# Kafka

Apache Kafka is a distributed software system that allows developers to process and store large amounts of real-time data

For this tutorial, Landoop kafka distribution and Windows version of the Kafka is used.

Landoop docker container installation steps exist in install\_and\_config folder.

Open landoop docker bash in CMD and mount local dev folder to docker -

*Directories are generally mounted for consuming the files present in host system by docker.*

*If the below command does not work then, docker copy command can be executed to move files from host system to docker*

|  |
| --- |
| *docker run --rm -it -v %cd%:/main --net=host landoop/fast-data-dev bash* |

Open docker bash in CMD without mounting any directory - use below command

|  |
| --- |
| *docker run --rm -it --net=host landoop/fast-data-dev bash* |

## Test Kafka executing below commands

### Create new topic

|  |
| --- |
| *kafka-topics --create --topic test-topic --bootstrap-server localhost:9092 --replication-factor 1 --partitions 2* |

### List kafka topic

|  |
| --- |
| *kafka-topics --bootstrap-server localhost:9092 --list* |

### Console kafka producer - Starts console producer

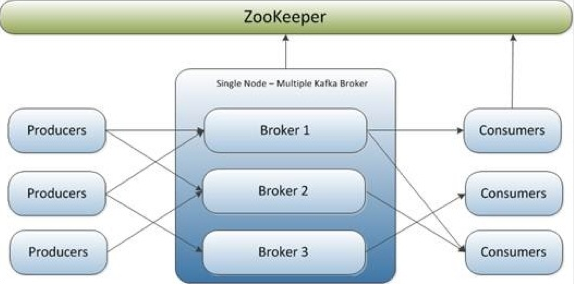
|  |
| --- |
| *kafka-console-producer --bootstrap-server localhost:9092 --topic test-topic* |

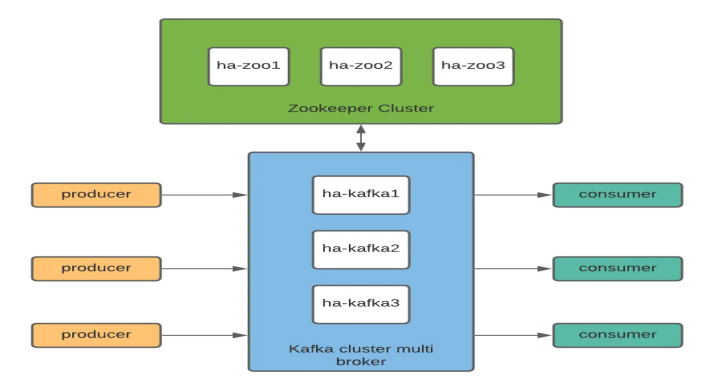
### Console kafka Consumer - Starts console consumer

|  |
| --- |
| *kafka-console-producer --bootstrap-server localhost:9092 --topic test-topic* *kafka-console-consumer --bootstrap-server localhost:9092 --topic test-topic --from-beginning* |

# Kafka Architecture

* In production systems, there will be multiple brokers and zookeepers.
* While configuring the topics we can specify only 1 broker server and configuration property ***replication-factor*** integer value will decide that how many times the topics will be replicated across multiple other brokers.
* If any Broker fails then other brokers can manage the messages to and from the topics
* Zookeeper will manage the health of all the brokers and manage the communication across them
* Multiple Zookeeper servers will be there in case primary zookeeper server fails then other redundant server will take its place





# Topics, partitions, and offsets in Kafka

* Kafka **Topic** is the queue where Producer pushes the messages and Consumer consumes it.
* Each Topic can be divided into multiple **partitions**. Partition Id for a kafka topic start from 0.
* Each partition can have an incremental message counter called **Offset**. Each partition Offset starts from 0 and it will incrementally increase. Offset denotes the unique position of a message within a partition.
* Offset guarantees the chronological order of the messages in a partition, but it does not guarantee the order across different partitions i.e. In a same partition, Message with Offset 0 came before the Message with Offset 1 however, Message with Offset 1 in Partition 1 may occur before Offset 0 of Partition 2.

