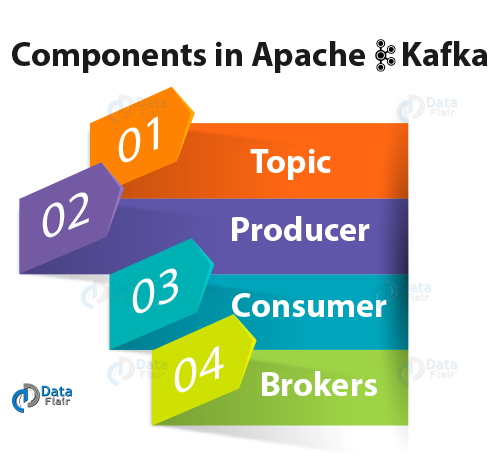
**Q.1 What is Apache Kafka?**

**Ans.** Apache Kafka is a publish-subscribe open source message broker application. This messaging application was coded in “[**Scala**](https://data-flair.training/blogs/scala-tutorial/)”. Basically, this project was started by the Apache software. Kafka’s design pattern is mainly based on the transactional logs design.

For detailed understanding of Kafka, go through, [**Kafka Tutorial**](https://data-flair.training/blogs/apache-kafka-tutorial/)

**Q.2 Enlist the several components in Kafka.**

**Ans.** The most important elements of Kafka are:

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/components-in-Apache-Kafka.png)

Kafka Interview Questions- Components of Kafka

* **Topic –**

Kafka Topic is the bunch or a collection of messages.

* **Producer –**

In Kafka, Producers issue communications as well as publishes messages to a Kafka topic.

* **Consumer –**

Kafka Consumers subscribes to a topic(s) and also reads and processes messages from the topic(s).

* **Brokers –**

While it comes to manage storage of messages in the topic(s) we use Kafka Brokers.

For detailed understanding of Kafka components, go through, [**Kafka – Architecture**](https://data-flair.training/blogs/kafka-architecture/)



**Q.3 Explain the role of the offset.**

**Ans.** There is a sequential ID number given to the messages in the partitions what we call, an offset. So, to identify each message in the partition uniquely, we use these offsets.

Each partition maintains the messages it has received in a sequential order where they are identified by an offset

**Q.4 What is a Consumer Group?**

**Ans.** The concept of Consumer Groups is exclusive to Apache Kafka. Basically, every Kafka consumer group consists of one or more consumers that jointly consume a set of subscribed topics.

For details, follow the link:[**Kafka Consumer Group**](https://data-flair.training/blogs/kafka-consumer/)

**Q.5 What is the role of the ZooKeeper in Kafka?**

**Ans.** Apache Kafka is a distributed system is built to use Zookeeper. Although, Zookeeper’s main role here is to build coordination between different nodes in a cluster. However, we also use Zookeeper to recover from previously committed offset if any node fails because it works as periodically commit offset.

**Q.6 Is it possible to use Kafka without ZooKeeper?**

**Ans.** It is impossible to bypass Zookeeper and connect directly to the Kafka server, so the answer is no. If somehow, ZooKeeper is down, then it is impossible to service any client request.

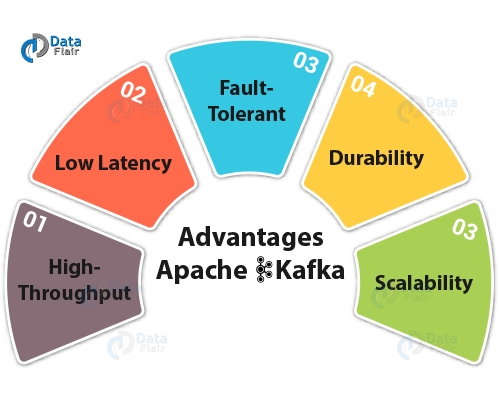
**Q.7 What do you know about Partition in Kafka?**

**Ans.** In every Kafka broker, there are few partitions available. And, here each partition in Kafka can be either a leader or a replica of a topic.

Kafka topics are divided into a number of partitions. 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. Consumers can also be parallelized so that multiple consumers can read from multiple partitions in a topic allowing for very high message processing throughput.

**Q.8 Why is Kafka technology significant to use?**

**Ans.** There are some advantages of Kafka, which makes it significant to use:

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/advantages-of-Kafka.png)

(HDFS LoL)

Kafka Interview Questions & Answers- Advantages of Apache Kafka

* **High-throughput**

We do not need any large hardware in Kafka, because it is capable of handling high-velocity and high-volume data. Moreover, it can also support message throughput of thousands of messages per second.

* **Low Latency**

Kafka can easily handle these messages with the very low latency of the range of milliseconds, demanded by most of the new use cases.

* **Fault-Tolerant**

Kafka is resistant to node/machine failure within a cluster.

* **Durability**

As Kafka supports messages replication, so, messages are never lost. It is one of the reasons behind durability.

* **Scalability**

Kafka can be scaled-out, without incurring any downtime on the fly by adding additional nodes.

To learn all advantages in detail, follow the link: [**Kafka – Pros & Cons**](https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/)

**Disadvantages**

**No Complete Set of Monitoring Tools**: It is seen that it lacks a full set of management and monitoring tools. Hence, enterprise support staff felt anxious or fearful about choosing Kafka and supporting it in the long run.

**Q.9 What are main APIs of Kafka?**

**Ans.** Apache Kafka has 4 main APIs:

1. Producer API
2. Consumer API
3. Streams API
4. Connector API

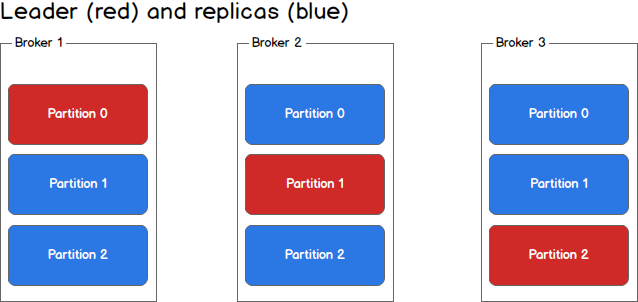
**Q.10 What are consumers or users?**

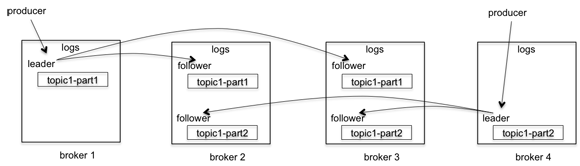
**Ans.** Mainly, [**Kafka Consumer**](https://data-flair.training/blogs/kafka-consumer/) subscribes to a topic(s), and also reads and processes messages from the topic(s). Moreover, with a consumer group name, Consumers label themselves. In other words, within each subscribing consumer group,

a topic is delivered to one consumer instance. Make sure it is possible that Consumer instances can be in separate processes or on separate machines.

**Q.11 Explain the concept of Leader and Follower.**

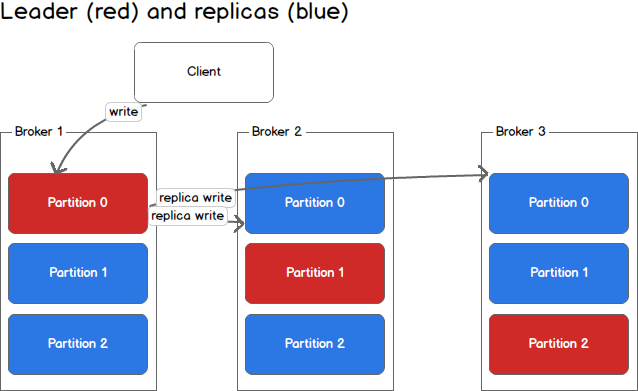
Ans. Each broker holds a number of partitions and each of these partitions can be either a leader or a replica for a topic. All writes and reads to a topic go through the leader and the leader coordinates updating replicas with new data. If a leader fails, a replica takes over as the new leader.





**Q.12 What ensures load balancing of the server in Kafka?**

Ans. As the main role of the Leader is to perform the task of all read and write requests for the partition, whereas Followers passively replicate the leader. Hence, at the time of Leader failing, one of the Followers takeover the role of the Leader. Basically, this entire process ensures load balancing of the servers.



**Q.13 what roles do Replicas and the ISR play?**

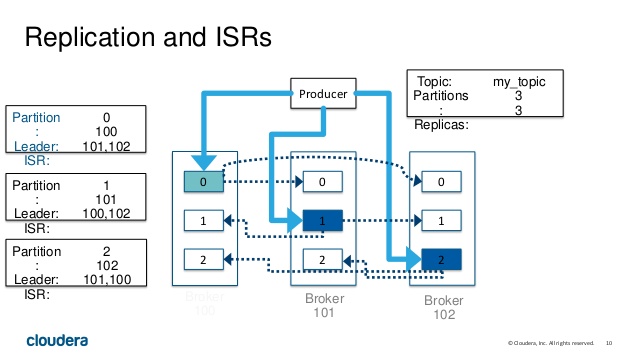
Ans. Basically, a list of nodes that replicate the log is Replicas. Especially, for a particular partition. However, they are irrespective of whether they play the role of the Leader.

When a producer publishes a message to a partition in a topic, the message is first forwarded to the leader replica of the partition and is appended to its log. The follower replicas keep pulling new messages from the leader. Once enough replicas have received the message, the leader commits it.

In addition, ISR refers to In-Sync Replicas. On defining ISR, it is a set of message replicas that are synced to the leaders.

One subtle issue is how the leader decides what's enough. The leader can't always wait for writes to complete on all replicas. This is because any follower replica can fail and the leader can't wait indefinitely.

To address this problem, for each partition of a topic, we maintain an in-sync replica set (ISR). This is the set of replicas that are alive and have fully caught up with the leader (note that the leader is always in ISR). When a partition is created initially, every replica is in the ISR. When a new message is published, the leader waits until it reaches all replicas in the ISR before committing the message. If a follower replica fails, it will be dropped out of the ISR and the leader then continues to commit new messages with fewer replicas in the ISR. Notice that now, the system is running in an under replicated mode.



**Q.14 Why are Replications critical in Kafka?**

Ans. Because of Replication, we can be sure that published messages are not lost and can be consumed in the event of any machine error, program error or frequent software upgrades.

**Q.15 If a Replica stays out of the ISR for a long time, what does it signify?**

Ans. Simply, it implies that the Follower cannot fetch data as fast as data accumulated by the Leader.

**Q.16 What is the process for starting a Kafka server?**

Ans. It is the very important step to initialize the ZooKeeper server because Kafka uses ZooKeeper.So, the process for starting a Kafka server is:

**In order to start the ZooKeeper server: > bin/zookeeper-server-start.sh config/zookeeper.properties**

**Next, to start the Kafka server: > bin/kafka-server-start.sh config/server.properties**

**Q.17 In the Producer, when does QueueFullException occur?**

Ans. whenever the [**Kafka Producer**](https://data-flair.training/blogs/kafka-producer/) attempts to send messages at a pace that the Broker cannot handle at that time QueueFullException typically occurs. However, to collaboratively handle the increased load, users will need to add enough brokers, since the Producer doesn’t block.

**Q.18 Explain the role of the Kafka Producer API.**

Ans. An API which permits an application to publish a stream of records to one or more Kafka topics is what we call Producer API.

**Q.19 What is the main difference between Kafka and Flume?**

Ans. The main difference between Kafka and Flume are:

* **Types of tool**

**Apache Kafka–**as Kafka is a general-purpose tool for both multiple producers and consumers.

**Apache Flume–** Whereas, Flume is considered as a special-purpose tool for specific applications.

* **Replication feature**

**Apache Kafka–**  Kafka can replicate the events.

**Apache Flume-** whereas, Flume does not replicate the events.

**Q.20 Is Apache Kafka is a distributed streaming platform? if yes, what you can do with it?**

Ans. Undoubtedly, Kafka is a streaming platform. It can help:

1. To push records easily
2. Also, can store a lot of records without giving any storage problems
3. Moreover, it can process the records as they come in

**Q. 21 what can you do with Kafka?**

**Ans.** It can perform in several ways, such as:

>> In order to transmit data between two systems, we can build a real-time stream of data pipelines with it.

>> Also, we can build a real-time streaming platform with Kafka, that can actually react to the data.

**Q.22 What is the purpose of retention period in Kafka cluster?**

**Ans.** However, retention period retains all the published records within the Kafka cluster. It doesn’t check whether they have been consumed or not. Moreover, the records can be discarded by using a configuration setting for the retention period. And, it results as it can free up some space.

**Q.23 Explain the maximum size of a message that can be received by the Kafka?**

**Ans.** The maximum size of a message that can be received by the Kafka is approx. 1000000 bytes.

**Q.24 What are the types of traditional method of message transfer?**

**Ans.** Basically, there are two methods of the traditional message transfer method, such as:

* **Queuing:** It is a method in which a pool of consumers may read a message from the server and each message goes to one of them.
* **Publish-Subscribe:** Whereas in Publish-Subscribe, messages are broadcasted to all consumers.

**Q.25 What does ISR stand in Kafka environment?**

**Ans.** ISR refers to In sync replicas. These are generally classified as a set of message replicas which are synced to be leaders.

**Q.26 What is Geo-Replication in Kafka?**

**Ans.** For our cluster, Kafka MirrorMaker offers geo-replication. Basically, messages are replicated across multiple data centers or cloud regions, with MirrorMaker. So, it can be used in active/passive scenarios for backup and recovery; or also to place data closer to our users, or support data locality requirements.

**Q.27 Explain Multi-tenancy?**

**Ans.** We can easily deploy Kafka as a multi-tenant solution. However, by configuring which topics can produce or consume data, Multi-tenancy is enabled. Also, it provides operations support for quotas.

**Q.28 What is the role of Consumer API?**

**Ans.** An API which permits an application to subscribe to one or more topics and also to process the stream of records produced to them is what we call Consumer API.

**Q.29 Explain the role of Streams API?**

**Ans.** An API which permits an application to act as a stream processor, and also consuming an input stream from one or more topics and producing an output stream to one or more output topics, moreover, transforming the input streams to output streams effectively, is what we call Streams API.

**Q.30 What is the role of Connector API?**

**Ans.** An API which permits to run as well as build the reusable producers or consumers which connect Kafka topics to existing applications or data systems is what we call the Connector API.

**Q.31 Explain Producer?**

**Ans.** The main role of Producers is to publish data to the topics of their choice. Basically, its duty is to select the record to assign to partition within the topic.

**Q.32 Compare: RabbitMQ vs Apache Kafka**

**Ans.** One of the Apache Kafka’s alternative is RabbitMQ. So, let’s compare both:

**i. Features**

Apache Kafka– Kafka is distributed, durable and highly available, here the data is shared as well as replicated.

RabbitMQ– There are no such features in RabbitMQ.

**ii. Performance rate**

Apache Kafka– To the tune of 100,000 messages/second.

RabbitMQ- In case of RabbitMQ, the performance rate is around 20,000 messages/second.

**Q.33 Compare: Traditional queuing systems vs Apache Kafka**

**Ans.** Let’s compare Traditional queuing systems vs Apache Kafka feature-wise:

* **Messages Retaining**

Traditional queuing systems– It deletes the messages just after processing completion typically from the end of the queue.

Apache Kafka– But in Kafka, messages persist even after being processed. That implies messages in Kafka don’t get removed as consumers receive them.

* **Logic-based processing**

Traditional queuing systems–Traditional queuing systems don’t permit to process logic based on similar messages or events.

Apache Kafka– Kafka permits to process logic based on similar messages or events.

**Q.34 Why Should we use Apache Kafka Cluster?**

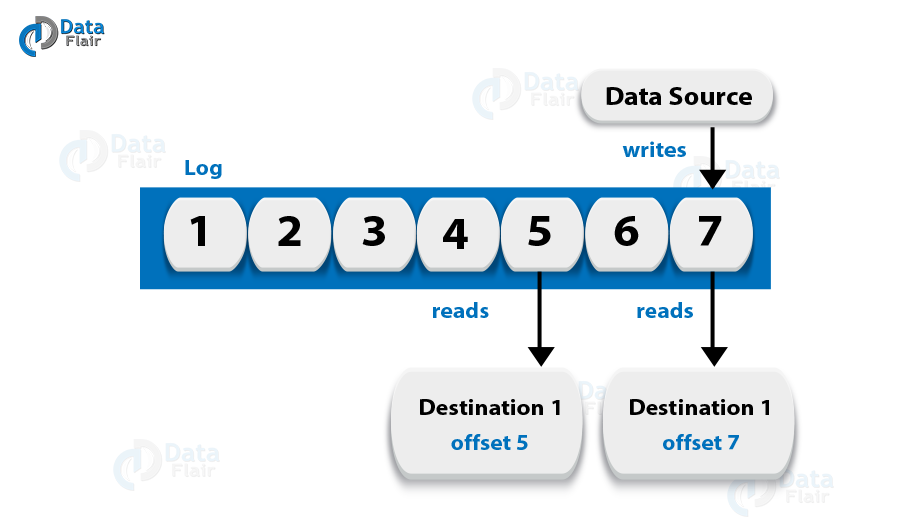
**Ans.** In order to overcome the challenges of collecting the large volume of data, and analyzing the collected data we need a messaging system. Hence Apache Kafka came in the story. Its benefits are:

* It is possible to track web activities just by storing/sending the events for real-time processes.
* Through this, we can Alert as well as report the operational metrics.
* Also, we can transform data into the standard format.
* Moreover, it allows continuous processing of streaming data to the topics.

Due to its this wide use, it is ruling over some of the most popular applications like ActiveMQ, RabbitMQ, AWS etc.

**Q.35 Explain the term “Log Anatomy”.**

**Ans.** We view log as the partitions. Basically, a data source writes messages to the log. One of the advantages is, at any time one or more consumers read from the log they select. Here, below diagram shows a log is being written by the data source and the log is being read by consumers at different offsets.



Kafka Interview Questions- Log Anatomy

**Q.36 What is Data Log in Kafka?**

**Ans.** As we know, messages are retained for a considerable amount of time in Kafka. Moreover, there is flexibility for consumers that they can read as per their convenience. Although, there is a possible case that if Kafka is configured to keep messages for 24 hours and possibly that time consumer is down for time greater than 24 hours, then the consumer may lose those messages. However, still, we can read those messages from last known offset, but only at a condition that the downtime on part of the consumer is just 60 minutes. Moreover, on what consumers are reading from a topic Kafka doesn’t keep state.

**Q.37 Explain how to Tune Kafka for Optimal Performance.**

**Ans.** So, ways to tune Apache Kafka it is to tune its several components:

1. Tuning Kafka Producers
2. Kafka Brokers Tuning
3. Tuning Kafka Consumers

Learn it in detail, follow the link: [**Kafka Performance Tuning – Ways for Kafka Optimization**](https://data-flair.training/blogs/kafka-performance-tuning/)

**Q.38 State Disadvantages of Apache Kafka.**

**Ans.** Limitations of Kafka are:

1. No Complete Set of Monitoring Tools
2. Issues with Message Tweaking
3. Not support wildcard topic selection
4. Lack of Pace

Learn it in detail, follow the link:[**Kafka – Pros & Cons**](https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/)

**Q.39 Enlist all Apache Kafka Operations.**

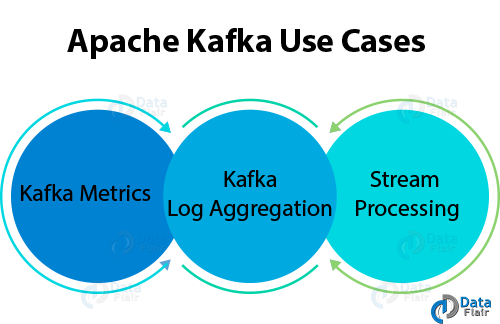
**Ans.** Apache Kafka Operations are:

1. Addition and Deletion of Kafka Topics
2. How to modify the Kafka Topics
3. Distinguished Turnoff
4. Mirroring Data between Kafka Clusters
5. Finding the position of the Consumer
6. Expanding Your Kafka Cluster
7. Migration of Data Automatically
8. Retiring Servers
9. Datacenters

Learn each operation in detail, follow the link: [**Kafka – Operations**](https://data-flair.training/blogs/kafka-operations/)

**Q.40 Explain Apache Kafka Use Cases?**

**Ans.** Apache Kafka has so many use cases, such as:

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Apache-Kafka-Use-Cases.png)

Kafka Interview Question- Use Cases of Kafka

* **Kafka Metrics**

It is possible to use Kafka for operational monitoring data. Also, to produce centralized feeds of operational data, it involves aggregating statistics from distributed applications.

* **Kafka Log Aggregation**

Moreover, to gather logs from multiple services across an organization.

* **Stream Processing**

While stream processing, Kafka’s strong durability is very useful.

**Q.41 Some of the most notable applications of Kafka.**

**Ans.** Some of the real-time applications are:

1. Netflix
2. Mozilla
3. Oracle

Learn more and detailed applications in detail, follow the link [**Kafka – Applications**](https://data-flair.training/blogs/kafka-use-cases-and-applications/)

**Q.42 Features of Kafka Stream.**

**Ans.** Some best features of Kafka Stream are

* Kafka Streams are highly scalable and fault-tolerant.
* Kafka deploys to containers, VMs, bare metal, cloud.
* We can say, Kafka streams are equally viable for small, medium, & large use cases.
* Also, it is fully in integration with Kafka security.
* Write standard Java applications.
* Exactly-once processing semantics.
* Moreover, there is no need of separate processing cluster.

**Q.43 What do you mean by Stream Processing in Kafka?**

**Ans.** The type of processing of data continuously, real-time, concurrently, and in a record-by-record fashion is what we call Kafka Stream processing.

**Q.44 What are the types of System tools?**

**Ans.** There are three types of System tools:

* **Kafka Migration Tool**

It helps to migrate a broker from one version to another.

* **Mirror Maker**

Mirror Maker tool helps to offer to mirror of one Kafka cluster to another.

* **Consumer Offset Checker**

For the specified set of Topics as well as Consumer Group, it shows Topic, Partitions, Owner.

**Q.45 What are Replication Tool and its types?**

**Ans.** For the purpose of stronger durability and higher availability, replication tool is available here. Its types are −

* Create Topic Tool
* List Topic Tool
* Add Partition Tool

**Q.46 What is Importance of Java in Apache Kafka?**

**Ans.** For the need of the high processing rates that come standard on Kafka, we can use java language. Moreover, for Kafka consumer clients also, Java offers a good community support. So, we can say it is a right choice to implement Kafka in Java.

**Q.47 State one best feature of Kafka.**

**Ans.** The best feature of Kafka is “Variety of Use Cases”.

It means Kafka is able to manage the variety of use cases which are very common for a Data Lake. For Example log aggregation, web activity tracking, and so on.

**Q.48 Explain the term “Topic Replication Factor”.**

**Ans.** It is very important to factor in topic replication while designing a Kafka system. Hence, if in any case, broker goes down its topics’ replicas from another broker can solve the crisis.

**Q.49 Explain some Kafka Streams real-time Use Cases.**

**Ans.** So, the use cases are:

* **The New York Times**

This company uses it to store and distribute, in real-time, published content to the various applications and systems that make it available to the readers. Basically, it uses Apache Kafka and the Kafka Streams both.

* **Zalando**

As an ESB (**Enterprise Service Bus**) as the leading online fashion retailer in Europe Zalando uses Kafka.

* **LINE**

Basically, to communicate to one another LINE application uses Apache Kafka as a central data hub for their services.

**Q.50 What are Guarantees provided by Kafka?**

**Ans.** They are:

* The order will be same for both the Messages sent by a producer to a particular topic partition. That
* Moreover, the consumer instance sees records in the order in which they are stored in the log.
* Also, we can tolerate up to N-1 server failures, even without losing any records committed to the log.

**2. What is Kafka Performance Tuning?**

There are few configuration parameters to be considered while we talk about Kafka Performance tuning. Hence, to improve performance, the most important configurations are the one, which controls the disk flush rate.

Also, we can divide these configurations on the component basis. So, let’s talk about Producer first. Hence, most important configurations which need to be taken care at [**Producer**](https://data-flair.training/blogs/kafka-producer/) side are –

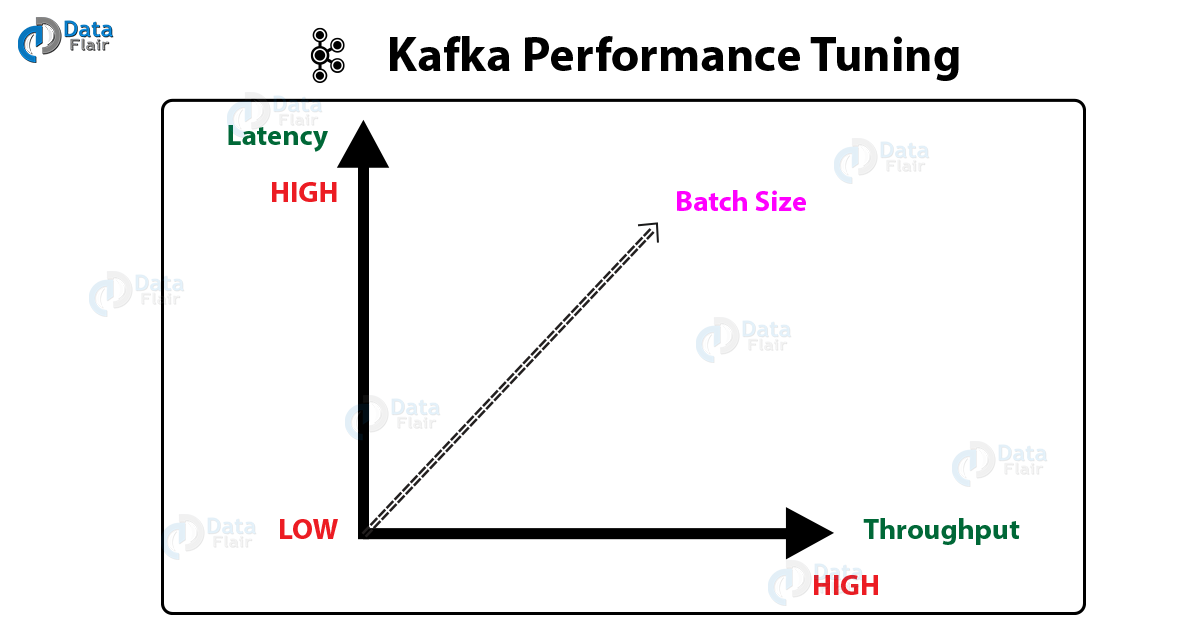
* Compression
* Batch size
* Sync or Async

And, at [**Consumer**](https://data-flair.training/blogs/kafka-consumer/) side the important configuration is –

* Fetch size

Although, it’s always confusing what batch size will be optimal when we think about batch size. We can say, large batch size may be great to have high throughput, it comes with latency issue. That implies latency and throughput is inversely proportional to each other.

It is possible to have low latency with high throughput where we have to choose a proper batch-size for that use queue-time or refresh-interval to find the required right balance.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Kafka-Performance-Tunning-1.png)

Kafka Performance Tuning Graph

## 3. Tuning Kafka for Optimal Performance

To be more specific, tuning involves two important metrics: Latency measures and throughput measures. Latency measures mean how long it takes to process one event, and similarly, how many events arrive within a specific amount of time, that means throughput measures. So, most systems are optimized for either latency or throughput, while Apache Kafka balances both. Moreover, we can say, a well-tuned Kafka system has just enough brokers to handle topic throughput, given the latency required to process information as it is received.

### a. Tuning Kafka Producers

As we know, Kafka uses an asynchronous [**publish/subscribe model**](https://data-flair.training/blogs/kafka-workflow/). While our producer calls the send() command, the result returned is a future. That future offers methods to check the status of the information in the process. Moreover, as the batch is ready, the producer sends it to the broker. Basically, broker waits for an event, then, receives the result, and further responds that the transaction is complete.

For latency and throughput, two parameters are particularly important:

[**Apache Kafka Use cases | Kafka Applications**](https://data-flair.training/blogs/kafka-use-cases-and-applications/)

#### i. Batch Size

Instead of the number of messages, batch.size measures batch size in total bytes. That means it controls how many bytes of data to collect, before sending messages to the Kafka broker. So, without exceeding available memory, set this as high as possible. Make sure the default value is 16384.

However, it might never get full, if we increase the size of our buffer. On the basis of other triggers, such as linger time in milliseconds, the Producer sends the information eventually. Although by setting the buffer batch size too high, we can impair memory usage, that does not impact latency.

Moreover, we are probably getting the best throughput possible, if our producer is sending all the time. Also, we might not be writing enough data to warrant the current allocation of resources, if the producer is often idle.

#### ii. Linger Time

In order to buffer data in asynchronous mode, linger.ms sets the maximum time. Let’s understand it with an example, a setting of 100 batches 100ms of messages to send at once. Here, the buffering adds message delivery latency but this improves throughput.

[**Read Apache Kafka Streams | Stream Processing Topology**](https://data-flair.training/blogs/kafka-streams/)

However, the producer does not wait, by default. Hence, it sends the buffer any time data is available.

Also, we can set linger.ms to 5 and send more messages in one batch, rather than sending immediately. This would add up to 5 milliseconds of latency to records sent, but also reduce the number of requests sent, even if the load on the system does not warrant the delay.

So, for higher latency and higher throughput in our producer, increase linger.ms.

### b. