Contents

[1 General Introduction 2](#_Toc37931158)

[2 Sources/Credits/Acknowledgements 2](#_Toc37931159)

[3 ZooKeeper 2](#_Toc37931160)

[4 Apache Kafka 2](#_Toc37931161)

[4.1 Important Terms 2](#_Toc37931162)

[4.2 More on Consumer Groups 7](#_Toc37931163)

[4.3 Architecture 8](#_Toc37931164)

[4.4 Kafka on Windows 10](#_Toc37931165)

[4.5 Kafkacat 11](#_Toc37931166)

# General Introduction

# Sources/Credits/Acknowledgements

<https://kafka.apache.org/documentation/> - Excellent introduction to Kafka

<https://jaceklaskowski.gitbooks.io/apache-kafka/kafka-brokers.html>

<https://sookocheff.com/post/kafka/kafka-in-a-nutshell/>

cloudera

data-flair

confluent

cloudkarafka

# ZooKeeper

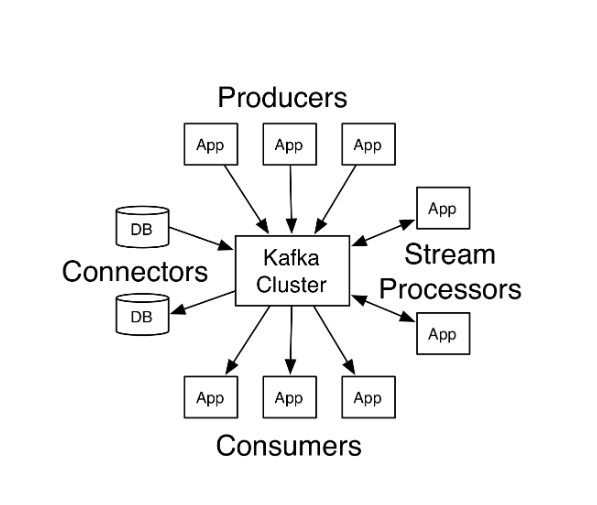
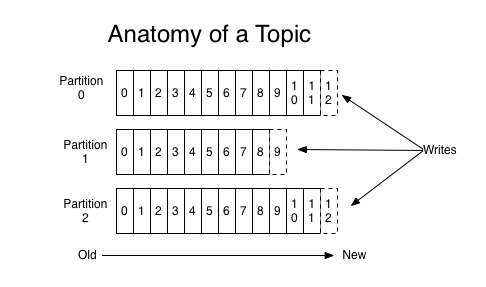
# Apache Kafka

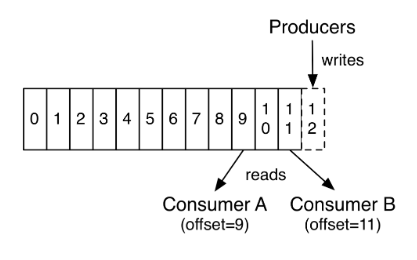
## Important Terms

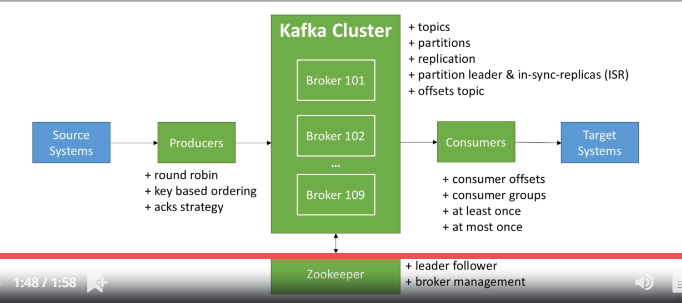
|  |  |
| --- | --- |
| **Cluster** | A running Kafka Instance that clients will connect to. Consists of an ensemble of stateless Kafka brokers that host topics. The cluster state is maintained using **ZooKeeper.** |
| **Broker** | Source - <https://jaceklaskowski.gitbooks.io/apache-kafka/kafka-brokers.html>  “A **Kafka server**, a **Kafka broker** and a **Kafka node** all refer to the same concept and are synonyms  The above line is significant. Assume that a broker is a server or a node. This assumption makes it easy to understand how Kafka works.   * A Kafka broker receives messages from producers and stores them on disk keyed by unique **offset**. * A Kafka broker allows consumers to fetch messages by topic, partition and offset. * **Kafka brokers can create a Kafka cluster by sharing information between each other directly or indirectly using Zookeeper** |
| **Partitions** | * You can lay out a topic (as partitions) across a cluster of machines to allow data streams larger than the capability of a single machine. Partitions are log files on disk with sequential write only. Kafka guarantees message ordering in a partition * Partitions mirror the physical manner in which a topic gets persisted across machines. * Partitions allow you to parallelize a topic by splitting the data in a particular topic across multiple brokers — each partition can be placed on a separate machine to allow for multiple consumers to read from a topic in parallel. Note – Multiple partitions can be on the same machine.   For a topic, the following examples are possible :-   * One can have 5 partitions across 5 brokers * 3 partitions across 4 brokers – One broker will have no partition * 7 partitions across 4 brokers – Some brokers will have more than 1 partition * > 1 partition on a single broker |
| **Producers** | These are third party applications that send streams of data / messages to topics in a Kafka cluster. |
| **Consumers** | These are third party applications that read messages from topics in a Kafka cluster. |
| **Offset** | Each partition is an ordered, immutable sequence of records that is continually appended to—a structured commit log. The records in the partitions are each assigned a sequential id number called the *offset* that uniquely identifies each record within the partition.  An offset is relevant only in the context of a specific topic and partition. |
| **Streams** |  |
| **Connectors** |  |
| **Topics** | <https://jaceklaskowski.gitbooks.io/apache-kafka/kafka-brokers.html>   * **Producers will always write to a topic**      * A Kafka topic is spread across a Kafka cluster as a logical group of one or more **partitions** * **Topics store messages and are partitioned and replicated across multiple brokers** * **Topics** are virtual groups of one or many [partitions](https://jaceklaskowski.gitbooks.io/apache-kafka/kafka-topics.html#partitions) across [Kafka brokers](https://jaceklaskowski.gitbooks.io/apache-kafka/kafka-brokers.html) in a Kafka cluster. |
| **Zookeeper** | Service that provides distributed co-ordination |
| **log end offset** | The **log end offset** is the offset of the last message written to a log. |
| **high watermark offset** | The **high watermark offset** is the offset of the last message that was successfully copied to all of the log’s replicas. |
| **Replication-factor** | See “Replicas” below |
| **Leaders** |  |
| **Followers** |  |
| **Replicas** | Topics are partitioned (split ) across multiple nodes so a topic can grow beyond the limits of a node. Partitions are replicated for fault tolerance.  Partitions in Kafka are like buckets within a topic used for better load balancing when you are dealing with large throughput where you can as many consumers as your partitions to process your data. For example if you are processing orders you can send all orders for iPad into one partition “0” and iPhones in another partition “1” of the same Orders Topic and you can have two services one processing iPhone order from partition “1” and iPad from partition “0” and also maintain ordering of messages per partition. If a consumer service goes down the another service will pickup messages from both partition until another service comes backup into same consumer group processing orders topic.  Whereas replication factor determines how many backups (copies) you want to create for your topic in the Kafka cluster for enabling HA for your data in times of cluster losing a node due to various reasons. |
| **Consumer groups** | * Consumers can join a group by using the samegroup.id. * The maximum parallelism of a group is that the number of consumers in the group ← no of partitions. * Kafka assigns the partitions of a topic to the consumer in a group, so that each partition is consumed by exactly one consumer in the group. * Kafka guarantees that a message is only ever read by a single consumer in the group. * Consumers can see the message in the order they were stored in the log. * Kafka automatically shares or re-balances the load between consumers in the same group when consumers are added or removed * Consumer Groups can subscribe to one or more topics   Schematic Representations of Consumer Groups |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Architecture

Kafka is run as a cluster on one or more servers that can span multiple datacenters

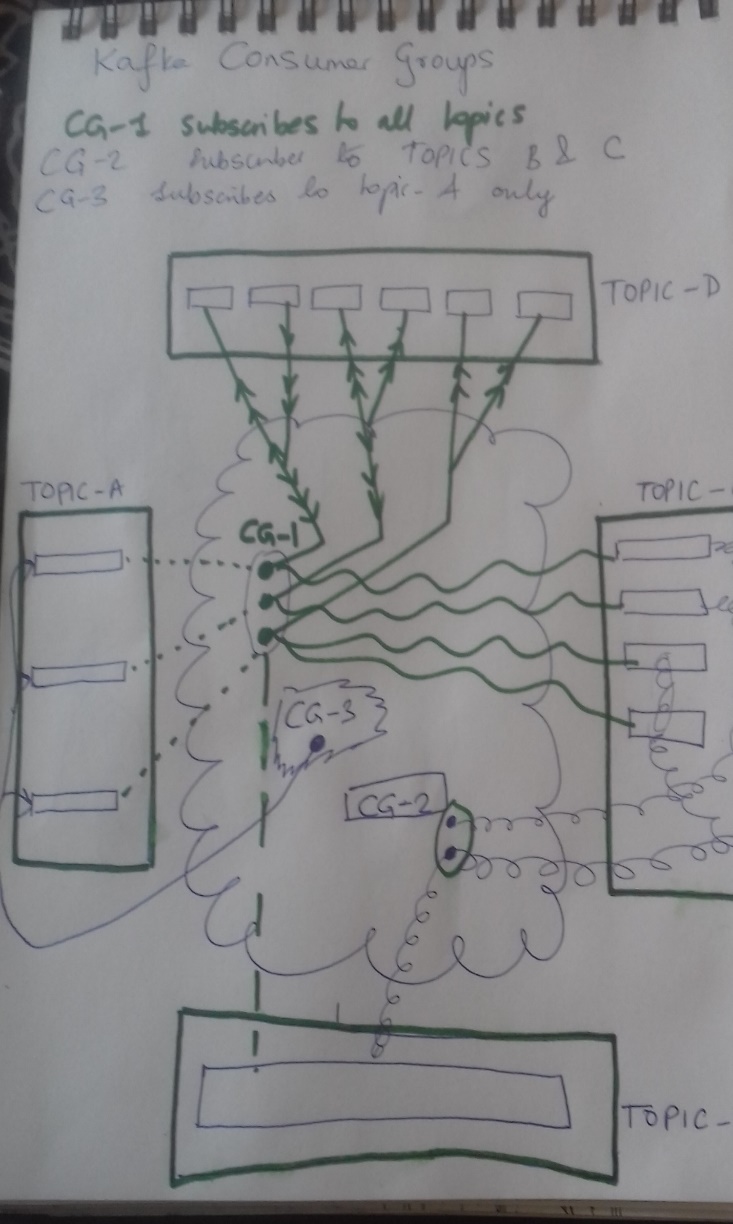




Source - <https://www.udemy.com/course/apache-kafka/learn/quiz/4494080#overview>

## More on Producers

## More on Consumer Groups





1. Consumer Groups can subscribe to multiple topics (>=1)
2. Within a consumer group, a partition will be read by only 1 consumer

(Two or more consumers WITHIN THE SAME CONSUMER GROUP will never read from the same topic-partition)

1. A consumer within the same consumer group can read messages from multiple partitions for a topic
2. A consumer within a consumer group can be inactive/idle
3. The same topic can be consumed by multiple consumer groups
   1. For a topic and a specific partition, messages can be processed by different consumers PROVIDED each consumer is in a separate consumer group
   2. One can easily draw a diagram to visualize this scenario
4. Resetting of offsets

## Kafka on Windows

Source/Credits - <https://www.learningjournal.guru/article/kafka/installing-kafka-on-windows/>

|  |  |
| --- | --- |
| 1 | config\zookeeper.properties  {Log/data Directory for ZooKeeper} |
| 2 | server.properties  {Log/data/config settings for Kafka} |
| 3 | PATH |
| 4 | \bin\windows\zookeeper-server-start.bat ………….\config\zookeeper.properties |
| 5 | \bin\windows\ kafka-server-start.bat …………….\config\server.properties |
| 6 | zookeeper-shell.bat localhost:2181 ls /brokers/ids |
| 7 | kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test |
| 8 | kafka-topics.bat --list --zookeeper localhost:2181 |
| 9 | kafka-console-producer.bat --broker-list localhost:9092 --topic test |
| 10 | kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning |

|  |
| --- |
| ###################################  #  #  #  docker network rm windockkaf  docker volume rm my-zookeeper-vol  docker volume rm my-kafka-vol  docker network create windockkaf  docker volume create my-zookeeper-vol  docker volume create my-kafka-vol  #  # Creating and running the Zookeeper container  docker run –-network=windockkaf -d –-name=zookeeper -p 2181:2181 -e ZOOKEEPER\_CLIENT\_PORT=2181 -e ALLOW\_ANONYMOUS\_LOGIN=yes -v my-zookeeper-vol:/bitnami bitnami/zookeeper:3  #  #  # KAFKA\_CFG\_ZOOKEEPER\_CONNECT=zookeeper:2181 or KAFKA\_CFG\_ZOOKEEPER\_CONNECT=localhost:2181  #  #  # Statement below will not work  docker run –-network=windockkaf -d -p 9092:9092 -p 29092:29092 –-name=kafka -e KAFKA\_CFG\_ZOOKEEPER\_CONNECT=localhost:2181 -e ALLOW\_PLAINTEXT\_LISTENER=yes -e KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT -e KAFKA\_CFG\_LISTENERS=PLAINTEXT://:9092,PLAINTEXT\_HOST://:29092 -e KAFKA\_CFG\_ADVERTISED\_LISTENERS=PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092 -e KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR=1 -e KAFKA\_ADVERTISED\_HOST\_NAME=localhost -v my-kafka-vol:/bitnami bitnami/kafka:2  #  # This works  **# make sure that in the hosts file - 127.0.0.1 kafka is set**  docker run –-network=windockkaf -d -p 9092:9092 -p 29092:29092 –-name=kafka -e KAFKA\_CFG\_ZOOKEEPER\_CONNECT=zookeeper:2181 -e ALLOW\_PLAINTEXT\_LISTENER=yes -e KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT -e KAFKA\_CFG\_LISTENERS=PLAINTEXT://:9092,PLAINTEXT\_HOST://:29092 -e KAFKA\_CFG\_ADVERTISED\_LISTENERS=PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092 -e KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR=1 -e KAFKA\_ADVERTISED\_HOST\_NAME=localhost -v my-kafka-vol:/bitnami bitnami/kafka:2  # |

## Kafkacat