**Kafka Setup**

1. Via WinSCP 🡪 Copy kafka\_2.11-1.1.0.gz from the windows location to the /home/notroot/downloads location in Ubuntu.
2. In Putty 🡪 extract the kafka tar file in /home/notroot/lab/software directory like this 🡺 tar -xvf ../../downloads/kafka\_2.11-1.1.0.gz
3. In WinSCP 🡪 Open the .bashrc file 🡪 Add the following 2 entries at the end of the document.

export KAFKA\_HOME=/home/notroot/lab/software/kafka\_2.11-1.1.0

export PATH=$PATH:$KAFKA\_HOME/bin

1. Then source the .bash file like this 🡪 . .bashrc [ dot and then the name of the file ]
2. Check via env command if the PATH has the kafka location like this:-

PATH= :/home/notroot/lab/software/kafka\_2.11-1.1.0/bin [ There will be other software’s also present in the path ]

Exercise 1

**Single Broker Kafka Setup and testing Kafka messaging with the default Consumer and Producer**

Note: We will be running all the commands from the Kafka Home directory in lab/software folder or give the relevant location to the config directory which will have the relevant properties files

1. create 1 kafka log directories. Do this in /home/notroot/lab/data as kafka-log1. The command is mkdir
2. configure kafka --> vi /config/server.properties
   1. Change log.dirs= /home/notroot/lab/data/kafka-log1
   2. look at the following properties to understand the **default values for each one**.
      1. broker.id 🡪 this is 0 by default
      2. listeners=PLAINTEXT://:9092 [ This would be commented ]
      3. num.partitions 🡪 1
      4. zookeeper.connect 🡪 localhost:2181
3. From the kafka installation directory :- start the zookeeper

**bin/zookeeper-server-start.sh config/zookeeper.properties**

1. start the first broker [ from a new command prompt ]

**bin/kafka-server-start.sh config/server.properties**

1. create a topic. [ from a new command prompt ]

[ write this in one line, it is shown on different lines for understanding purpose ]

**bin/kafka-topics.sh**

**--zookeeper localhost:2181**

**--create**

**--topic first**

**--partitions 1**

**--replication-factor 1**

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic first

We will get a line like this after we execute the command in the console

Created topic "first".

1. To get a list of topics

**bin/kafka-topics.sh --list --zookeeper localhost:2181**

1. To delete a topic. We will not run this now. This is only for understanding

**bin/kafka-topics.sh --delete --topic first --zookeeper localhost:2181**

We will get the following info on the prompt after executing the above command.

Topic first is marked for deletion.

Note: This will have no impact if delete.topic.enable is not set to true.

The topics will be deleted only after zookeeper and kafka server restart.

1. create producer:

**bin/kafka-console-producer.sh --broker-list localhost:9092 --topic first**

In the window console is what we will be typing the messages

By default, every new line is published as a new message and the default producer properties are specified in config/producer.properties file.

1. create consumer on a **new command** prompt

**bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic first**

We will be getting a warning, because we are currently using the old Consumer API:-

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-server] instead of [zookeeper].

Now type some lines in the producer console with new lines and you would see the same text coming in the consumer console window in separate new lines.

1. Now let us simulate another consumer on a different command prompt again.

**bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic first**

Now when we type anything in the producer console, you will see the message coming in both the consumer console windows. [But note that the **old messages** will **not be shown** in the second consumer window ]

1. On a new console window, we will start another consumer and when a new consumer starts, we can configure it to read only new messages like the earlier consumer or to get all the messages which is present in topic from beginning and do do this we will have to provide additional parameter like this

**bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic first --from-beginning**

Now you will see in the second consumer window all the old messages also getting printed.

1. Now we will start a new producer on a different console.

**bin/kafka-console-producer.sh --broker-list localhost:9092 --topic first**

Now we will try writing in this producer console and the messages will be displayed on both the consumer consoles.

End: After testing the above, we will press Ctrl+C to close the terminal windows. Then we will kill the process ids of Kafka and the Zookeeper [ This deamon is called QuorumPeerMain]

The actual messages are stored at the log location of kafka at:-

/home/notroot/lab/data/kafka-log1/first-0

In a file called 00000000000000000000.log

Exercise 2

**Multi Broker Setup**

1. To enable multi broker setup on a single system, we should create different server.properties file and change the parameters in the same.
2. Copy the server.properties in the kafka install location and rename it as server1.properties and change
   1. the broker id [i.e. 1 ]
   2. the port no [ listeners=PLAINTEXT://:9093 ]
   3. Also change the logs.dir property to /home/notroot/lab/data/kafka-log2
3. Copy the server.properties in the kafka install location and rename it as server2.properties and change
   1. the broker id [i.e. 2 ]
   2. the port no [ listeners=PLAINTEXT://:**9094** ]
   3. Also change the logs.dir property to /home/notroot/lab/data/kafka-log3 [ Note you will have to create this directory ]
4. **Re-Start the zookeeper** [ Stop and start again ]:-

**bin/zookeeper-server-start.sh config/zookeeper.properties**

[] INFO Creating new log file: log.2c1 org.apache.zookeeper.server.persistence.FileTxnLog)

A new file for zookeeper will be created at /tmp/zookeeper/versionXXX directory.

1. Start the first broker. Very Important – Note down the PID for this kafka process.

**bin/kafka-server-start.sh config/server1.properties**

1. Start the first broker. Very Important – Note down the PID for this kafka process.

**bin/kafka-server-start.sh config/server2.properties**

Now we have 2 kafka brokers running.

Check with same with the **jps** command.

1. Now let us create a new topic with replication factor as 2

**bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2**

**--partitions 1 --topic my-replicated-topic**

1. To get more details on the topic just created.

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

**my-replicated-topic**

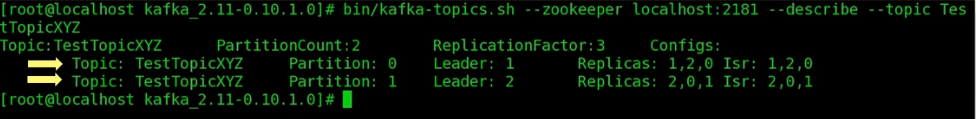
Topic:my-replicated-topic PartitionCount:1 ReplicationFactor:2 Configs:

Topic: my-replicated-topic Partition: 0 Leader: 2 Replicas: 2,1 Isr: 2,1

"leader" is the node responsible for all reads and writes for the given partition. Each node will be the leader for a randomly selected portion of the partitions.

"replicas" is the list of nodes that replicate the log for this partition regardless of whether they are the leader or even if they are currently alive.

"isr" is the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to the leader.



1. Let us publish some messages and check how it is stored.

**bin/kafka-console-producer.sh --broker-list localhost:9093 --topic**

**my-replicated-topic**

1. Also let us launch a consumer to check how the messages are been displayed.

**bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic**

**my-replicated-topic**

1. Try printing a couple of messages on the producer to see if it is getting printed on the consumer. Let us check if the messages are being stored in the log directory of the my-replicated-topic-0 folder in the kafka logs. Note: It should be present on both the kafka-log directories for the different brokers.
2. Now to test the fault tolerance. We will bring down the leader for this topic which is broker with the id 2 in my case. This would be 1 in your case also.

Kill -9 relevantkafkaPID

Then describe the topic 🡪 bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic my-replicated-topic and we will see that the other node had become the leader [ Leader 1 in my case ]

Topic:my-replicated-topic PartitionCount:1 ReplicationFactor:2 Configs:

Topic: my-replicated-topic Partition: 0 Leader: 1 Replicas: 2,1 Isr: 1

Since the producer was running, the information provided on the producer console will be as under:-

WARN [Producer clientId=console-producer] Connection to node 2 could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)

The information provided on the consumer console will be as under:-

java.nio.channels.ClosedChannelException

1. Since kafka is fault tolerant, we can continue writing on the producer and the messages will be shown on the consumer.

Exercise 3 & 4

**Writing Custom Producer and Consumers**

**Writing our custom Producer and Consumer classes**

To check if the default services are running :- Type jps and see if we see 2 Kafka daemons and QuorumPeerMain daemons are running

We will now create a **new Topic for this exercise** with the following command:

bin/kafka-topics.sh -create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic topic1

In the INFO logs you should see:- Created topic "topic1".

Pre-Requisites for writing our custom classes

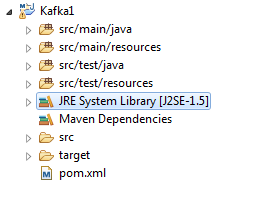
1. Install jdk-8u60-windows-x64.exe.
2. Extract eclipse-jee-mars-2-win32-x86\_64.zip
3. Open eclipse and on the workspace location, point to a folder where all the codes will be setup.
4. In the putty window 🡪 sudo apt-get update 🡪 This is to ensure that we have the latest Ubuntu repository. [ Already done in the image ]
5. In the putty window 🡪 sudo apt-get install maven 🡪 This is to ensure that we have maven installed. [ Already done in the image ]

Now let us create a new Maven Project. Start eclipse

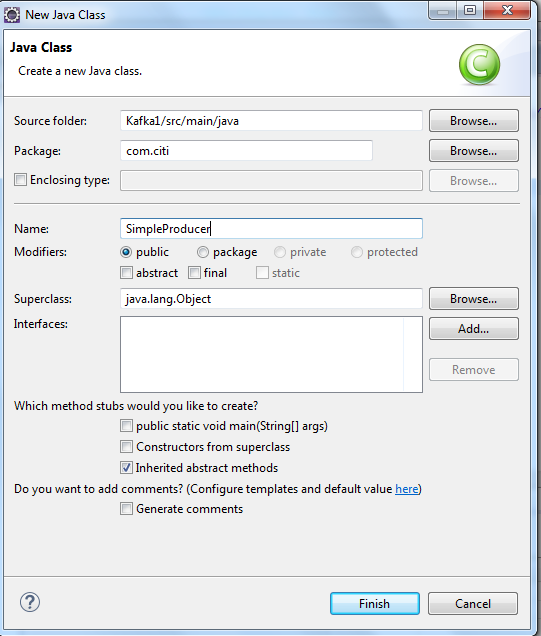
1. File – New – Maven Project
2. Click on the checkbox called “Create Simple Project” 🡪 next

In the Group ID – CitiKafka

In the Artifact ID – Kafka1 🡪 Finish



Right click on src/main/java 🡪 new class and give the package name as com.citi. and the class as SimpleProducer



**Producer Code**

package com.jpmc;

import java.util.Properties;

import java.util.Scanner;

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

import org.apache.kafka.clients.producer.Producer;

import org.apache.kafka.clients.producer.ProducerRecord;

public class SimpleProducer {

private static Scanner in;

public static void main(String[] argv)throws Exception {

if (argv.length != 1) {

System.err.println("Please specify 1 parameters ");

System.exit(-1);

}

String topicName = argv[0];

in = new Scanner(System.in);

System.out.println("Enter message(type exit to quit)");

//Configure the Producer

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9093");

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

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

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

String line = in.nextLine();

while(!line.equals("exit")) {

ProducerRecord<String, String> rec = new ProducerRecord<String, String>(topicName, line);

producer.send(rec);

line = in.nextLine();

}

in.close();

producer.close();

}

}

**Consumer Code**

package com.jpmc;

import java.util.Arrays;

import java.util.Properties;

import java.util.Scanner;

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

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

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

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

public class SimpleConsumer {

private static Scanner in;

public static void main(String[] argv)throws Exception{

if (argv.length != 2) {

System.err.printf("Usage: %s <topicName> <groupId>\n",

SimpleConsumer.class.getSimpleName());

System.exit(-1);

}

in = new Scanner(System.in);

String topicName = argv[0];

String groupId = argv[1];

ConsumerThread consumerRunnable = new ConsumerThread(topicName,groupId);

consumerRunnable.start();

String line = "";

while (!line.equals("exit")) {

line = in.next();

}

consumerRunnable.getKafkaConsumer().wakeup();

System.out.println("Stopping consumer .....");

consumerRunnable.join();

}

private static class ConsumerThread extends Thread{

private String topicName;

private String groupId;

private KafkaConsumer<String,String> kafkaConsumer;

public ConsumerThread(String topicName, String groupId){

this.topicName = topicName;

this.groupId = groupId;

}

public void run() {

Properties configProperties = new Properties();

configProperties.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9093");

configProperties.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

configProperties.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

configProperties.put(ConsumerConfig.GROUP\_ID\_CONFIG, groupId);

configProperties.put(ConsumerConfig.CLIENT\_ID\_CONFIG, "simple");

//Figure out where to start processing messages from

kafkaConsumer = new KafkaConsumer<String, String>(configProperties);

kafkaConsumer.subscribe(Arrays.asList(topicName));

//Start processing messages

try {

while (true) {

ConsumerRecords<String, String> records = kafkaConsumer.poll(100);

for (ConsumerRecord<String, String> record : records)

System.out.println(record.value());

}

}catch(Exception ex){

System.out.println("Exception caught " + ex.getMessage());

}finally{

kafkaConsumer.close();

System.out.println("After closing KafkaConsumer");

}

}

public KafkaConsumer<String,String> getKafkaConsumer(){

return this.kafkaConsumer;

}

}

}

**Pom.xml**

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>JPMCKafka</groupId>

<artifactId>Kafka3</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka-clients</artifactId>

<version>1.0.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.12</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.12</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

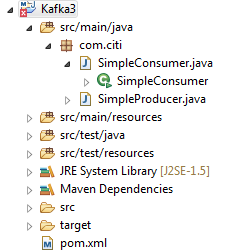
</configuration>

</plugin>

</plugins>

</build>

</project>



If the project shows error like this :-

Project configuration is not up-to-date with pom.xml. Select: Maven->Update Project... from the project context menu or use Quick Fix.

Right Click on the project – Maven – Update Project - OK

**Deployment Steps**

1. Copy the Kafka1 complete Folder to lab/programs directory
2. From within the directory, we will clean, built and create a package with the following commands :-

mvn clean  
mvn package -DskipTests

Note: Ensure that we have Java as JDK and not as JRE. [ Change the Installed JREs ]

We can also do this in windows itself 🡪

* Right Click on the project – Run As – Maven – Maven Clean 🡺 After that.
* Right Click on the project – Run As – Maven – Maven Install

The final jar will be created inside the target directory.

1. On a new terminal window, we will start the Producer with the following command.

java -cp target/Kafka1-0.0.1-SNAPSHOT-jar-with-dependencies.jar com.jpmc.SimpleProducer topic1

[Note the name of the jar should be based on what is your project name ]

1. On a separate terminal window, we will start with the Consumer with the following command.

java -cp target/Kafka1-0.0.1-SNAPSHOT-jar-with-dependencies.jar com.jpmc.SimpleConsumer topic1 group1

[Note the name of the jar should be based on what is your project name ]

Explanation

Properties that we added in the Producer Class.

* BOOTSTRAP\_SERVERS\_CONFIG
* KEY\_SERIALIZER\_CLASS\_CONFIG
* VALUE\_SERIALIZER\_CLASS\_CONFIG

BOOTSTRAP\_SERVERS\_CONFIG (bootstrap.servers) sets a list of host:port pairs used for establishing the initial connections to the Kakfa cluster in the host1:port1,host2:port2,... format. Even if we have more than one broker in our Kafka cluster, we only need to specify the value of the first broker's host:port.

The Kafka client will use this value to make a discover call on the broker, which will return a list of all the brokers in the cluster. It's a good idea to specify more than one broker in the BOOTSTRAP\_SERVERS\_CONFIG, so that if that first broker is down the client will be able to try other brokers.

The Kafka server expects messages in byte[] key, byte[] value format. Rather than converting every key and value, Kafka's client-side library permits us to use friendlier types like String and int for sending messages. The library will convert these to the appropriate type. For example, the sample app doesn't have a message-specific key, so we'll use *null* for the key. For the value we'll use a String, which is the data entered by the user on the console.

To configure the *message key*, we set a value of KEY\_SERIALIZER\_CLASS\_CONFIG on the org.apache.kafka.common.serialization.ByteArraySerializer. This works because *null* doesn't need to be converted into byte[]. For the *message value*, we set VALUE\_SERIALIZER\_CLASS\_CONFIG on the org.apache.kafka.common.serialization.StringSerializer, because that class knows how to convert a String into a byte[].

Four mandatory properties on the Consumer side.

* BOOTSTRAP\_SERVERS\_CONFIG (bootstrap.servers)
* KEY\_DESERIALIZER\_CLASS\_CONFIG (key.deserializer)
* VALUE\_DESERIALIZER\_CLASS\_CONFIG (value.deserializer)
* GROUP\_ID\_CONFIG (bootstrap.servers)

The GROUP\_ID\_CONFIG is a group name in string format.

First, ConsumerThread is an inner class that takes a topic name and group name as its arguments. In the run() method it creates a KafkaConsumer object, with appropriate properties. It subscribes to the topic that was passed as an argument in the constructor, by calling the kafkaConsumer.subscribe() method, then polls the Kafka server every 100 milliseconds to check if there are any new messages in the topic. It will iterate through the list of any new messages and print them to the console.

In the Consumer class we create a new object of ConsumerThread and start it in a different thread. The ConsumerThead starts an infinite loop and keeps polling the topic for new messages. Meanwhile in the Consumer class, the main thread waits for a user to enter exit on the console. Once a user enters exit, it calls the KafkaConsumer.wakeup() method, causing the KafkaConsumer to stop polling for new messages and throw a WakeupException. We can then close the KafkaConsumer gracefully, by calling kafkaConsumer's close() method.

**Important Broker Configuration Parameters**

**The complete list is available at:** [**https://kafka.apache.org/documentation/#brokerconfigs**](https://kafka.apache.org/documentation/#brokerconfigs)

Zookeeper.connect

This parameter takes zookeeper connection string. The connection string is simply a hostname with a port number. We already know that Kafka uses zookeeper for various coordination purposes, so it is critical that every broker knows the zookeeper address. This parameter is also necessary to form a cluster.

Well, all brokers are running on different systems, how do they know about each other. If they don't know each other, they are not part of the cluster. So, the zookeeper is the connecting link among all brokers to form a cluster.

delete.topic.enable

If you want to delete a topic, you can use topic management tool. But by default, deleting a topic is not allowed. You can't remove a topic because the default value for this parameter is false. That is reasonable protection for production environments. But in development or testing environment, you may want to delete topics. So, if you want Kafka to allow deleting a topic, you need to set this parameter to true.

auto.create.topics.enable

We have already discussed auto-create topic feature. If a producer starts sending messages to a non-existent topic, Kafka will create the topic automatically and accept the data. This behaviour is suitable for dev environments.

default.replication.factor and num.partitions

The default values for both of them is one, and they are effective when you have auto create topics enabled. So, if Kafka is creating your topic automatically, the new topic will have only one partition and a single copy.

log.retention.ms and log.retention.bytes

So, whatever data you send to Kafka, it is not retained by Kafka forever. Kafka is not a database. You don't send data to Kafka for storage so that you can query it later. It is a message broker. It should deliver the data to the consumer and then clean it up. There is no reason to retain messages for longer than needed.   
Kafka gives you two options to configure the retention period. The default option is retention by time, and the default retention period is seven days.   
Kafka gives you another option to define this retention period. You can specify it by size. That's where the second parameter log retention bytes is applicable. But this size applies to partition. So, if you set log.retention.bytes = 1 GB, Kafka will trigger a clean-up activity when the partition size reaches to 1 GB. Remember that it is not a topic size. It is partition size.   
If you specify both configurations, the clean-up will start on meeting either of the criteria.