partitions 1 \
--topic weblogs
```

Here, Topic name

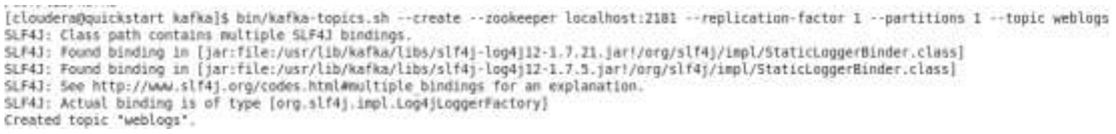
The name of the topic that you wish to create.

Replication Factor

The number of replicas of the topic to maintain within the cluster.

Partitions

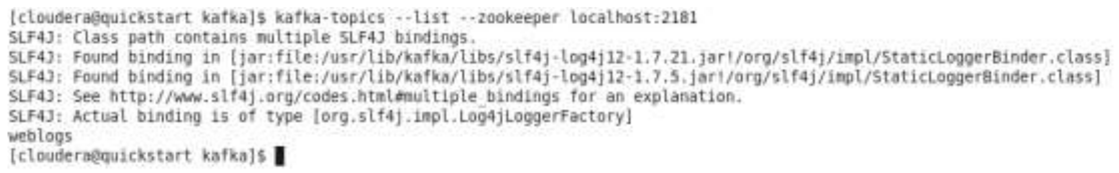
The number of partitions to create for the topic.



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "weblogs".
```

1. Display all Kafka topics to confirm that the new topic you just created is listed:

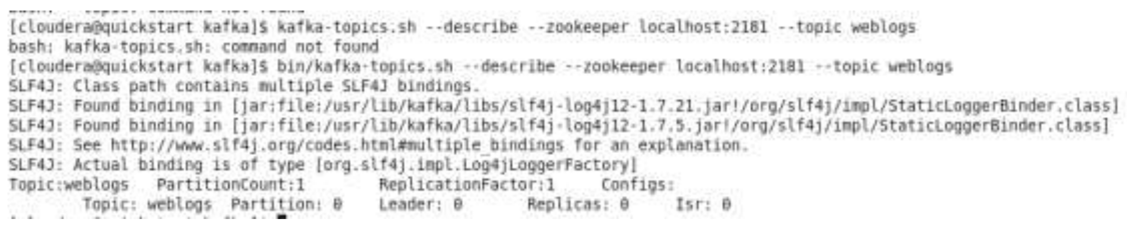
```
$ kafka-topics --list --zookeeper localhost:2181
```



```
[cloudera@quickstart kafka]$ kafka-topics --list --zookeeper localhost:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$
```

2. Describe the Kafka topic created

```
$ bin/kafka-topics.sh --describe --zookeeper localhost:2181

--topic weblogs
```



```
[cloudera@quickstart kafka]$ kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
bash: kafka-topics.sh: command not found
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:weblogs   PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: weblogs  Partition: 0    Leader: 0       Replicas: 0     Isr: 0
```

Producing and Consuming Messages

You will now use Kafka command line utilities to start producers and consumers for the topic created earlier.

3. Start a Kafka producer for the weblogs topic:

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic

weblogs
```

Tip : This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each one by selecting Set Title... on the Terminal menu:

Set the title for this window to "Kafka Producer."

4. Publish a test message to the weblogs topic by typing the message text and then pressing Enter. For example:

```
test weblog entry 1
```

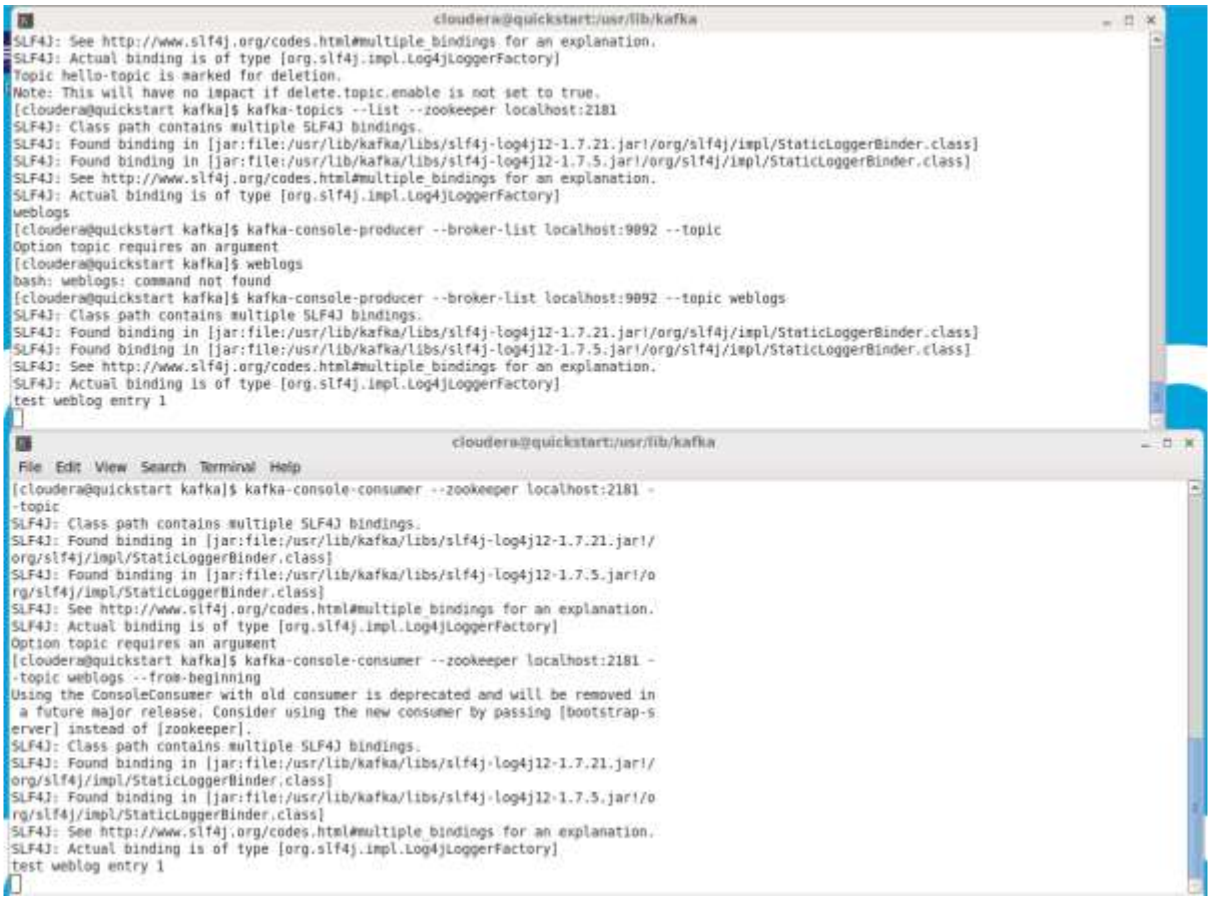
5. Open a new terminal window and adjust it to fit on the window beneath the producer window. Set the title for this window to "Kafka Consumer."

6. In the new terminal window, start a Kafka consumer that will read from the beginning of the weblogs topic:

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic

weblogs --from-beginning
```

You should see the status message you sent using the producer displayed on the consumer's console, such as:

```
test weblog 1
```


cloudera@quickstart:/usr/lib/kafka
cloudera@quickstart:/usr/lib/kafka
File Edit View Search Terminal Help

```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic hello-topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
[cloudera@quickstart kafka]$ kafka-topics --list --zookeeper localhost:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --topic
Option topic requires an argument
[cloudera@quickstart kafka]$ weblogs
bash: weblogs: command not found
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 1
```

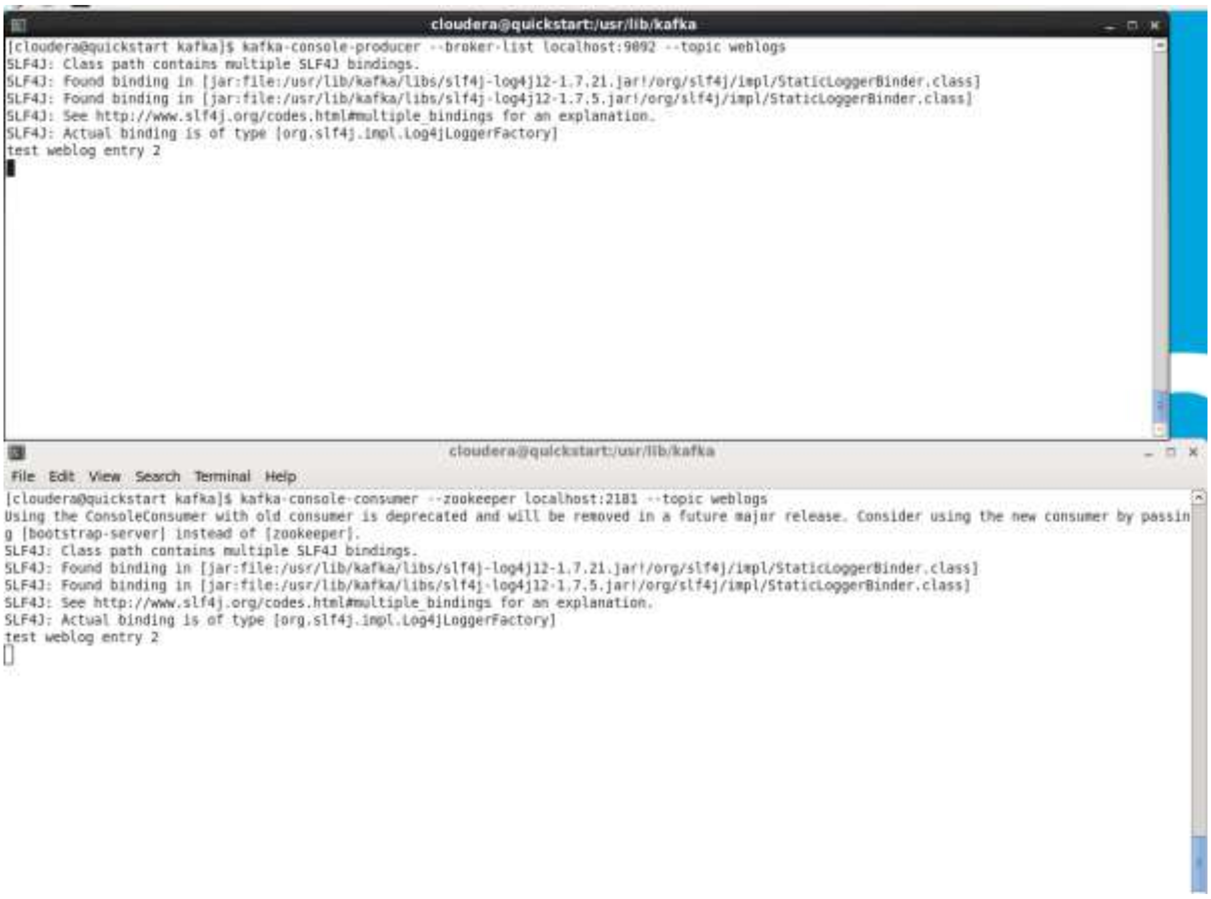
```
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 -
-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/o
rg/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Option topic requires an argument
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 -
-topic weblogs --from-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
 a future major release. Consider using the new consumer by passing [bootstrap-s
erver] instead of [zookeeper].
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/o
rg/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 1
```

7. Press Ctrl+C to stop the weblogs consumer, and restart it, but this time omit the from-beginning option to this command. You should see that no messages are

displayed.

8. Switch back to the producer window and type another test message into the terminal, followed by the Enter key:

```
test weblog entry 2
```



```
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 2
```

cloudera@quickstart:/usr/lib/kafka
File Edit View Search Terminal Help

```
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 --topic weblogs
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major release. Consider using the new consumer by passin
g [bootstrap-server] instead of [zookeeper].
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 2
```

9. Return to the consumer window and verify that it now displays the alert message you published from the producer in the previous step.

Cleaning Up

10. Press Ctrl+C in the consumer terminal window to end its process.

11. Press Ctrl+C in the producer terminal window to end its process.

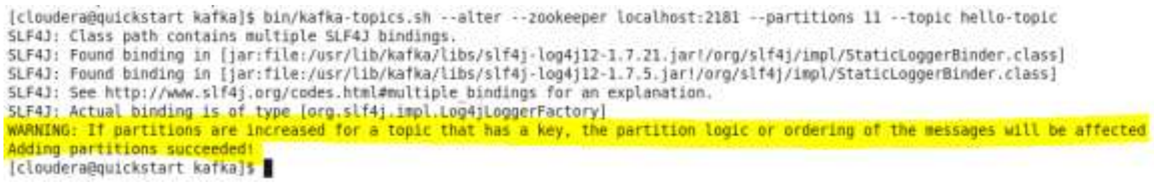
Exercise 16: Alter Apache Kafka Topics

Let us try altering some parameters for the topic, but instead of using the earlier created topic <weblogs>, we will create another topic “hello-topic” to alter some parameters.

Exercise: Create a topic called hello-topic with replication factor 1 and partition 1.

1. We will add more partitions in the topic created as “hello-topic”. Below command will add 10 more partitions to the hello-topic topic. Note that before, the topic has only 1 partition.

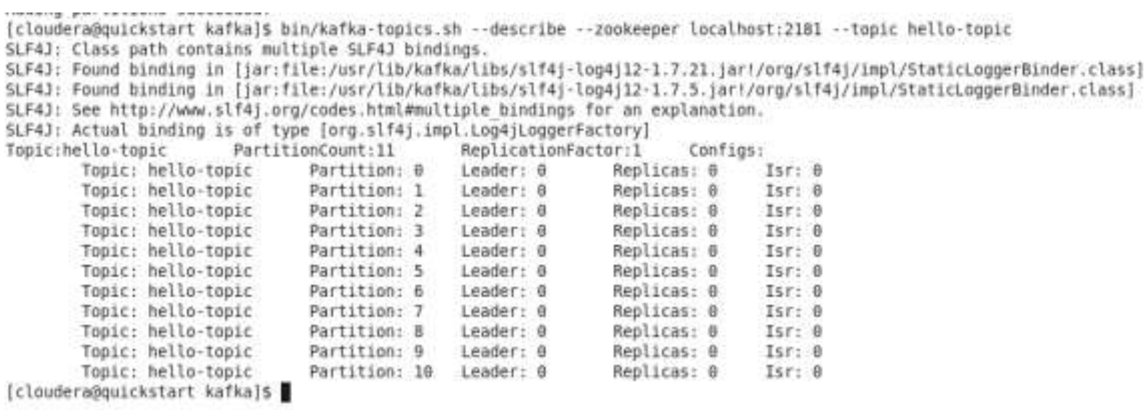
```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --
partitions 11 --topic hello-topic
```



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --alter --zookeeper localhost:2181 --partitions 11 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected
Adding partitions succeeded!
[cloudera@quickstart kafka]$ 
```

Let us verify the number of partitions is being changed with the command

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --
topic hello-topic
```



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:hello-topic       PartitionCount:11       ReplicationFactor:1     Configs:
        Topic: hello-topic      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 7    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 8    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 9    Leader: 0       Replicas: 0     Isr: 0
        Topic: hello-topic      Partition: 10   Leader: 0       Replicas: 0     Isr: 0
[cloudera@quickstart kafka]$ 
```

Tip: REDUCING PARTITION COUNTS

It is not possible to reduce the number of partitions for a topic. The reason this is not supported is because deleting a partition from a topic would cause part of the data in that topic to be deleted as well, which would be inconsistent from a client point of view. In addition, trying to redistribute the data to remaining partitions would be difficult and result in out-of-order messages. Should you need to reduce the number of partitions, you will need to delete the topic and recreate it.

2. Add configurations to the Kafka topic

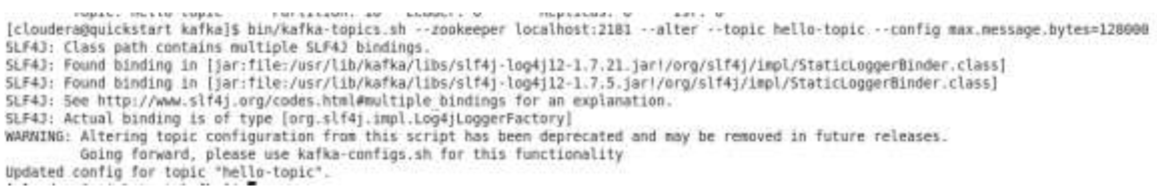
   The general syntax is:

```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --topic
kafkatopic --config <key>=<value>
```

There are various configuration fields we can set for the topic, here we will set up the

max.message.bytes - this is the largest size of the message the broker will allow to be appended to the topic. This size is validated pre-compression. (Defaults to broker's message.max.bytes.)

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --config max.message.bytes=128000
```



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic hello-topic --config max.message.bytes=128000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: Altering topic configuration from this script has been deprecated and may be removed in future releases.
         Going forward, please use kafka-configs.sh for this functionality
Updated config for topic "hello-topic".
```

3. To remove above overridden configuration, we can use command:

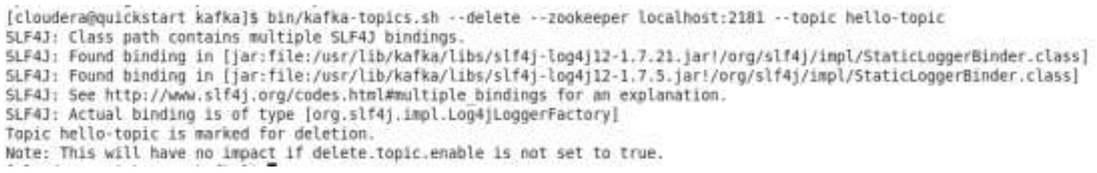
```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --delete-config max.message.bytes
```

Exercise 18: Delete a topic

If a topic is no longer needed, it can be deleted in order to free up these resources. In order to perform this action, the brokers in the cluster must have been configured with the delete.topic.enable option set to true. If this option has been set to false, then the request to delete the topic will be ignored.

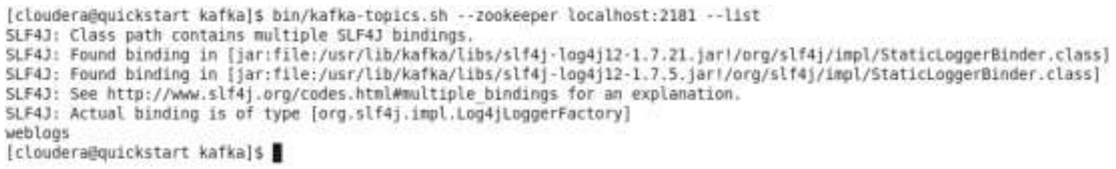
We will delete the topic created with the below command

```
bin/kafka-topics.sh --delete --zookeeper localhost:2181 --
topic hello-topic
```



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic hello-topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Lets verify if the topic has been deleted, with the –list command



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --list
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$ █
```

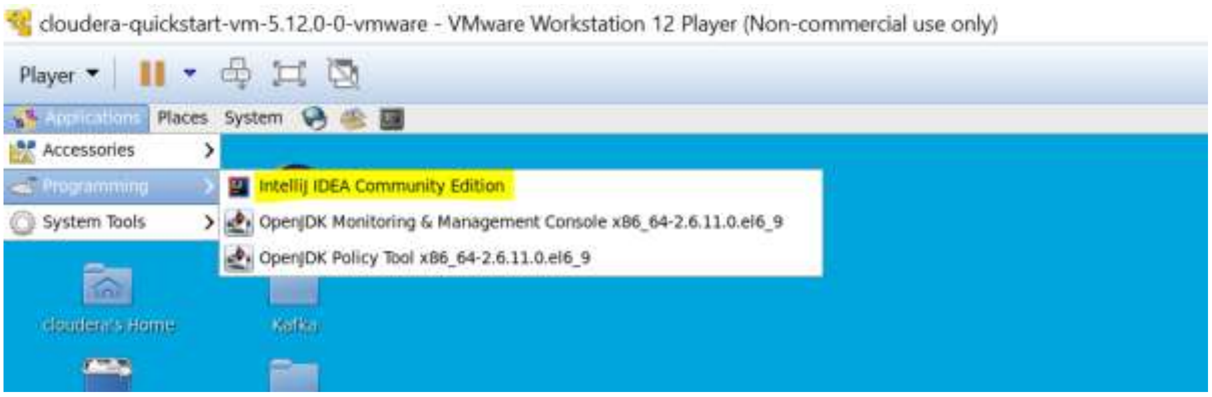
Tip: DATA LOSS AHEAD

Deleting a topic will also delete all its messages. This is not a reversible operation, so make sure it executed carefully.

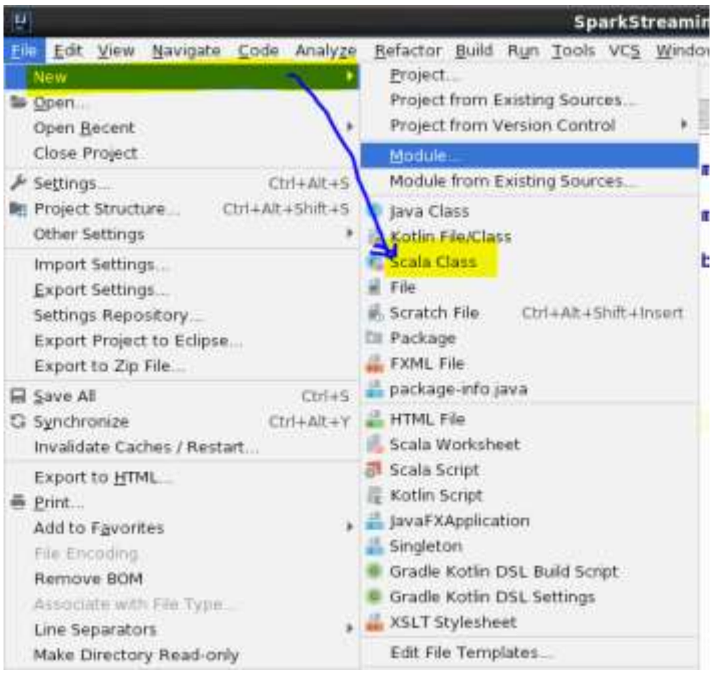
Exercise 19: Building Spark Application using IntelliJ IDE Apache Kafka Scala Client API

Goal: We are going to create a producer and consumer by using **Apache Kafka Scala client API.**

1. Create new SBT Project using the menu bar option File –>New Project -> Select Scala and SBT. Give a name of your choice for the project.


cloudera-quickstart-vm-5.12.0-0-vmware - VMware Workstation 12 Player (Non-commercial use only)
Player
Applications
Places
System
Accessories
Programming
System Tools
IntelliJ IDEA Community Edition
OpenJDK Monitoring & Management Console x86_64-2.6.11.0.el6_9
OpenJDK Policy Tool x86_64-2.6.11.0.el6_9
cloudera's Home
Kafka

Under src→main→scala, Add a new Scala class.



2. The content of build.sbt

```
version := "1.0"

scalaVersion := "2.11.6"

libraryDependencies ++= Seq (

"org.apache.kafka" %"kafka-clients" %"0.11.0.0

)
```

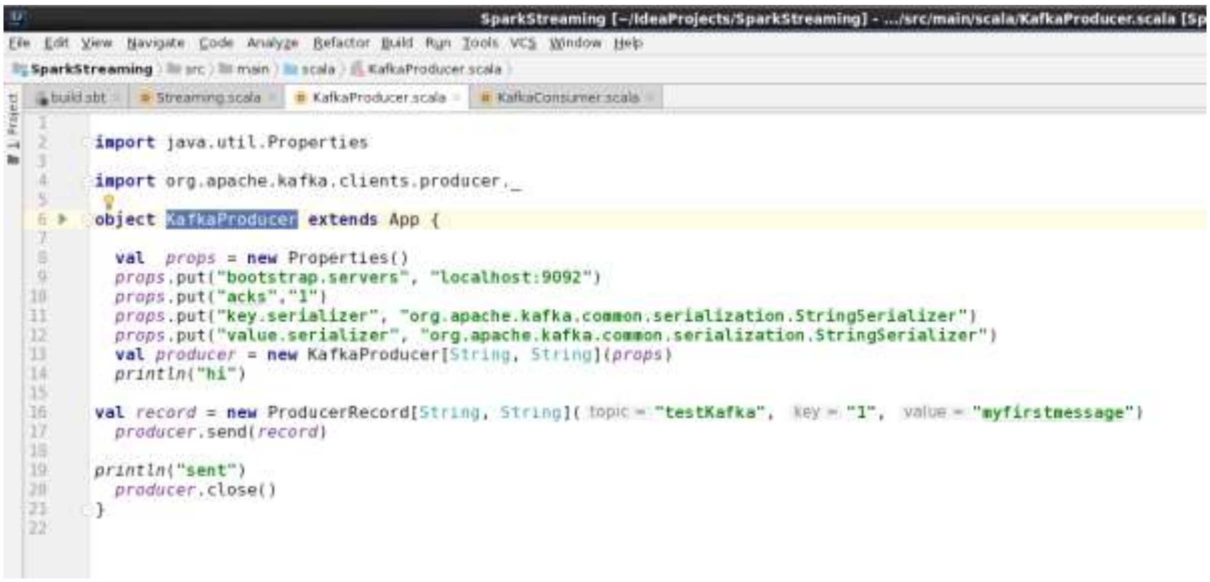
Note: A pop up of “Enable Auto Import”, please click on it to enable it.

3. Create a KafkaProducer Scala Object, as below:-

4. Now add a new Scala class file to project, Right click on project > New > ScalaClass > KafkaProducerTest.

```
object KafkaProducerTest extends App {


   import Scala.util.Properties


    import org.apache.kafka.clients.producer._
```

```
        val  props = new Properties()
          props.put("bootstrap.servers", "localhost:9092")
            props.put("acks","1")
              props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
                props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

                  val producer = new KafkaProducer[String, String](props)

                    val topic="testKafka"
                      for(i<- 1 to 50) {
                              val record = new ProducerRecord(topic, "key"+i, "value"+i)
                                  producer.send(record)
                                    }
                                println("sent")
                                      producer.close()
}
```


SparkStreaming [~/IdeaProjects/SparkStreaming] - .../src/main/scala/KafkaProducer.scala [Sp
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SparkStreaming src main scala KafkaProducer.scala
build.sbt Streaming.scala KafkaProducer.scala KafkaConsumer.scala
1: Project

```
import java.util.Properties

import org.apache.kafka.clients.producer._

object KafkaProducer extends App {

  val  props = new Properties()
  props.put("bootstrap.servers", "localhost:9092")
  props.put("acks","1")
  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  val producer = new KafkaProducer[String, String](props)
  println("hi")

val record = new ProducerRecord[String, String]( topic = "testKafka",  key = "1",  value = "myfirstmessage")
  producer.send(record)

println("sent")
  producer.close()
}
```

<u>A Kafka producer has three mandatory properties</u>:

<u>bootstrap.servers</u>

List of host:port pairs of brokers that the producer will use to establish initial connection to the Kafka cluster. This list doesn't need to include all brokers, since the producer will get more information after the initial connection. But it is recommended to include at least two, so in case one broker goes down, the producer will still be able to connect to the cluster.

key.serializer

Name of a class that will be used to serialize the keys of the records we will produce to Kafka. Kafka brokers expect byte arrays as keys and values of messages. Tthe producer has to know how to convert these objects to byte arrays. key.serializer should be set to a name of a class that implements the org.apache.kafka.common.serialization.Serializer interface. The producer will use this class to serialize the key object to a byte array.

The Kafka client package includes

ByteArraySerializer (which doesn't do much),

StringSerializer, and

IntegerSerializer, so if you use common types, there is no need to implement your own serializers.

Setting key.serializer is required even if you intend to send only values.

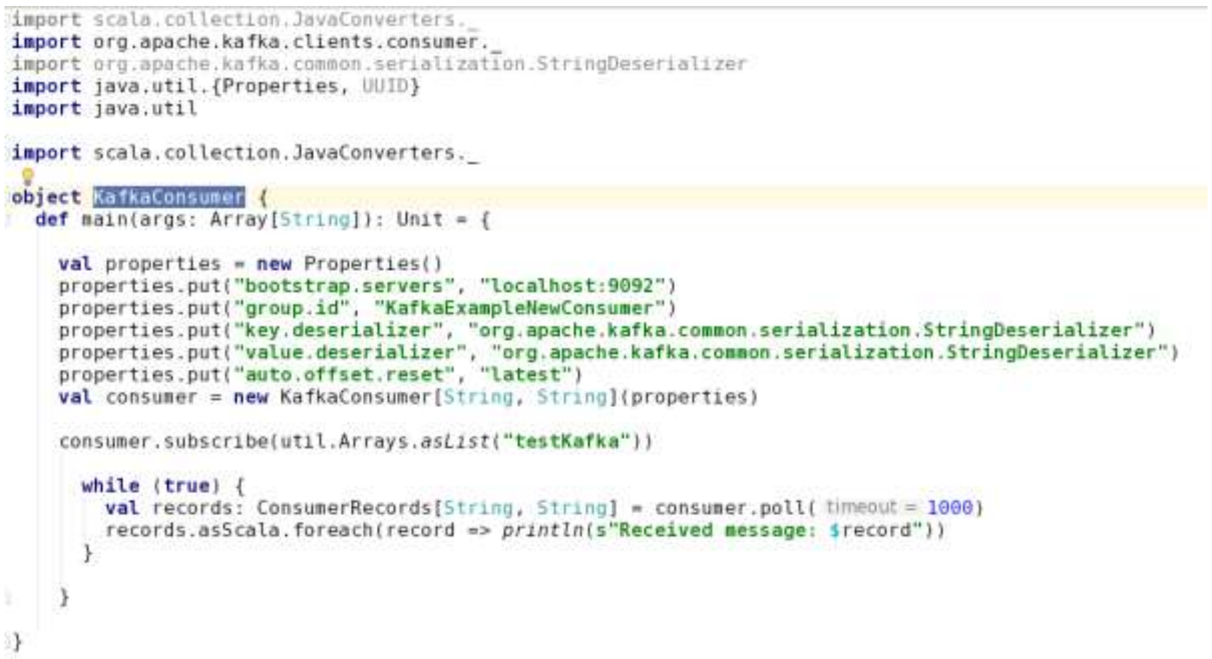
value.serializer

Name of a class that will be used to serialize the values of the records we will produce to Kafka. The same way you set key.serializer to a name of a class that will serialize the message key object to a byte array, you set value.serializer to a class that will serialize the message value object.

5. Now let us add a new Scala class file to project, Right click on project >New > Class > KafkaConsumerTest

```
import scala.collection.ScalaConverters._

import Scala.util

import Scala.util.Properties


import org.apache.kafka.clients.consumer.{ConsumerRecords, KafkaConsumer}


object KafkaConsumerTest extends App {
```

```
val properties = new Properties()

properties.put("bootstrap.servers", "localhost:9092")

properties.put("group.id", "KafkaExampleNewConsumer")

properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")

properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")

properties.put("auto.offset.reset", "latest")

val consumer = new KafkaConsumer[String, String](properties)


consumer.subscribe(util.Arrays.asList("testKafka"))


while (true) {

  val records: ConsumerRecords[String, String] = consumer.poll(1000)

  records.asScala.foreach(record => println(s"Received message: "+record))

}


}
```



```
import scala.collection.JavaConverters._
import org.apache.kafka.clients.consumer._
import org.apache.kafka.common.serialization.StringDeserializer
import java.util.{Properties, UUID}
import java.util

import scala.collection.JavaConverters._

object KafkaConsumer {
  def main(args: Array[String]): Unit = {

    val properties = new Properties()
    properties.put("bootstrap.servers", "localhost:9092")
    properties.put("group.id", "KafkaExampleNewConsumer")
    properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    properties.put("auto.offset.reset", "latest")
    val consumer = new KafkaConsumer[String, String](properties)

    consumer.subscribe(util.Arrays.asList("testKafka"))

      while (true) {
        val records: ConsumerRecords[String, String] = consumer.poll( timeout = 1000)
        records.asScala.foreach(record => println(s"Received message: $record"))
      }

    }

}
```

6. Now we will test our KafkaProducerTest sending some messages, which will be received by KafkaConsumerTest, before testing ensure that

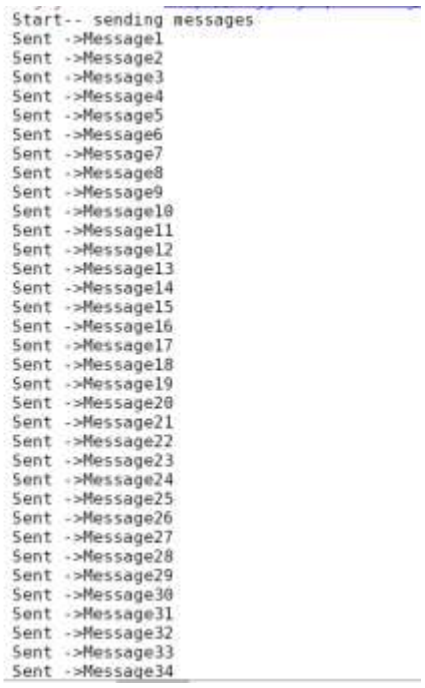
Pre-requisites:-

Topic 'testKafka' is created.

Zookeeper should be running on localhost:2181.

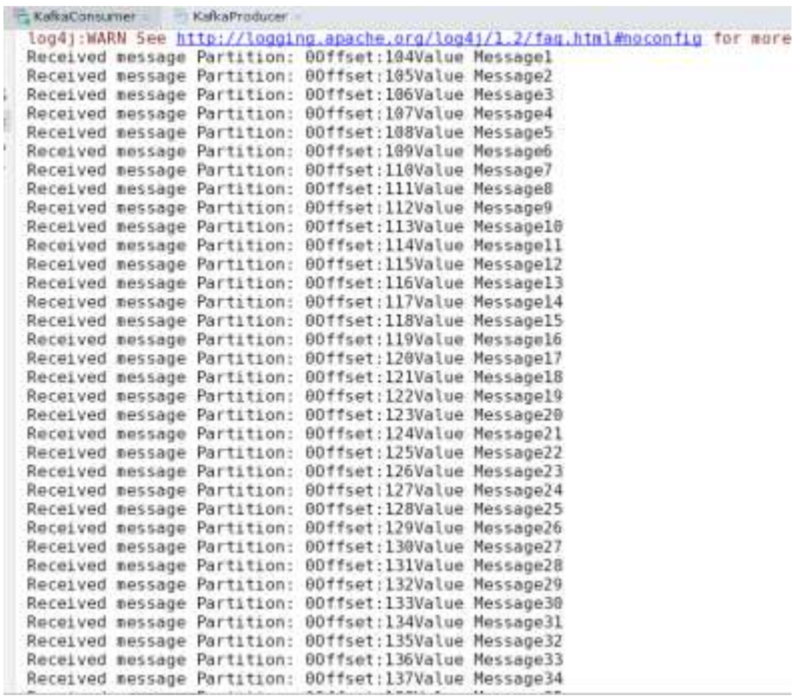
Kafka should be running on localhost:9092.

Run ProducerTest.Scala > Scala Application.



```
Start-- sending messages
Sent ->Message1
Sent ->Message2
Sent ->Message3
Sent ->Message4
Sent ->Message5
Sent ->Message6
Sent ->Message7
Sent ->Message8
Sent ->Message9
Sent ->Message10
Sent ->Message11
Sent ->Message12
Sent ->Message13
Sent ->Message14
Sent ->Message15
Sent ->Message16
Sent ->Message17
Sent ->Message18
Sent ->Message19
Sent ->Message20
Sent ->Message21
Sent ->Message22
Sent ->Message23
Sent ->Message24
Sent ->Message25
Sent ->Message26
Sent ->Message27
Sent ->Message28
Sent ->Message29
Sent ->Message30
Sent ->Message31
Sent ->Message32
Sent ->Message33
Sent ->Message34
```

Right click on the scala class→Run KafkaConsumerTest.scala, the console of the receiver will receive all the messages from the broker. The console will print the key and the message.



```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
Received message Partition: 0Offset:104Value Message1
Received message Partition: 0Offset:105Value Message2
Received message Partition: 0Offset:106Value Message3
Received message Partition: 0Offset:107Value Message4
Received message Partition: 0Offset:108Value Message5
Received message Partition: 0Offset:109Value Message6
Received message Partition: 0Offset:110Value Message7
Received message Partition: 0Offset:111Value Message8
Received message Partition: 0Offset:112Value Message9
Received message Partition: 0Offset:113Value Message10
Received message Partition: 0Offset:114Value Message11
Received message Partition: 0Offset:115Value Message12
Received message Partition: 0Offset:116Value Message13
Received message Partition: 0Offset:117Value Message14
Received message Partition: 0Offset:118Value Message15
Received message Partition: 0Offset:119Value Message16
Received message Partition: 0Offset:120Value Message17
Received message Partition: 0Offset:121Value Message18
Received message Partition: 0Offset:122Value Message19
Received message Partition: 0Offset:123Value Message20
Received message Partition: 0Offset:124Value Message21
Received message Partition: 0Offset:125Value Message22
Received message Partition: 0Offset:126Value Message23
Received message Partition: 0Offset:127Value Message24
Received message Partition: 0Offset:128Value Message25
Received message Partition: 0Offset:129Value Message26
Received message Partition: 0Offset:130Value Message27
Received message Partition: 0Offset:131Value Message28
Received message Partition: 0Offset:132Value Message29
Received message Partition: 0Offset:133Value Message30
Received message Partition: 0Offset:134Value Message31
Received message Partition: 0Offset:135Value Message32
Received message Partition: 0Offset:136Value Message33
Received message Partition: 0Offset:137Value Message34
```

Terminate the KafkaConsumerTest.Scala class.

Assignment: In this assignment, we will use Sending a Message Synchronously

1. Create a Producer class as “KafkaPublisher” for the topic “topic-1”.
2. Imports
   import org.apache.kafka.clients.consumer.ConsumerRecords;

   import org.apache.kafka.clients.consumer.ConsumerRecord;

   import org.apache.kafka.clients.producer.ProducerRecord;
3. Set the mandatory values for the KafkaPublisher, which is the producer class
   bootstrap.servers=broker2:9092

   key.serializer=org.apache.kafka.common.serialization.StringSerializer

   value.serializer=org.apache.kafka.common.serialization.StringSerializer
4. Set the values in a Properties object.
5. Instead of Fire-and-forget which was done before we will use the synchronous way of sending the message , let us send a message, the send() method returns a Future object, and we use get() to wait on the future and see if the send() was successful or not.
6. The message record is created as

val record: ProducerRecord[String, String] = new ProducerRecord(TOPIC, inlineMessage)

The ProducerRecord objects we created included a topic name and value.

7. Send the message i.e. record using
   val futureResponse: Future[RecordMetadata] = producer.send(record)

   futureResponse.isDone

   in the KafkaPublisher.Scala class.

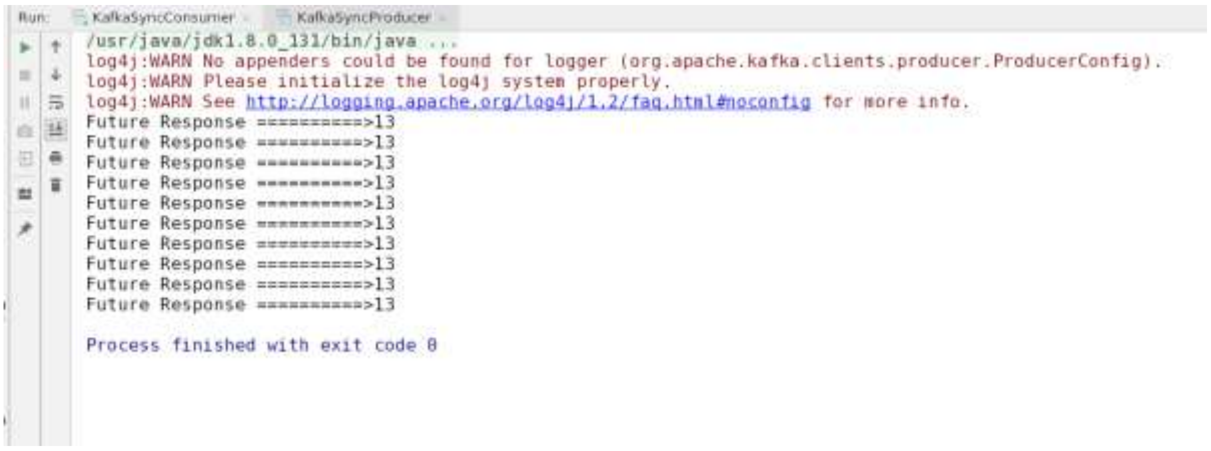
8. Create a Consumer class as KafkaSubscriber.
9. Set the three mandatory properties: bootstrap.servers, key.deserializer, and value.deserializer (deserializer should be of same type as of serializer).
10. Now set the fourth property, which is not strictly mandatory, but for now we will pretend it is. The property is group.id and it specifies the consumer group the KafkaSubscriber instance belongs to.

    props.put("group.id", "testgroup")

11. Poll the message on the topic mytopic using the poll() method

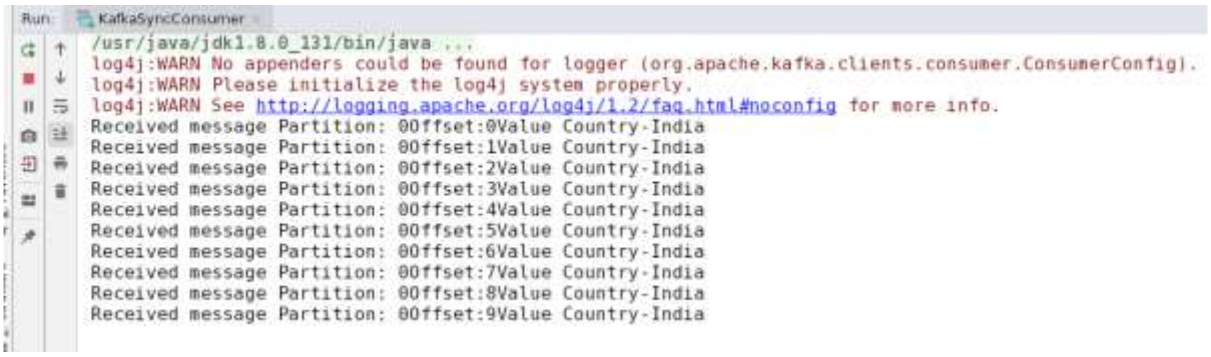
12. Run the KafkaPublisher as Scala Application and also run the KafkaSubscriber as Scala Application.
13. The received message will be printed in the console.

Producer



```
Run:   KafkaSyncConsumer   KafkaSyncProducer
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13
Future Response ==========>13

Process finished with exit code 0
```

Consumer



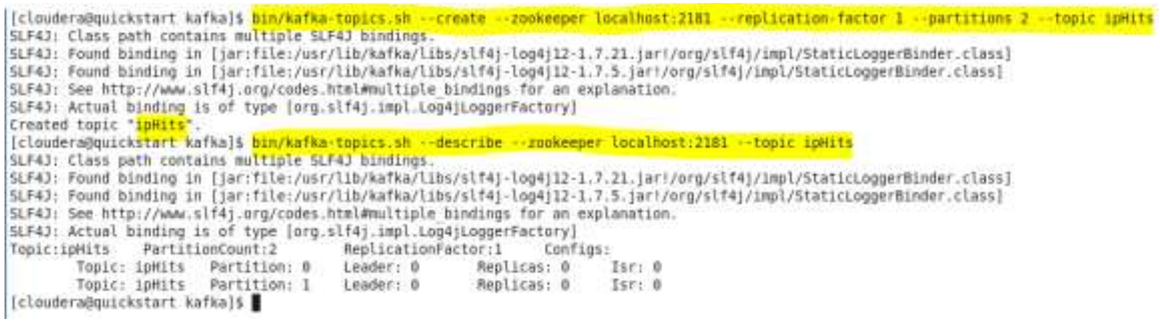
```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 0Offset:0Value Country-India
Received message Partition: 0Offset:1Value Country-India
Received message Partition: 0Offset:2Value Country-India
Received message Partition: 0Offset:3Value Country-India
Received message Partition: 0Offset:4Value Country-India
Received message Partition: 0Offset:5Value Country-India
Received message Partition: 0Offset:6Value Country-India
Received message Partition: 0Offset:7Value Country-India
Received message Partition: 0Offset:8Value Country-India
Received message Partition: 0Offset:9Value Country-India
```

Exercise 20: Implementing a Custom Partitioning Strategy

We can create a class implementing the org.apache.kafka.clients.producer.Partitioner interface. This custom Partitioner will implement the business logic to decide where messages are sent.

Use Case: Let's jump to the next level by writing another program that implements customized message partitioning. The example consists of recollecting the IPs visiting a web site, which are recorded and published. The message has three parts: web site name, and IP address.

Let us create a topic with two partitions.



```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic ipHits
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "ipHits".
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic ipHits
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:ipHits    PartitionCount:2        ReplicationFactor:1     Configs:
        Topic: ipHits   Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: ipHits   Partition: 1    Leader: 0       Replicas: 0     Isr: 0
[cloudera@quickstart kafka]$ 
```

Let us create a CountryPartitioner that implements the org.apache.kafka.clients.producer.Partitioner interface as Scala class.

1. Create a SBT project and define all the dependencies.
2. Create a Scala class as CountryPartitioner

```
import kafka.producer._


import Scala.util


import org.apache.kafka.clients.producer.KafkaProducer

import org.apache.kafka.clients.producer.Partitioner

import org.apache.kafka.common.Cluster
```

```
object CountryPartitioner  {

  private var producer: KafkaProducer[String, String] = _
}

class CountryPartitioner  extends Partitioner {

  def partition(key: AnyRef, a_numPartitions: Int): Int = {
    var partition = 1
    val partitionKey = key.asInstanceOf[String]
    val offset = partitionKey

    if (offset == "USA") {
      partition = 0
      println("its"+offset +"and its partition is "+partition)
    } else if (offset == "INDIA") {
      partition = 1
      println("its"+offset +"and its partition is "+partition)
    }
    partition
  }

  override def partition(topic: String,
                         key: AnyRef,
                         keyBytes: Array[Byte],
                         value: AnyRef,
                         valueBytes: Array[Byte],
                         cluster: Cluster): Int = partition(key, 10)
  override def close() {

  }


  override def configure(configs: util.Map[String, _]) {

  }

}
```

Note: We set a config property with a key equal to the value of **ProducerConfig.PARTITIONER_CLASS_CONFIG**, which matches the fully qualified name of our **CountryPartitioner** class. We also set countryName to partitionId, thus mapping the properties that we want to pass to CountryPartitioner.

3. Now let us create the Producer class

```
import Scala.util.Properties

import org.apache.kafka.clients.producer._


object KafkaProducerwithPartitioner extends App {

  val  props = new Properties()

  props.put("bootstrap.servers", "localhost:9092")

  props.put("acks","1")

  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")

  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")


  props.put("partitioner.class","CountryPartitioner")


  val producer = new KafkaProducer[String, String](props)

  println("Start-- sending messages")

  var ipCountry="INDIA"
  for(i <- 1 to 100) {
    if( i %2 == 0){
      ipCountry = "USA"


    }
    else{
      ipCountry = "INDIA"
    }


    val record = new ProducerRecord[String, String]("ipHits", ipCountry,"Message" +
ipCountry)
    producer.send(record)
    println("Sent ->Message" + i)
  }


  println("Sent--closing connection")
  producer.close()
}
```

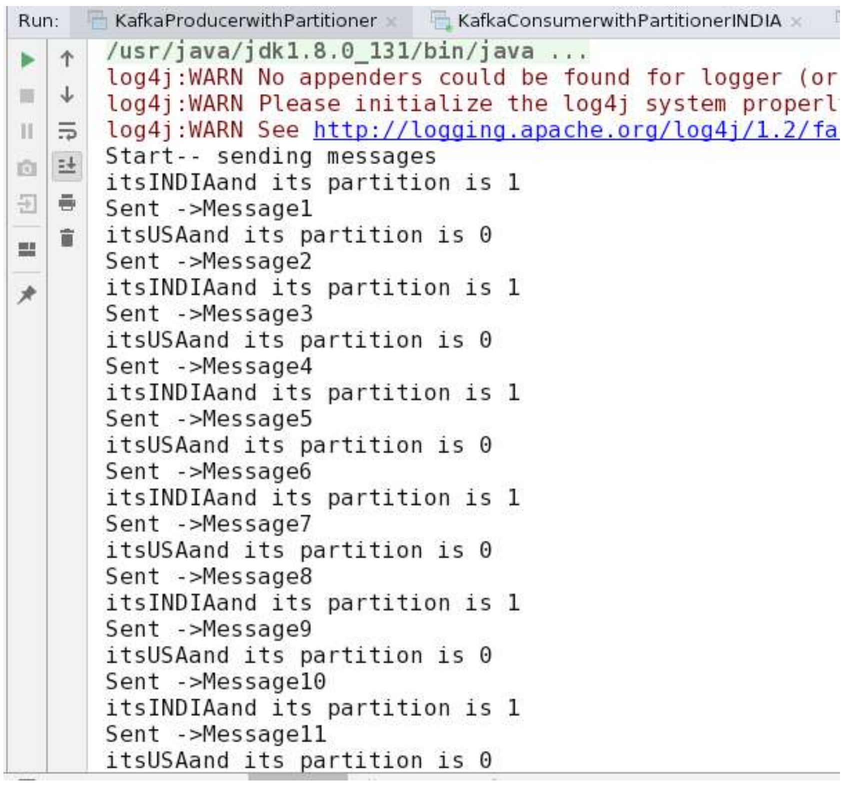
4. Let us create one partitioned consumer which will consume all messages from partition 1.

```
import org.apache.kafka.clients.consumer.ConsumerRecords

import org.apache.kafka.clients.consumer.KafkaConsumer
import Scala.util.{Collections, Properties}

import org.apache.kafka.common.TopicPartition


import scala.collection.ScalaConverters._


object KafkaConsumerwithPartitionerINDIA {

  def main(args: Array[String]): Unit = {


    val properties = new Properties()

    properties.put("bootstrap.servers", "localhost:9092")

    properties.put("group.id", "KafkaExampleNewConsumer2")

    properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("auto.offset.reset", "latest")

    val consumer = new KafkaConsumer[String, String](properties)


    //consumer.subscribe(util.Arrays.asList("testKafka"))

    val topicPartition = new TopicPartition("ipHits",1)

    consumer.assign(Collections.singletonList(topicPartition))


    while (true) {

      val records: ConsumerRecords[String, String] = consumer.poll(1000)

      records.asScala.foreach(record => println(s"Received message Partition:
"+record.partition() +"Offset:" +record.offset() +"Value "+record.value()))

    }


  }

}
```

5. Let us create one partitioned consumer which will consume all messages from partition 0.

```
import org.apache.kafka.clients.consumer._

import Scala.util.{Collections, Properties}

import org.apache.kafka.common.TopicPartition


import scala.collection.ScalaConverters._


object KafkaConsumerwithPartitionerUSA {

  def main(args: Array[String]): Unit = {

    val properties = new Properties()

    properties.put("bootstrap.servers", "localhost:9092")

    properties.put("group.id", "KafkaExampleNewConsumer")

    properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer")

    properties.put("auto.offset.reset", "latest")

    val consumer = new KafkaConsumer[String, String](properties)

    val topicPartition = new TopicPartition("ipHits",0)

    consumer.assign(Collections.singletonList(topicPartition))

      while (true) {

        val records: ConsumerRecords[String, String] = consumer.poll(1000)

        records.asScala.foreach(record => println(s"Received message Partition:
"+record.partition() +"Offset:" +record.offset() +"Value "+record.value()))

      }

    }

}
```

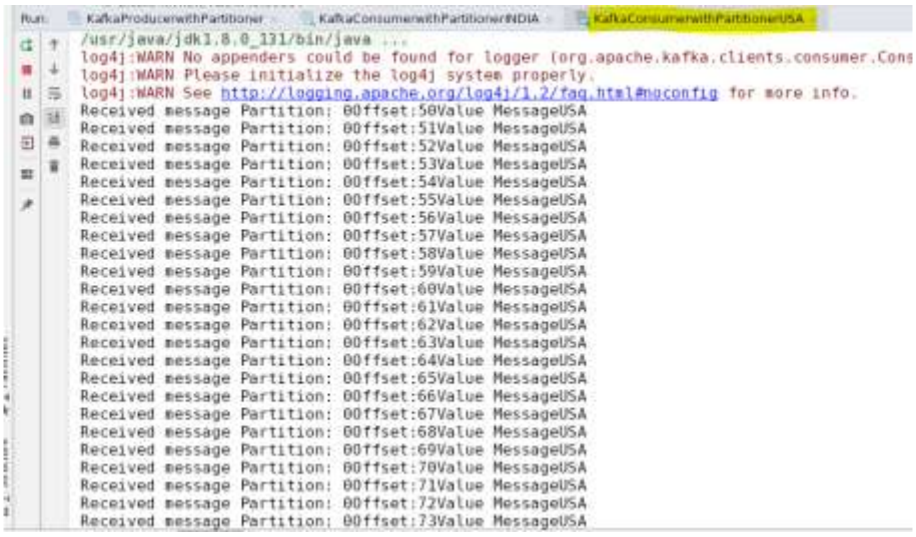
6. Start a producer as Scala Application.


Run:
KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (or
log4j:WARN Please initialize the log4j system properl
log4j:WARN See http://logging.apache.org/log4j/1.2/fa
Start-- sending messages
itsINDIAand its partition is 1
Sent ->Message1
itsUSAand its partition is 0
Sent ->Message2
itsINDIAand its partition is 1
Sent ->Message3
itsUSAand its partition is 0
Sent ->Message4
itsINDIAand its partition is 1
Sent ->Message5
itsUSAand its partition is 0
Sent ->Message6
itsINDIAand its partition is 1
Sent ->Message7
itsUSAand its partition is 0
Sent ->Message8
itsINDIAand its partition is 1
Sent ->Message9
itsUSAand its partition is 0
Sent ->Message10
itsINDIAand its partition is 1
Sent ->Message11
itsUSAand its partition is 0
```

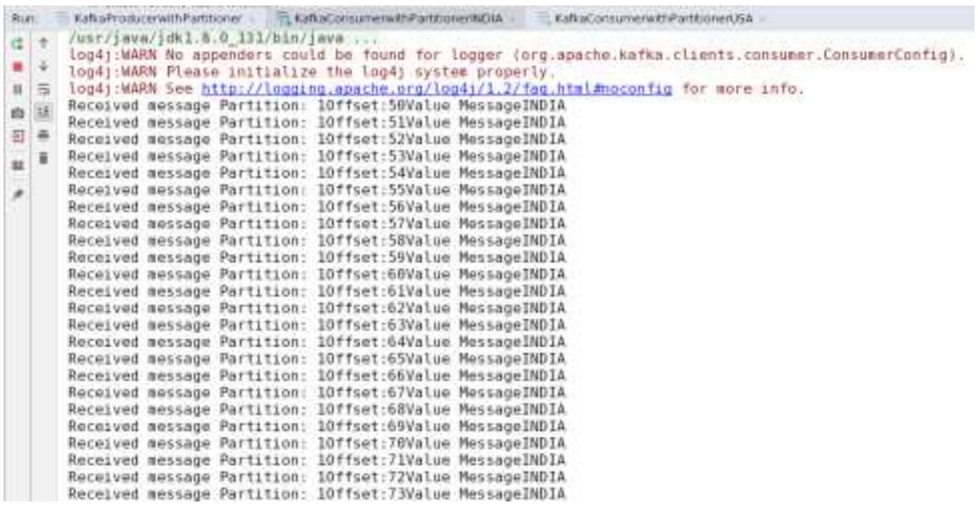
Note: Observe the value of the offset for message send to each partition.

7. Start the consumers as Scala Application.
   Navigate to <consumer-console>. Consumer for partition 0.


Run: KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA
KafkaConsumerwithPartitionerUSA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.Cons
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 0Offset:50Value MessageUSA
Received message Partition: 0Offset:51Value MessageUSA
Received message Partition: 0Offset:52Value MessageUSA
Received message Partition: 0Offset:53Value MessageUSA
Received message Partition: 0Offset:54Value MessageUSA
Received message Partition: 0Offset:55Value MessageUSA
Received message Partition: 0Offset:56Value MessageUSA
Received message Partition: 0Offset:57Value MessageUSA
Received message Partition: 0Offset:58Value MessageUSA
Received message Partition: 0Offset:59Value MessageUSA
Received message Partition: 0Offset:60Value MessageUSA
Received message Partition: 0Offset:61Value MessageUSA
Received message Partition: 0Offset:62Value MessageUSA
Received message Partition: 0Offset:63Value MessageUSA
Received message Partition: 0Offset:64Value MessageUSA
Received message Partition: 0Offset:65Value MessageUSA
Received message Partition: 0Offset:66Value MessageUSA
Received message Partition: 0Offset:67Value MessageUSA
Received message Partition: 0Offset:68Value MessageUSA
Received message Partition: 0Offset:69Value MessageUSA
Received message Partition: 0Offset:70Value MessageUSA
Received message Partition: 0Offset:71Value MessageUSA
Received message Partition: 0Offset:72Value MessageUSA
Received message Partition: 0Offset:73Value MessageUSA
```

Consumer for partition 1.


Run: KafkaProducerwithPartitioner
KafkaConsumerwithPartitionerINDIA
KafkaConsumerwithPartitionerUSA

```
/usr/java/jdk1.8.0_131/bin/java ...
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Received message Partition: 1Offset:50Value MessageINDIA
Received message Partition: 1Offset:51Value MessageINDIA
Received message Partition: 1Offset:52Value MessageINDIA
Received message Partition: 1Offset:53Value MessageINDIA
Received message Partition: 1Offset:54Value MessageINDIA
Received message Partition: 1Offset:55Value MessageINDIA
Received message Partition: 1Offset:56Value MessageINDIA
Received message Partition: 1Offset:57Value MessageINDIA
Received message Partition: 1Offset:58Value MessageINDIA
Received message Partition: 1Offset:59Value MessageINDIA
Received message Partition: 1Offset:60Value MessageINDIA
Received message Partition: 1Offset:61Value MessageINDIA
Received message Partition: 1Offset:62Value MessageINDIA
Received message Partition: 1Offset:63Value MessageINDIA
Received message Partition: 1Offset:64Value MessageINDIA
Received message Partition: 1Offset:65Value MessageINDIA
Received message Partition: 1Offset:66Value MessageINDIA
Received message Partition: 1Offset:67Value MessageINDIA
Received message Partition: 1Offset:68Value MessageINDIA
Received message Partition: 1Offset:69Value MessageINDIA
Received message Partition: 1Offset:70Value MessageINDIA
Received message Partition: 1Offset:71Value MessageINDIA
Received message Partition: 1Offset:72Value MessageINDIA
Received message Partition: 1Offset:73Value MessageINDIA
```

Exercise 17: Spark Streaming

Let’s start with simple streaming example, to count the number of words in text data received from a data server listening on a TCP socket.

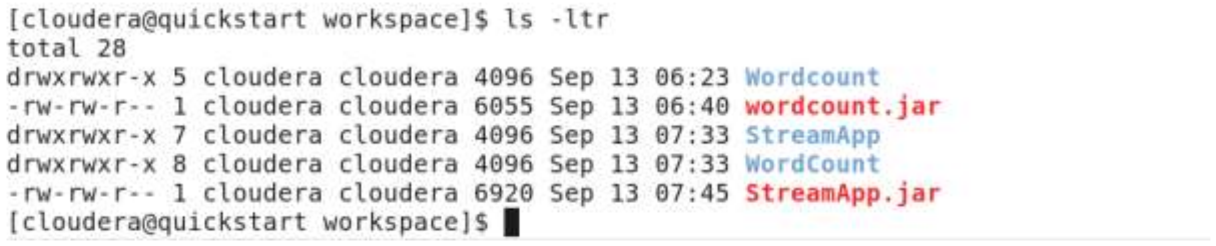
Create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

Create a Scala project with below inputs:

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext._

import org.apache.spark.streaming.StreamingContext

import org.apache.spark.streaming.Seconds

import org.apache.spark.storage.StorageLevel;

import org.apache.spark.streaming.StreamingContext._

object SparkStreamReceiver {

  def main(args: Array[String]): Unit = {

    var port:Int = 9999;

    val conf = new SparkConf()

                .setMaster("local[*]")

                .setAppName("Simple spark streaming")


    val ssc = new StreamingContext(conf, Seconds(3));

    val dstream = ssc.socketTextStream("localhost", port,
StorageLevel.MEMORY_ONLY);

    val words = dstream.flatMap(_.split(" "))

    val pairs = words.map(word => (word, 1))

    val wordcounts = pairs.reduceByKey(_ + _)

    wordcounts.print();

    ssc.start();

    ssc.awaitTermination();

  }

}
```

```

```

Export the jar file i.e. StreamApp.jar

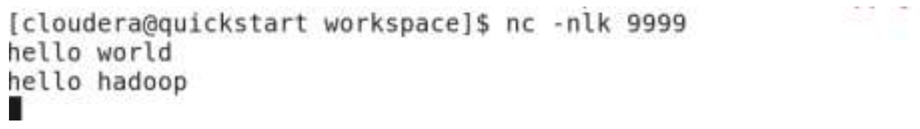


```
[cloudera@quickstart workspace]$ ls -ltr
total 28
drwxrwxr-x 5 cloudera cloudera 4096 Sep 13 06:23 Wordcount
-rw-rw-r-- 1 cloudera cloudera 6055 Sep 13 06:40 wordcount.jar
drwxrwxr-x 7 cloudera cloudera 4096 Sep 13 07:33 StreamApp
drwxrwxr-x 8 cloudera cloudera 4096 Sep 13 07:33 WordCount
-rw-rw-r-- 1 cloudera cloudera 6920 Sep 13 07:45 StreamApp.jar
[cloudera@quickstart workspace]$ █
```

Run the program using spark-submit

Run Netcat command and open another terminal to submit the job

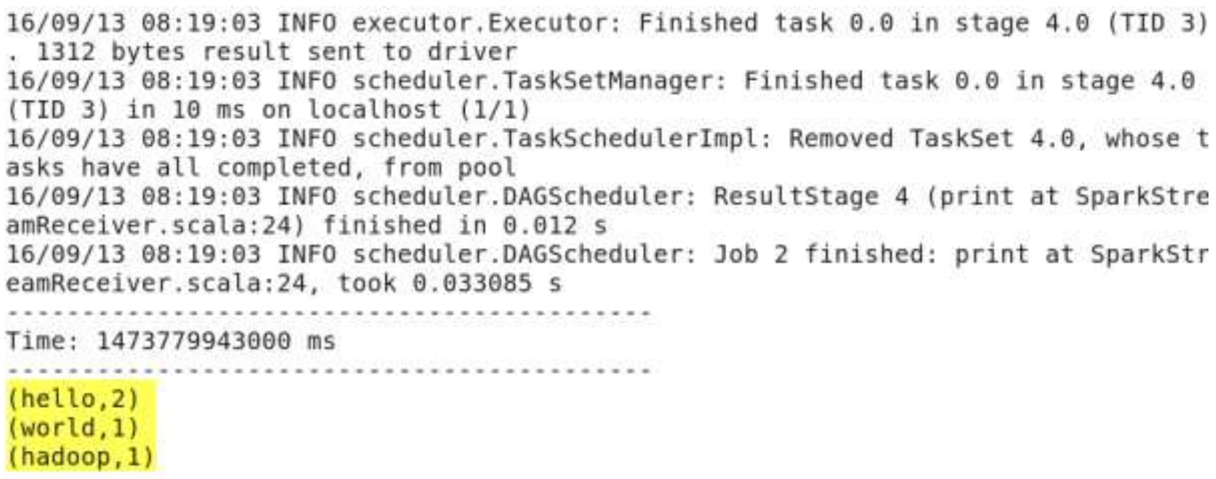
```
nc -nlk 9999
```



```
[cloudera@quickstart workspace]$ nc -nlk 9999
hello world
hello hadoop
█
```

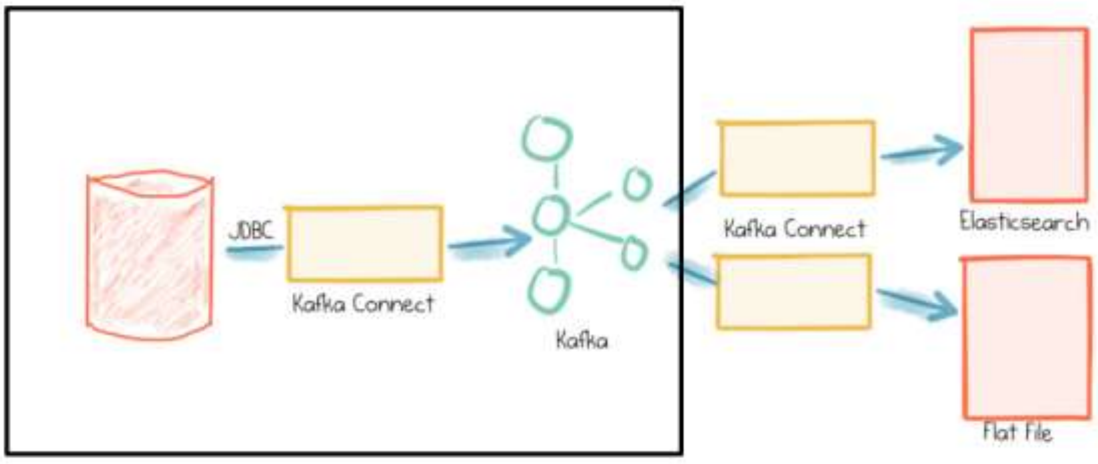
```
bin/spark-submit –master yarn --class solution.StreamingExample
/home/cloudera/eclipse-workspace/Lesson4/target/sfpdapp-1.0.jar
```

Any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following



```
16/09/13 08:19:03 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 3)
. 1312 bytes result sent to driver
16/09/13 08:19:03 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 3) in 10 ms on localhost (1/1)
16/09/13 08:19:03 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
16/09/13 08:19:03 INFO scheduler.DAGScheduler: ResultStage 4 (print at SparkStre
amReceiver.scala:24) finished in 0.012 s
16/09/13 08:19:03 INFO scheduler.DAGScheduler: Job 2 finished: print at SparkStr
eamReceiver.scala:24, took 0.033085 s
-------------------------------------------
Time: 1473779943000 ms
-------------------------------------------
(hello,2)
(world,1)
(hadoop,1)
```

Exercise 18: Apache Kafka Connector Example – Import Data from MySQL into Kafka



We are going to take care of the black box, as surrounded in the diagram above.

The Confluent JDBC Connector for Kafka Connect enables you to stream data to and from Kafka and any RDBMS that supports JDBC (which is to say pretty much any). It can stream entire schemas or just individual tables.

Pre-requisites to work with Kafka connect

1. Confluent should be installed
2. Java version 1.8 should be installed
3. Mysql database should be installed
4. Kafka and Schema Registry are running locally on the default ports.

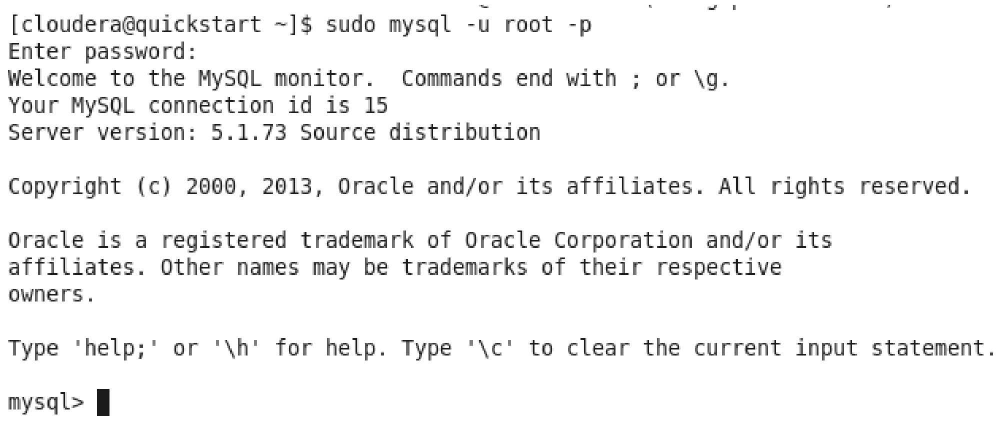
Steps to study the functionality of jdbc-connector using kafka connect

1. cd /home/cloudera/training/confluent-4.1.0

```
bin/confluent start schema-registry
```

2. Connect to MySQL with user root and password cloudera

```
[cloudera@quickstart ~]$ mysql -u root -p
```



```
[cloudera@quickstart ~]$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

3. Next create a user, database, and a table:

```
mysql> create database kafkaconnect;

Query OK, 1 row affected (0.00 sec)
```

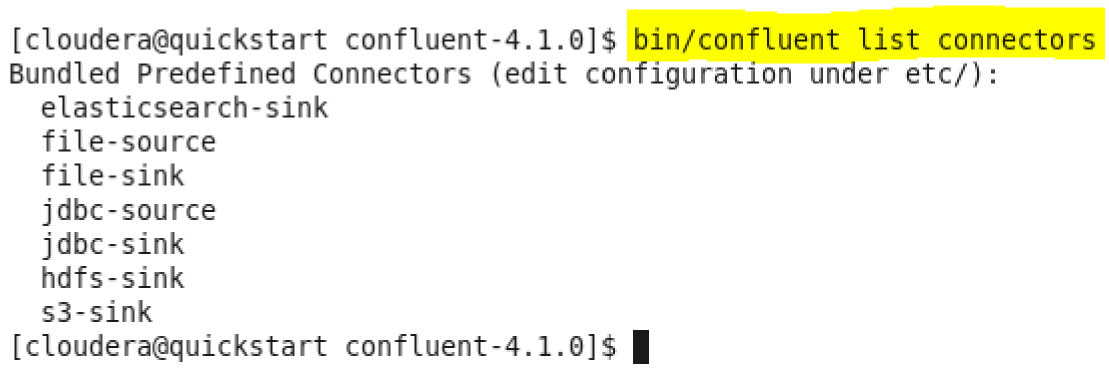
```
mysql> use kafkaconnect;

Database changed

mysql> CREATE TABLE authors(id INTEGER PRIMARY KEY

AUTO_INCREMENT NOT NULL, name VARCHAR(255));

mysql> INSERT INTO authors(name) VALUES('Manish');

mysql> INSERT INTO authors(name) VALUES('Chanchal');
```

Note: If the table exists in the database, drop the table.

Note: Please note that this will not work on MySQL versions earlier than 5.6

4. Loading the jdbc connector
   a) Checking the predefined connectors in confluent using the below command:
   b) Navigate to the location where confluent is installed.

```
bin/confluent list connectors
```



```
[cloudera@quickstart confluent-4.1.0]$ bin/confluent list connectors
Bundled Predefined Connectors (edit configuration under etc/):
  elasticsearch-sink
  file-source
  file-sink
  jdbc-source
  jdbc-sink
  hdfs-sink
  s3-sink
[cloudera@quickstart confluent-4.1.0]$ 
```

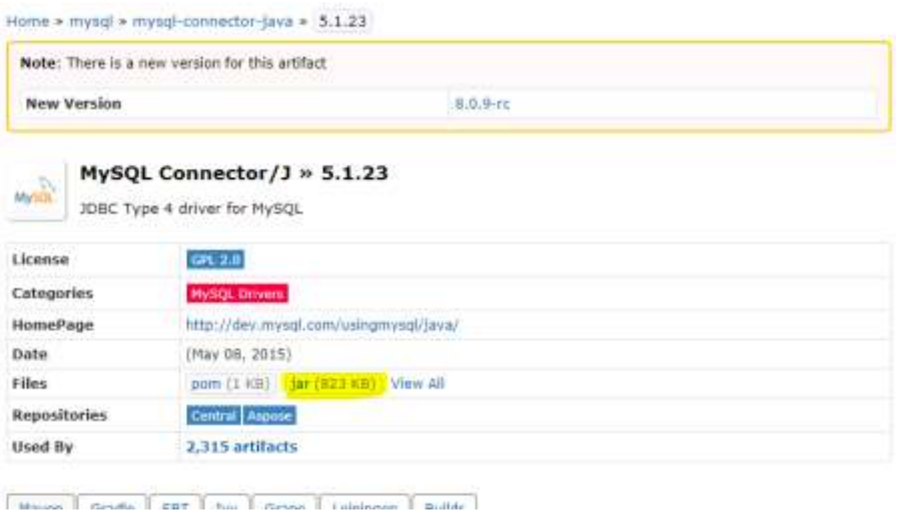
   c) Load the schema registry with the command to start schema registry on port 8081:
   from the location /home/cloudera/training/confluent-oss-4.0.0-2.11.tar/confluent-4.0.0 (navigate to the location)

```
bin/connect-standalone /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-
registry/connect-avro-standalone.properties
/home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/quickstart-
mysql.properties
```

```
(base) [cloudera@quickstart confluent-5.5.0]$ bin/connect-standalone /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/connect-avro-standalone.pro
perties /home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/quickstart-mysql.properties
```

5. Download the mysql connector java jar from below location :
   Download the jar from https://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.23

   Click on the highlighted link "jar" which enables downloading the jar(yellow)



Home » mysql » mysql-connector-java » 5.1.23

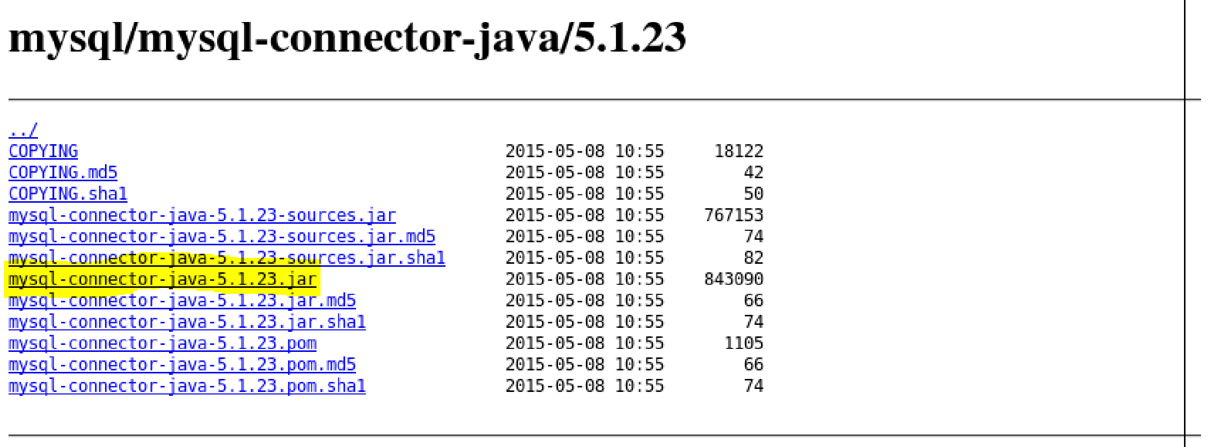
Note: There is a new version for this artifact

New Version	8.0.9-rc

MySQL Connector/J » 5.1.23

JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
HomePage	http://dev.mysql.com/usingmysql/java/
Date	(May 08, 2015)
Files	pom (1 KB) jar (823 KB) View All
Repositories	Central Aspose
Used By	2,315 artifacts



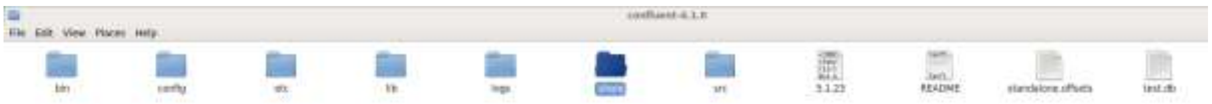
mysql/mysql-connector-java/5.1.23

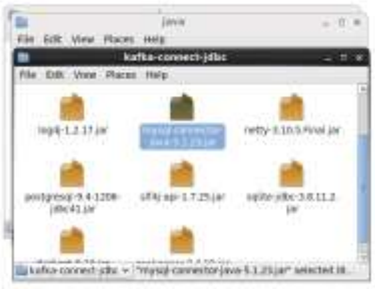
```
../
COPYING                                           2015-05-08 10:55     18122
COPYING.md5                                       2015-05-08 10:55        42
COPYING.sha1                                      2015-05-08 10:55        50
mysql-connector-java-5.1.23-sources.jar           2015-05-08 10:55    767153
mysql-connector-java-5.1.23-sources.jar.md5       2015-05-08 10:55        74
mysql-connector-java-5.1.23-sources.jar.sha1      2015-05-08 10:55        82
mysql-connector-java-5.1.23.jar                   2015-05-08 10:55    843090
mysql-connector-java-5.1.23.jar.md5               2015-05-08 10:55        66
mysql-connector-java-5.1.23.jar.sha1              2015-05-08 10:55        74
mysql-connector-java-5.1.23.pom                   2015-05-08 10:55      1105
mysql-connector-java-5.1.23.pom.md5               2015-05-08 10:55        66
mysql-connector-java-5.1.23.pom.sha1              2015-05-08 10:55        74
```

Note: As part of the lab, the jar is already downloaded and is kept in the location.

6. Copy the above MySQL Connector Jar in the following location :

```
/home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc
```





7. Create a quickstart-mysql.properties in the location </home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc> with the below content:

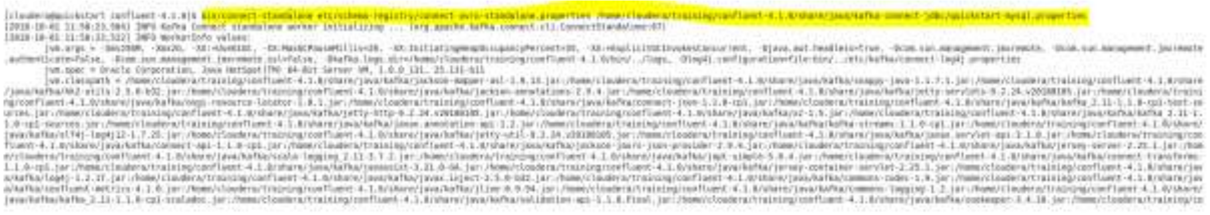
```
name=test-mysql-jdbc-autoincrement

connector.class=io.confluent.connect.jdbc.JdbcSourceConnector

tasks.max=1

connection.url=jdbc:mysql://localhost:3306/kafkaconnect?user=root&password=cl
oudera

mode=incrementing

incrementing.column.name=id

topic.prefix=test-
```

Notes:
connection.url : In the connector configuration there are no security parameters. This is because SSL is not part of the JDBC standard and will depend on the JDBC driver in use. In general, you will need to configure SSL via the connection.url parameter which this jdbc driver connection url for mysql.

table.whitelist : is the configuration for setting the table which we want to copy in kafka

if autoincrementing column is set in a table , the column has to be specified.

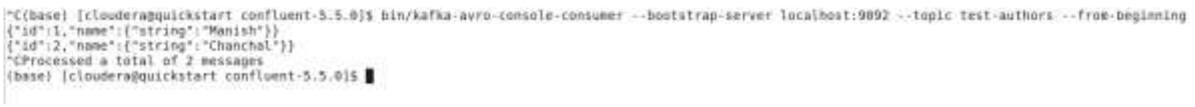
topic-prefix : Prefix to prepend to table names to generate the name of the Kafka topic to publish data to.



```
[cloudera@quickstart confluent-4.1.0]$ bin/connect-standalone etc/schema-registry/connect-avro-standalone.properties /home/cloudera/training/confluent-4.1.0/share/java/kafka-connect-jdbc/quickstart-mysql.properties
[2018-10-01 11:58:23,564] INFO Kafka Connect standalone worker initializing ... (org.apache.kafka.connect.cli.ConnectStandalone:67)
[2018-10-01 11:58:23,522] INFO WorkerInfo values:
        jvm.args = [illegible]
[illegible]
        jvm.spec = Oracle Corporation, Java HotSpot(TM) 64-Bit Server VM, 1.8.0_131, 25.131-b11
        jvm.classpath = /home/cloudera/training/confluent-4.1.0/share/java/kafka/jackson-mapper-asl-1.9.13.jar:[illegible]
[illegible]
```

8. To check the data in kafka , run the Connect Avro console consumer

```
bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic test-authors --from-beginning
```



```
^C(base) [cloudera@quickstart confluent-5.5.0]$ bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic test-authors --from-beginning
{"id":1,"name":{"string":"Manish"}}
{"id":2,"name":{"string":"Chanchal"}}
^CProcessed a total of 2 messages
(base) [cloudera@quickstart confluent-5.5.0]$ 
```

9. Add another record via the mysql command prompt:

```
mysql INSERT INTO authors(name) VALUES('Rohan');
```

You can switch back to the console consumer and see the new record is added and, importantly, the old entries are not repeated:

Note: that the default polling interval is five seconds, so it may take a few seconds to show up. Depending on your expected rate of updates or desired latency, a smaller poll interval could be used to deliver updates more quickly.

Exercise 19: Apache Kafka Connector Example – Kafka Connect JDBC Sink

Prerequisites:

- Confluent should be installed
- Java version 1.8 should be installed
- Mysql database should be installed
- Kafka and Schema Registry are running locally on the default ports.
- MySQL connector is installed.

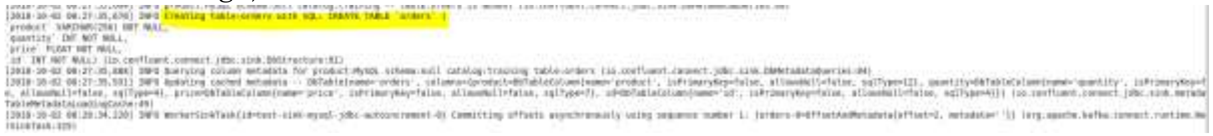
1. To Configure Data Sink Properties- Add a file as **sink-quickstart-mysql.properties** in the location etc/kafka-connect-jdbc.
   The content is

```
name=test-sink-mysql-jdbc-autoincrement

connector.class=io.confluent.connect.jdbc.JdbcSinkConnector

tasks.max=1

catalog=training

topics=orders

connection.url=jdbc:mysql://localhost:3306/training?user=root&password=clouder
a

auto.create=true
```

2. Let us Start standalone connector, with the command in terminal

```
bin/connect-standalone etc/schema-registry/connect-avro-standalone.properties
/home/cloudera/Desktop/SW/confluent-5.5.0/etc/schema-registry/sink-quickstart-
mysql.properties
```

   Observe the logs, the table is created



```
[illegible]
[2018-10-02 06:27:35,676] INFO Creating table:orders with SQL: CREATE TABLE `orders` (
`product` VARCHAR(256) NOT NULL,
`quantity` INT NOT NULL,
`price` FLOAT NOT NULL,
`id` INT NOT NULL) (io.confluent.connect.jdbc.sink.DbStructure:82)
[2018-10-02 06:27:35,[illegible]] INFO Querying column metadata for product:MySQL schema:null catalog:training table:orders (io.confluent.connect.jdbc.sink.DbMetadataQueries:[illegible])
[2018-10-02 06:27:35,[illegible]] INFO Updating cached metadata -- DbTable{name='orders', columns={product=DbTableColumn{name='product', isPrimaryKey=false, allowsNull=false, sqlType=12}, quantity=DbTableColumn{name='quantity', isPrimaryKey=f
e, allowsNull=false, sqlType=4}, price=DbTableColumn{name='price', isPrimaryKey=false, allowsNull=false, sqlType=7}, id=DbTableColumn{name='id', isPrimaryKey=false, allowsNull=false, sqlType=4}}} (io.confluent.connect.jdbc.sink.metada
TableMetadataLoadingCache:[illegible])
[2018-10-02 06:28:34,[illegible]] INFO WorkerSinkTask{id=test-sink-mysql-jdbc-autoincrement-0} Committing offsets asynchronously using sequence number 1: {orders-0=OffsetAndMetadata{offset=2, metadata=''}} (org.apache.kafka.connect.runtime.Wo
rkerSinkTask:[illegible])
```

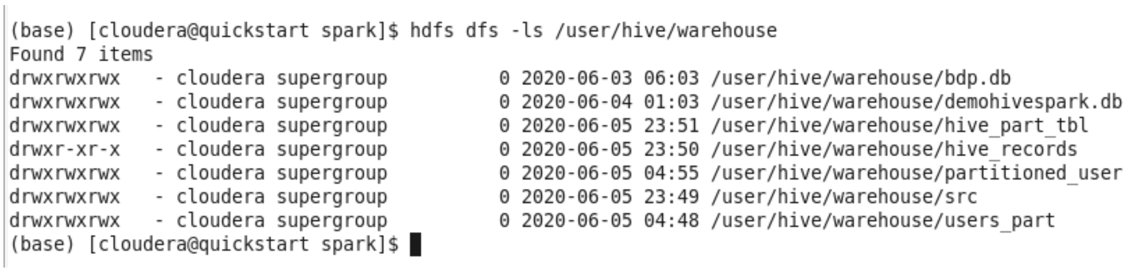
3. Let us Produce some record into the orders topic, passing the schema details of the table which will be automatically created

```
bin/kafka-avro-console-producer --broker-list localhost:9092 --topic orders --
property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"id","type":"in
t"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"},
{"name":"price","type": "float"}]}'
```

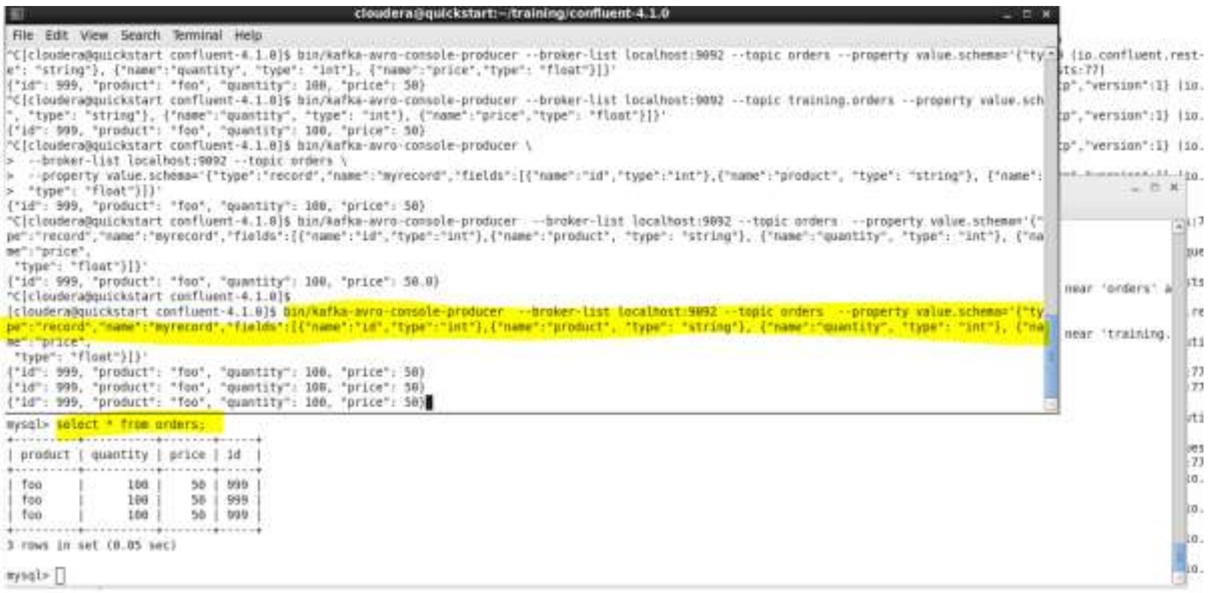
The console producer is waiting for input. Copy and paste the following record into the terminal and press **ENTER**

```
{"id": 999, "product": "foo", "quantity": 100, "price": 50}

{"id": 999, "product": "foo", "quantity": 100, "price": 50}

{"id": 999, "product": "foo", "quantity": 100, "price": 50}
```

4. Now if we query the database, we will see that the orders table was automatically created and contains the record.



```
(base) [cloudera@quickstart spark]$ hdfs dfs -ls /user/hive/warehouse
Found 7 items
drwxrwxrwx   - cloudera supergroup          0 2020-06-03 06:03 /user/hive/warehouse/bdp.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-04 01:03 /user/hive/warehouse/demohivespark.db
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:51 /user/hive/warehouse/hive_part_tbl
drwxr-xr-x   - cloudera supergroup          0 2020-06-05 23:50 /user/hive/warehouse/hive_records
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:55 /user/hive/warehouse/partitioned_user
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 23:49 /user/hive/warehouse/src
drwxrwxrwx   - cloudera supergroup          0 2020-06-05 04:48 /user/hive/warehouse/users_part
(base) [cloudera@quickstart spark]$ 
```



```
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer --broker-list localhost:9092 --topic orders --property value.schema='{"ty
e": "string"}, {"name":"quantity", "type": "int"}, {"name":"price","type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer --broker-list localhost:9092 --topic training.orders --property value.sch
", "type": "string"}, {"name":"quantity", "type": "int"}, {"name":"price","type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer \
>  --broker-list localhost:9092 --topic orders \
>  --property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":
>  "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
^C[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer  --broker-list localhost:9092 --topic orders  --property value.schema='{"
pe":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"}, {"na
me":"price",
 "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50.0}
^C[cloudera@quickstart confluent-4.1.0]$
[cloudera@quickstart confluent-4.1.0]$ bin/kafka-avro-console-producer  --broker-list localhost:9092 --topic orders  --property value.schema='{"ty
pe":"record","name":"myrecord","fields":[{"name":"id","type":"int"},{"name":"product", "type": "string"}, {"name":"quantity", "type": "int"}, {"na
me":"price",
 "type": "float"}]}'
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
{"id": 999, "product": "foo", "quantity": 100, "price": 50}
```

```
mysql> select * from orders;
+---------+----------+-------+-----+
| product | quantity | price | id  |
+---------+----------+-------+-----+
| foo     |      100 |    50 | 999 |
| foo     |      100 |    50 | 999 |
| foo     |      100 |    50 | 999 |
+---------+----------+-------+-----+
3 rows in set (0.05 sec)

mysql> 
```

Exercise 20: Add Scala Kernal in Jupyter

Open a terminal and navigate to root folder

```
cd ~
```

- Download almond and scala libs

```
$ curl -Lo coursier https://git.io/coursier-cli && chmod +x coursier
```

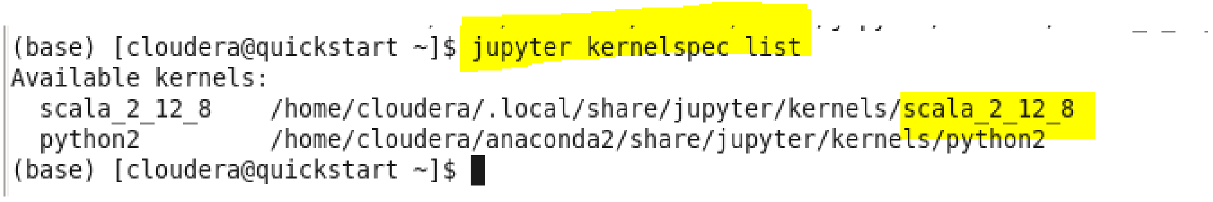
```
$ ./coursier bootstrap \
   -r jitpack \
   -i user -I user:sh.almond:scala-kernel-api_2.12.8:0.7.0 \
   sh.almond:scala-kernel_2.12.8:0.7.0 \
-o almond
```

➢ Add Scala kernel to Jupyter

```
$ ./almond --install --id scala_2_12_8  --display-name "Scala 2.12.8"
```

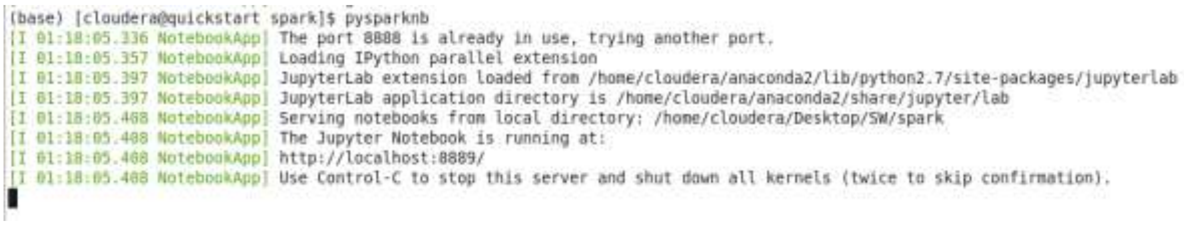
➢ List kernelsPermalink

```
jupyter kernelspec list
```



```
(base) [cloudera@quickstart ~]$ jupyter kernelspec list
Available kernels:
  scala_2_12_8    /home/cloudera/.local/share/jupyter/kernels/scala_2_12_8
  python2         /home/cloudera/anaconda2/share/jupyter/kernels/python2
(base) [cloudera@quickstart ~]$
```

Launch pysparknb to launch jupyter notebook from the terminal.

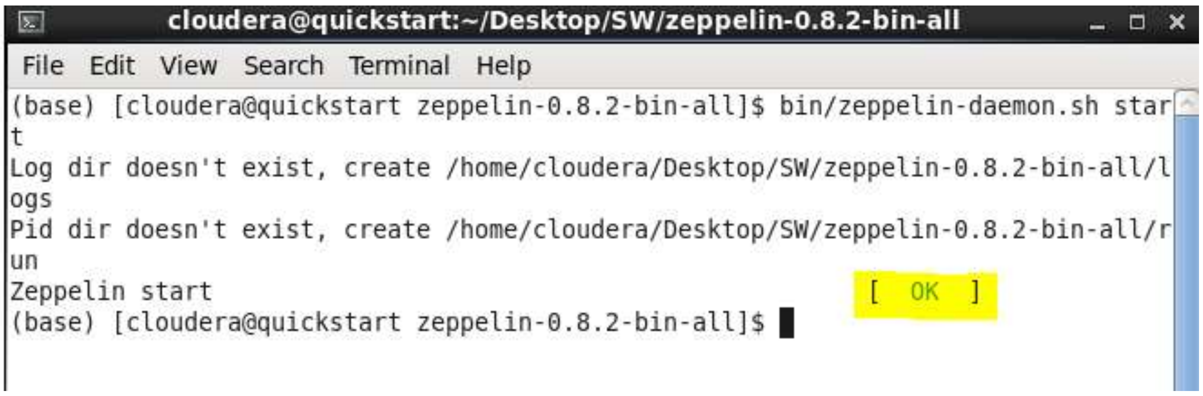


```
(base) [cloudera@quickstart spark]$ pysparknb
[I 01:18:05.336 NotebookApp] The port 8888 is already in use, trying another port.
[I 01:18:05.357 NotebookApp] Loading IPython parallel extension
[I 01:18:05.397 NotebookApp] JupyterLab extension loaded from /home/cloudera/anaconda2/lib/python2.7/site-packages/jupyterlab
[I 01:18:05.397 NotebookApp] JupyterLab application directory is /home/cloudera/anaconda2/share/jupyter/lab
[I 01:18:05.408 NotebookApp] Serving notebooks from local directory: /home/cloudera/Desktop/SW/spark
[I 01:18:05.408 NotebookApp] The Jupyter Notebook is running at:
[I 01:18:05.408 NotebookApp] http://localhost:8889/
[I 01:18:05.408 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

Its running, open mozilla with the provided url.

To run Zeppelin and create a Zeppelin notebook, run the following command :

zeppelin-daemon.sh start


cloudera@quickstart:~/Desktop/SW/zeppelin-0.8.2-bin-all
File Edit View Search Terminal Help

```
(base) [cloudera@quickstart zeppelin-0.8.2-bin-all]$ bin/zeppelin-daemon.sh star
t
Log dir doesn't exist, create /home/cloudera/Desktop/SW/zeppelin-0.8.2-bin-all/l
ogs
Pid dir doesn't exist, create /home/cloudera/Desktop/SW/zeppelin-0.8.2-bin-all/r
un
Zeppelin start                                            [  OK  ]
(base) [cloudera@quickstart zeppelin-0.8.2-bin-all]$
```

There are some useful Zeppelin commands one should know :

$ zeppelin-daemon.sh start -> To start the Daemon

$ zeppelin-daemon.sh stop -> To stop the Daemon

$ zeppelin-daemon.sh restart -> To restart the Daemon

Your Zeppelin Notebook should be accessible from the following link : http://localhost:8080/