**What is Kafka?**

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the Zookeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

**Apache Kafka Concepts**

This chapter describes several basic concepts that support fault-tolerant, scalable messaging provided by Apache Kafka:

• Topics

• Producers

• Consumers

• Brokers

For additional introductory information about Kafka, see the Apache introduction to Kafka. For an example that simulates the use of streaming geo-location information (based on a previous version of Kafka), see Simulating and Transporting the Real-Time Event Stream with Apache Kafka.

**Topics/Partitions**

Kafka maintains feeds of messages in categories called topics. Each topic has a user-defined category (or feed name), to which messages are published.

**Benefits**

Following are a few benefits of Kafka:

 **Reliability** - Kafka is distributed, partitioned, replicated and fault tolerance.

 **Scalability** - Kafka messaging system scales easily without down time.

 **Durability** - Kafka uses “Distributed commit log” which means messages persists on disk as fast as possible, hence it is durable.

 **Performance** - Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

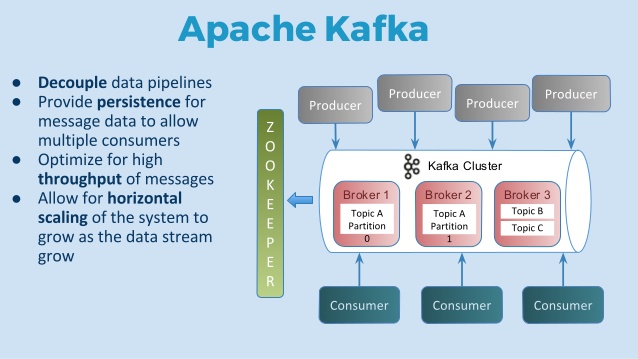
Kafka is very fast and guarantees zero downtime and zero data loss.

****

|  |  |
| --- | --- |
| **Components** | **Description** |
|  |  |
| Topics | A stream of messages belonging to a particular category is called a topic. Data is stored in topics. |
| Partition | Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.  Topics may have many partitions, so it can handle an arbitrary amount of data. |
| Partition offset | Each partitioned message has a unique sequence id called as “offset”. |
| Replicas of partition | Replicas are nothing but “backups” of a partition. Replicas are never read or write data. They are used to prevent data loss. |
| Brokers | i) Brokers are simple system responsible for maintaining the published data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.  ii) Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.  iii) Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load  Distribution among the broker. |
| Kafka Cluster | Kafka’s having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the Persistence and replication of message data. |
| Producers | Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice. |

|  |  |
| --- | --- |
| Consumers | Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers. |
| Leader | "Leader" is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader. |
| Follower | Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store. |

**Kafka – Cluster Architecture**

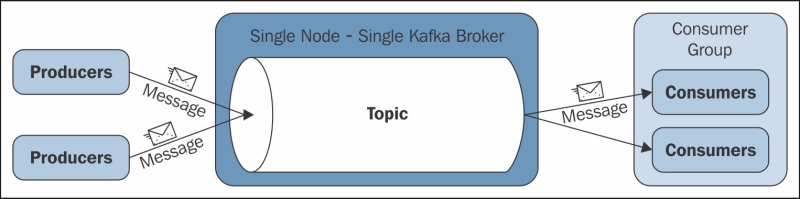
****

The following table describes each of the components shown in the above diagram.

|  |  |
| --- | --- |
| **Components** | **Description** |
| Broker | Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use Zookeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each broker can handle TB of messages without performance impact. Kafka broker leader election can be done by Zookeeper. |
| Zookeeper | Zookeeper is used for managing and coordinating Kafka broker. Zookeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker. |
| Producers | Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn’t wait for acknowledgements from the broker and sends messages as fast as the broker can handle. |
| Consumers | Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by Zookeeper.. |

**Design**

In a very basic structure, a producer publishes messages to a Kafka topic, which is created on a Kafka broker acting as a Kafka server. Consumers then subscribe to the Kafka topic to get the messages. This is described in the following diagram:

****

In the preceding diagram a single node – single broker architecture is shown. This architecture considers that all three parties—producers, Kafka broker, and consumers—are running on different machines. Here, each consumer is represented as a process and these processes are organized within groups called **consumer groups**.

A message is consumed by a single process (consumer) within the consumer group, and if the requirement is such that a single message is to be consumed by multiple consumers, all these consumers need to be kept in different consumer groups.

By Kafka design, the message state of any consumed message is maintained within the message consumer, and the Kafka broker does not maintain a record of what is consumed by whom, which also means that poor designing of a custom consumer ends up in reading the same message multiple times.

Important Kafka design facts are as follows:

• The fundamental backbone of Kafka is message caching and storing it on the filesystem. In Kafka, data is immediately written to the OS kernel page. Caching and flushing of data to the disk is configurable.

• Kafka provides longer retention of messages ever after consumption, allowing consumers to re-consume, if required.

• Kafka uses a message set to group messages to allow lesser network overhead.

• Unlike most of the messaging systems, where metadata of the consumed messages are kept at server level, in Kafka, the state of the consumed messages is maintained at consumer level. This also addresses issues such as:

°° loosing messages due to failure

°° Multiple deliveries of the same message

By default, consumers store the state in Zookeeper, but Kafka also allows storing it within other storage systems used for **Online Transaction Processing** (**OLTP**) applications as well.

• In Kafka, producers and consumers work on the traditional push-and-pull model, where producers push the message to a Kafka broker and consumers pull the message from the broker.

• Kafka does not have any concept of a master and treats all the brokers as peers. This approach facilitates addition and removal of a Kafka broker at any point, as the metadata of brokers are maintained in Zookeeper and shared with producers and consumers.

• In Kafka 0.7.x, Zookeeper-based load balancing allows producers to discover the broker dynamically. A producer maintains a pool of broker connections, and constantly updates it using Zookeeper watcher callbacks. But in Kafka 0.8.x, load balancing is achieved through Kafka metadata API and Zookeeper can only be used to identify the list of available brokers.

• Producers also have an option to choose between asynchronous or synchronous mode for sending messages to a broker.

**Configuration**

**Downloading the Required Files**

Download Server JRE according to your OS & CPU architecture from <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html> .

Download Zookeeper from <http://zookeeper.apache.org/releases.html>

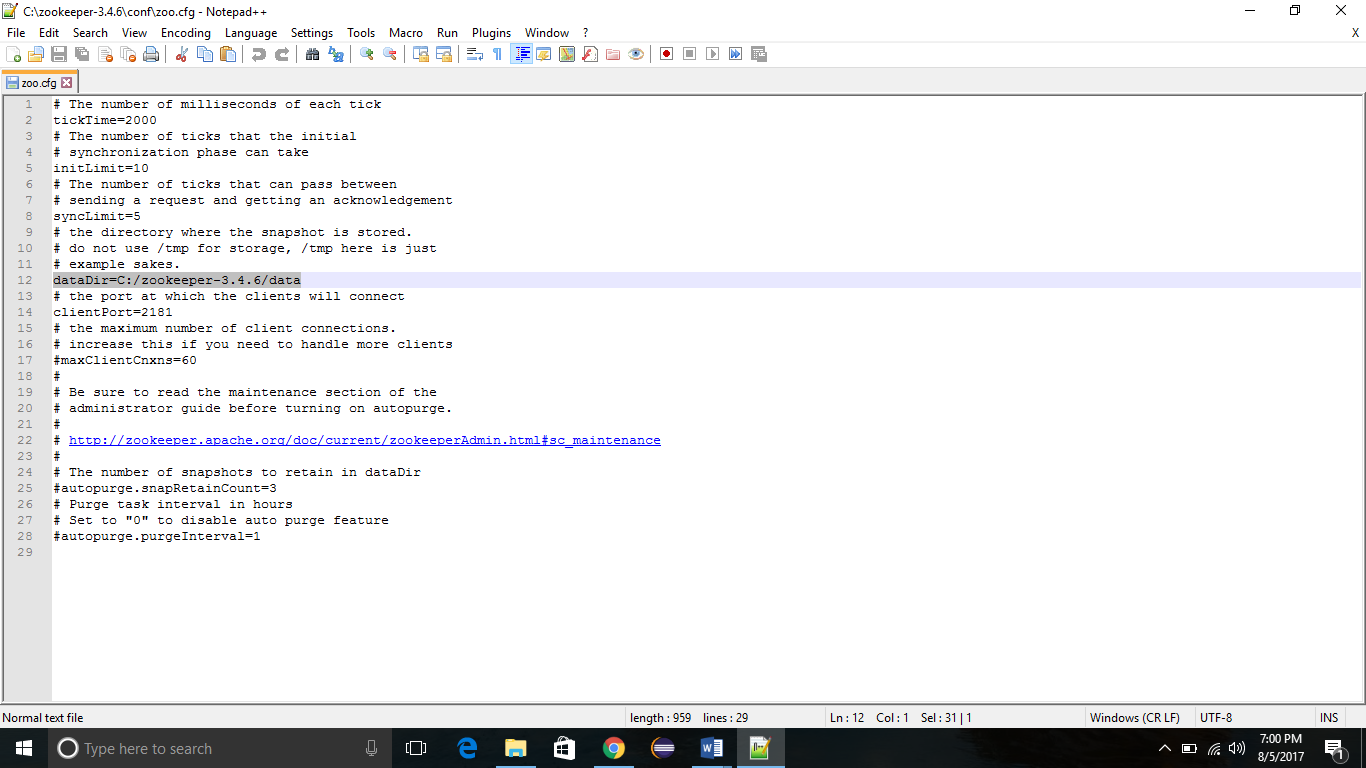
Download Kafka from <http://kafka.apache.org/downloads.html>

**Start JRE installation**

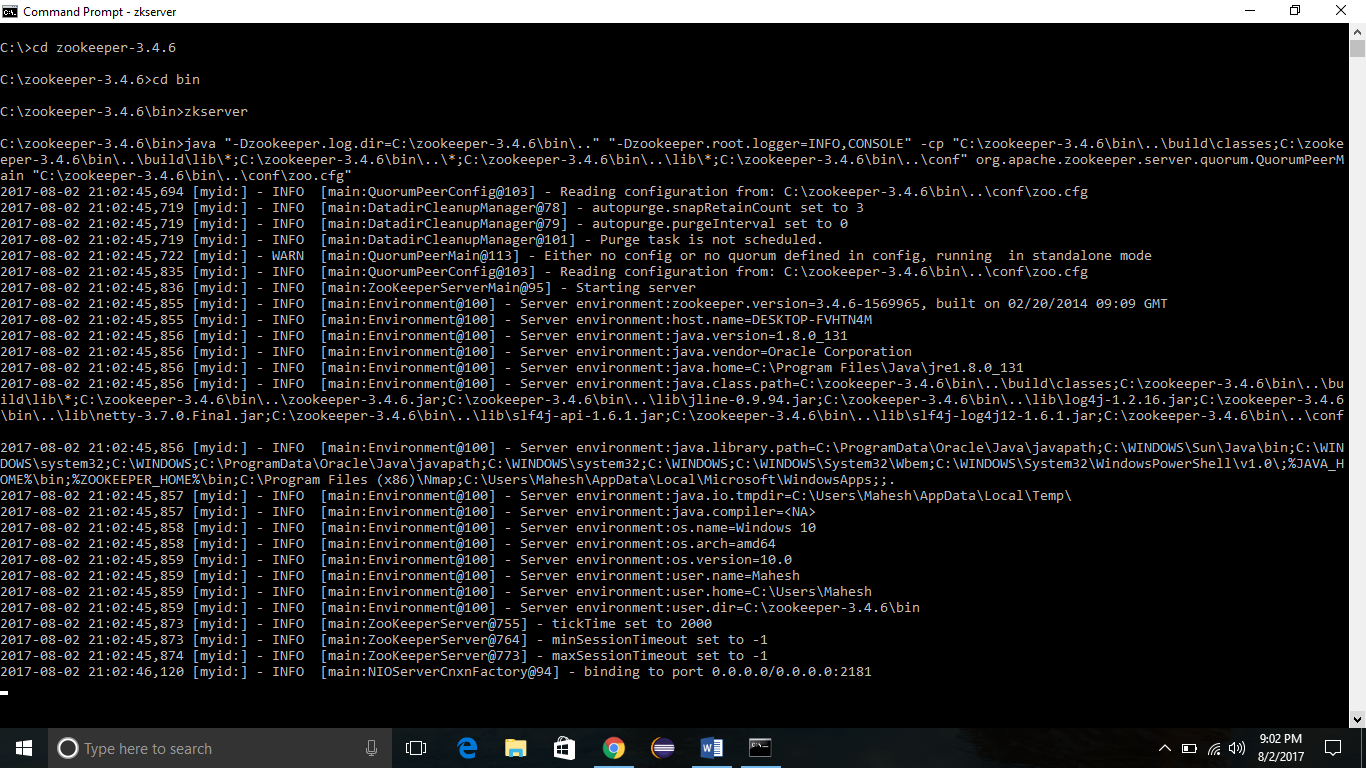
1. Now open system environment variables dialogue by opening Control Panel -> System -> Advanced system settings -> Environment Variables…
2. Hit New… button in User variables section then type JAVA\_HOME in Variable name & give your jre path in Variable value. JAVA\_HOME = C:\Program Files\Java
3. Search for Path variable in the “System Variable” section in “Environment Variables” dialogue box you just opened.
4. Edit the path and type “;%JAVA\_HOME%\bin” at the end of the text already written in the path.
5. To confirm java installation just open cmd and type “java –version”, you should be able to see version of the java you just installed.

**Installing & Running Zookeeper**

1. Goto your zookeeper config directory. For me its C:\zookeeper-3.4.7\conf
2. Rename file “zoo\_sample.cfg” to “zoo.cfg”
3. Open zoo.cfg in any text editor like notepad but I’ll prefer notepad++.
4. Find & edit dataDir=/tmp/zookeeper to dataDir=C:\zookeeper-3.4.7\data a. Add entry in System Environment Variables as we did for java b. Add in System Variables ZOOKEEPER\_HOME = C:\zookeeper-3.4.7

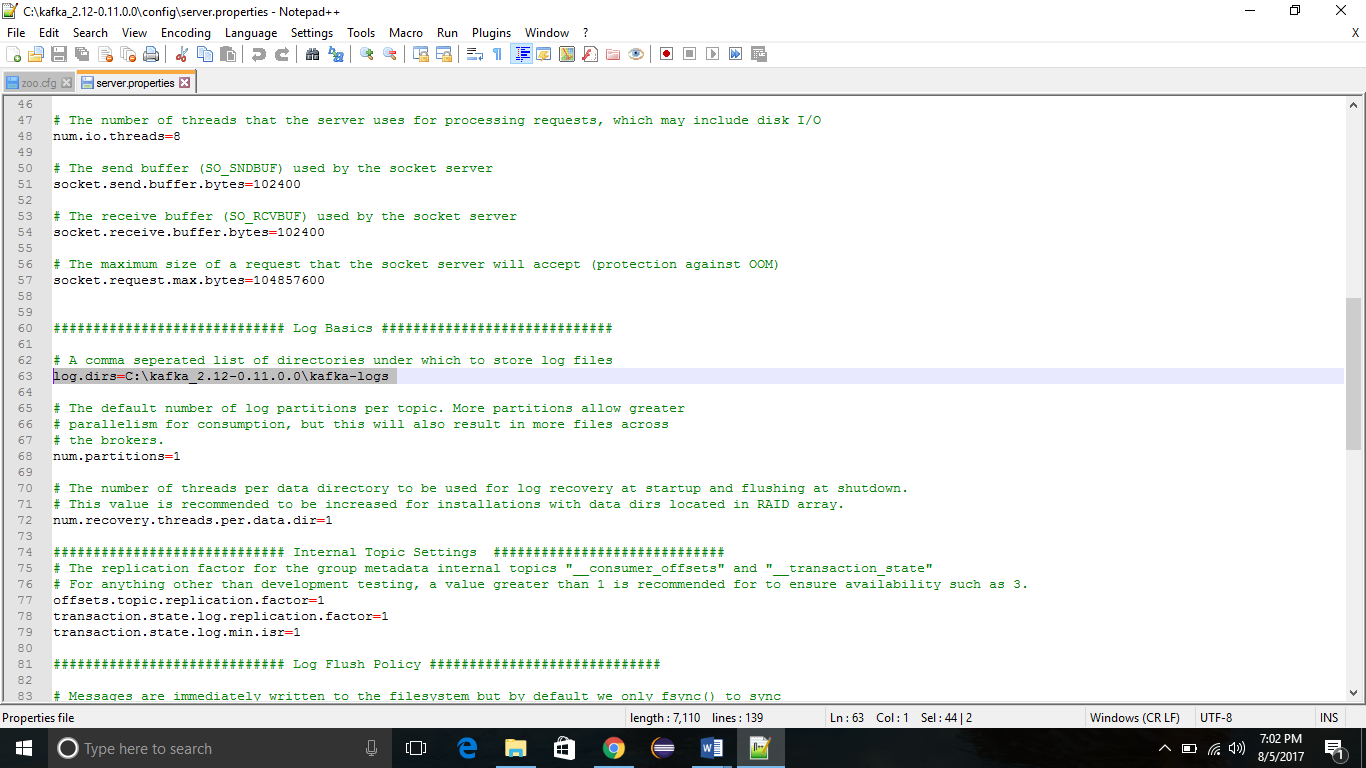


1. Edit System Variable named “Path” add ;%ZOOKEEPER\_HOME%\bin;
2. You can change the default zookeeper port in zoo.cfg file (Default port 2181).
3. Run zookeeper by opening a new cmd & Type C:\zookeeper-3.4.6\bin>zkserver.



**Setting Up Kafka**

1. Go to your Kafka config directory. For me its C:\kafka\_2.11-0.9.0.0\config
2. Edit file “server.properties”
3. Find & edit line “log.dirs=/tmp/kafka-logs” to “log.dir= C:\kafka\_2.11-0.9.0.0\kafka-logs”.

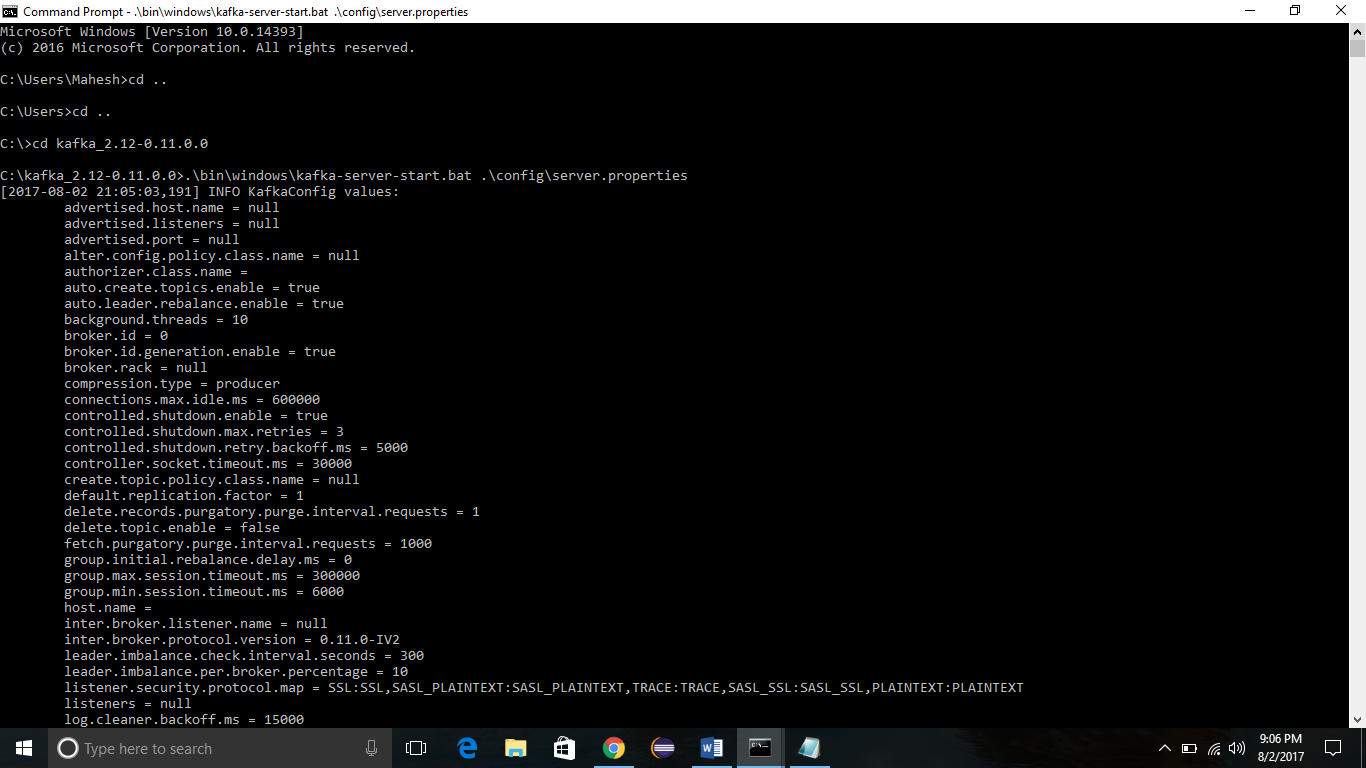


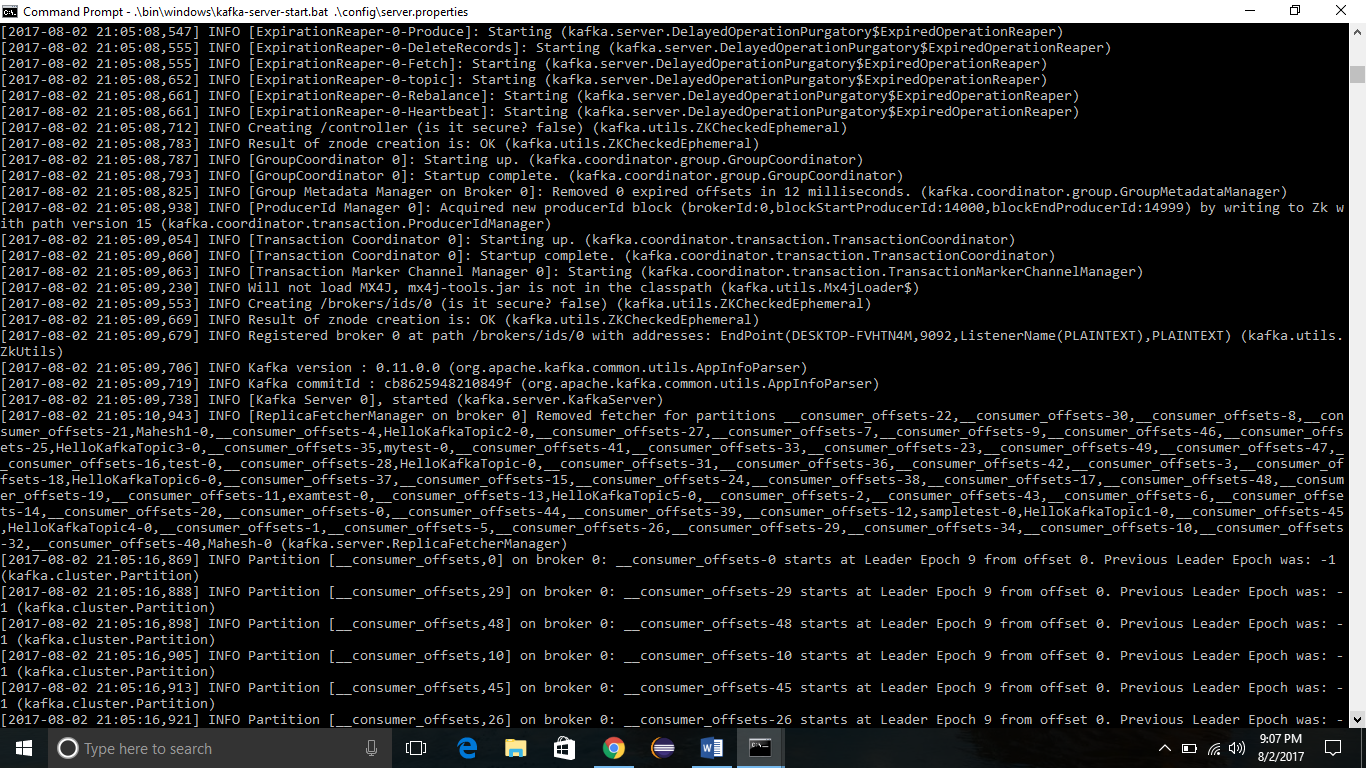
1. If your zookeeper is running on some other machine or cluster you can edit “zookeeper.connect=localhost:2181” to your custom IP & port. For this demo we are using same machine so no need to change. Also Kafka port & broker.id are configurable in this file. Leave other settings as it is.
2. Your Kafka will run on default port 9092 & connect to zookeeper’s default port which is 2181.

**Running Kafka Server**

Note: Please ensure that your zookeeper is up & running before starting Kafka server.

1. Go to your kafka installation directory C:\kafka\_2.11-0.9.0.0\
2. Open a command prompt here by pressing Shift + right click and choose “Open command window here” option)
3. Now type .\bin\windows\kafka-server-start.bat .\config\server.properties & press enter.





**Creating Topic**

1. Open a new command prompt in the location C:\kafka\_2.11-0.9.0.0\bin\windows
2. Type following command and hit enter kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic kafkatest
3. The last word is your Topic Name........

**Creating a producer & consumer to test server...**

1. Open a new command prompt in the location C:\kafka\_2.11-0.9.0.0\bin\windows
2. To start a producer type command “kafka-console-producer.bat --broker-list localhost:9092 --topic kafkatest”.
3. Again open a new command prompt in the same location as C:\kafka\_2.11-0.9.0.0\bin\windows
4. Now start a consumer by typing command “kafka-console-consumer.bat --zookeeper localhost:2181 --topic kafkatest”.
5. Now type anything in the producer command prompt & press enter and you should be able to see the message in the other consumer command prompt.

**Some Other Useful Commands**

1. List Topics: kafka-topics.bat --list --zookeeper localhost:2181
2. Describe Topic: kafka-topics.bat --describe --zookeeper localhost:2181 --topic [Topic Name]
3. Read messages from beginning: kafka-console-consumer.bat --zookeeper localhost:2181 --topic [Topic Name] --from- beginning
4. Delete Topic: kafka-run-class.bat kafka.admin.TopicCommand --delete --topic [topic\_to\_delete] --zookeeper localhost:2181