A screen shot of a screen

Description automatically generated

* Messages stored in the topics as immutable objects.
* Messages in a topic are stored in partitions. Each message is composed of a key, a value, and additional metadata (such as headers and timestamps).
* The key-value pair is the main component of each message, where the key is optional and is often used to determine the partition to which the message is assigned.
* Based on the key, Kafka stores messages in specific partitions, ensuring that messages with the same key are consistently placed in the same partition. If the key is absent, Kafka distributes the messages across partitions using a round-robin or another default strategy.
* Consumers are not required to consume messages in chronological order, especially when consuming from multiple partitions, as each partition can be processed independently and concurrently.
* Consumers can consume the messages from a specific Partition and can also be configured with the Offset (offset indicates the specific position from which a consumer starts or continues consuming messages in a partition)
* While configuring Offset in the consumers, Partition needs to be configured else error will occur.
* While creating Topic, partitions is the property used to specify the number of partitions.
* While consuming data from a Topic by a Consumer, ***partition*** and ***offset*** is the property used to specify the partition number from where message needs to be read and offset number for the position

(from that position, messages will be consumed by the consumer) .

***Bash Scripts:***

|  |
| --- |
| * **Create new topic with 3 partitions**   *kafka-topics --create --topic test-topic --bootstrap-server localhost:9092 --replication-factor 1 --partitions 3*   * **Console kafka Consumer - Configure consumer to get the messages from a specific partition**   *kafka-console-consumer --bootstrap-server localhost:9092 --topic test-topic --partition 1*   * **Console kafka Consumer - Configure consumer to get the messages from a specific Offset**   *kafka-console-consumer --bootstrap-server localhost:9092 --topic test-topic --partition 1 --offset 1*   * **Console kafka Consumer - configure consumer to get the messages from a specific Offset without configuring partition. Below code will throw error, while configuring offset, partition needs to be configured**   *kafka-console-consumer --bootstrap-server localhost:9092 --topic test-topic --offset 1* |

# Kafka Cluster

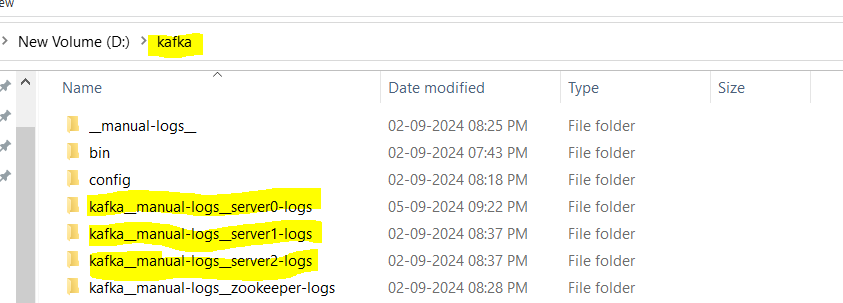
* In production environment multiple brokers are used for fault tolerance.
* For this exercise
  + Kafka is downloaded on Windows system and multiple copies of server.properties files will be used to spin up (mimic) cluster with multiple brokers.
  + Each copy of server.properties file will have different listener, different broker Id, different server log folder path and Zookeeper host will remain same in all the files, so that Zookeeper can track all the brokers.
  + In consumer.properties and producer.properties file, update bootstrap-server property to include all the broker server.
* Installation steps exist in install\_and\_config folder.

***Bash Scripts:***

|  |
| --- |
| * **Start Zookeeper using below code - change path of zookeeper.properties**   *zookeeper-server-start D:\kafka\config\zookeeper.properties*   * **Start kafka clsuter - multiple brokers**   *kafka-server-start D:\kafka\config\server0.properties*  *kafka-server-start D:\kafka\config\server1.properties*  *kafka-server-start D:\kafka\config\server2.properties*   * **Create a topic with 5 partitions**   *kafka-topics --create --topic test-cluster-topic --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --replication-factor 1 --partitions 5*   * **Create a producer with key separator. Test Message - 1: Jerin, 2:ABCD etc...**   *kafka-console-producer --topic multi-broker-topic --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --property key.parse=true --property key.seperator=: --property "parse.key=true" --property "key.separator=:"*   * **Create kafka console consumer to populate key and its values**   *kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --topic multi-broker-topic --from-beginning --property print.key=true --property "key.separator=:"* |

* Whenever a topic is created with multiple partitions then we can see folders for each partition in log folder.

***Below Screenshot shows 3 log folders for 3 Kafka brokers***

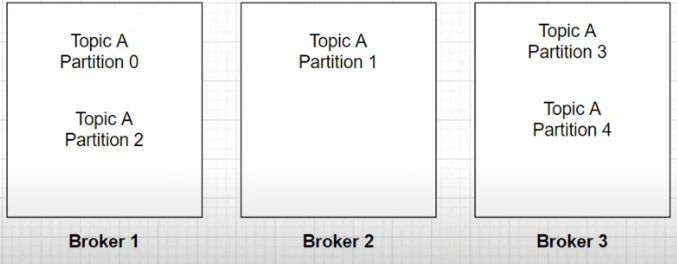


***After getting into 1 of the log folders, we can see multiple sub-folders for each partition (applicable in standalone as well as multi broker Kafka cluster).***

A screenshot of a computer

Description automatically generated

* In Kafka Cluster with multiple brokers (replication-factor = 1), whenever a topic is created with multiple partitions, those partitions will reside in different brokers so that if any broker crashes, then partitions in other brokers should still be receiving and sending messages.



Refer above screenshot, if Broker 1 crashes, then Partition 1, 3 and 4 will still be receiving and sending messages.

However, in standalone cluster i.e. cluster with 1 broker, if the broker crashes, the entire topic will be lost since all the partitions reside in that single broker.

# Topic with Replication in Multiple Broker Kafka Cluster

* Replication is a process of having multiple copies of the data for the sole purpose of availability in case one of the brokers goes down and is unavailable to server the request.

Topic’s Partition from one broker will be replicated to another broker i.e. In the below snippet, Topic 1 Partition 0 from Broker is replicated to Broker 2.

A diagram of a diagram

Description automatically generated

So, if 1 of the broker crashes then Partitions of that broker can be accessible from the other brokers where those partitions are replicated.

* In Kafka, replication happens at partition level or partition granularity.
* While creating Topic, replication-factor is the property used to specify the number of replicas for a partition.
* If replication factor is **r** and total partitions are **p** then there will be **p\*r** partitions (replica partitions) . For e.g., Let’s say replication factor is 2 and Partitions are 4 then there will be 2\*4=8 partitions, which includes ***replicated partitions + primary partitions***
* For all these replica partitions, there will be a Leader and In Sync replica(s). As the name suggests, **Leader** will be responsible for all the read and writes for that partition and **In Sync replica** will have data replicated from the primary partition.

e.g.

In above snippet, Topic 1 Partition 0 in Broker 1 is the Leader which will be responsible for all the read and writes.

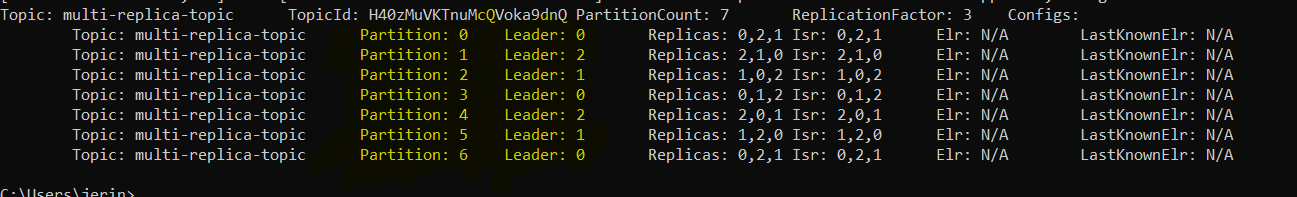
Topic 1 Partition 0 in Broker 2 is the In Sync replica which will have all the data replicated from the primary partition.

* **Leader and In-Sync Replicas (ISR)**:
  + In Kafka, each partition has one **Leader** replica, which is responsible for **handling all reads and writes** for that partition.
  + Other replicas, if any, are referred to as **In-Sync Replicas (ISR)**. These replicas keep a copy of the partition data and **replicate data from the Leader**.
  + The **Leader** ensures that all messages are replicated to the ISRs.
  + For a replica to be considered an **In-Sync Replica**, it must stay in sync with the Leader by consistently replicating the data.
  + **Key Points:** 
    - **Leader** handles all the **client requests** (both reads and writes) for a partition.
    - **In-Sync Replicas (ISRs)** are replicas that have the most up-to-date data from the Leader and are ready to take over leadership if the current Leader fails.
    - The **ISR set** may not include all replicas. Only those replicas that are synchronized with the Leader will be part of the ISR.

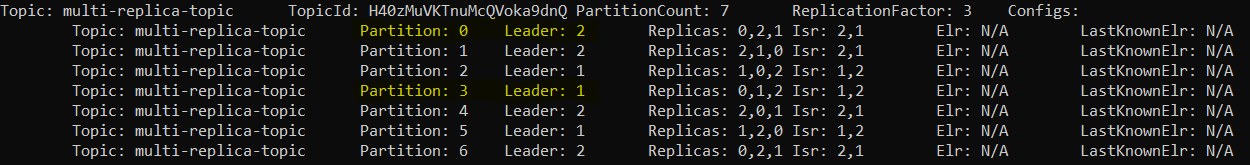
***Scripts:***

|  |
| --- |
| * **Start Zookeeper using below code - change path of zookeeper.properties**   *zookeeper-server-start D:\kafka\config\zookeeper.properties*   * **Start kafka clsuter - multiple brokers**   *kafka-server-start D:\kafka\config\server0.properties*  *kafka-server-start D:\kafka\config\server1.properties*  *kafka-server-start D:\kafka\config\server2.properties*   * **Create a topic with 7 partitions and 3 as replication-factor**   *kafka-topics --create --topic multi-replica-topic --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --replication-factor 3 --partitions 7*   * **Describe topic to check which partition is Leader**   *kafka-topics --topic multi-replica-topic --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --describe*   * **Create kafka console Producer**   *kafka-console-producer --topic multi-replica-topic --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --property key.parse=true --property "key.separator=:"*   * **Create kafka console Consumer**   *kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --topic multi-replica-topic --from-beginning --property print.key=true --property "key.separator=:"* |

* When above scripts are executed, a topic with 21 partitions will be created i.e. 3 replication-factor \* 7 partitions = 21 replica partitions.
* There will be 7 leaders and remaining will be In sync replicas.
* Total brokers in our exercise are 3 i.e. broker 0,1 and 2.
* To check, which replica partition is considered as Leader, use ***--describe*** property.

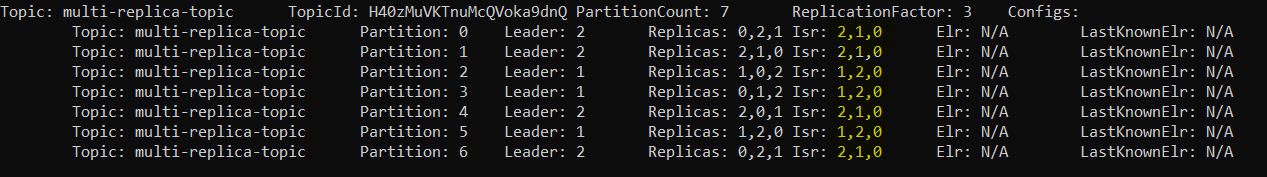


* + As per above snip, For Partition 0 Leader is Broker 0 i.e. for the given topic Partition present in Broker 0 is the Leder and other replica partitions will be In Sync replicas.
  + Replicas:0,2,1 in the above results indicate, in which broker replica partitions exist.
  + Isr denotes In Sync Replica. Isr:0,2,1 indicate, Partition 0 in broker 0,2,1 are in sync
* If one of the brokers goes down, then Kafka will promote one of the In Sync replica from other brokers as Leader.
  + For e.g. If Broker 0 crashes, then as can be seen in below screenshot:
    - Partition 0’s Leader is changed from broker 0 to Partition 0 replica present in broker 2
    - Partition 3’s Leader is changed from broker 0 to Partition 3 replica present in broker 1.



* + As can be seen in Isr section, Broker 0 is now removed from Isr.
* If the broker which earlier crashed comes back alive, then change in Leader depends on how many brokers are available.
  + Let’s say 2 out of 3 broker goes down, then Leader will be the Partitions present in the broker which is alive. Now once both the crashed brokers are up and running then Leader will be shuffled across all the available brokers. In this case, Kafka shuffle the Leader because Kafka tries to do load balancing
  + Let’s say only 1 out of 3 broker goes down, then Leader will be the Partitions present in the brokers which are alive. Now once the crashed brokers are up and running then Leader will not be shuffled.

For e.g., If Broker 0 is back online which was previously crashed, then as can be seen below there is no change in Leader.



* + As can be seen in Isr section, Broker 0 is now added back to Isr

# In-depth Intuition on Kafka Rack Awareness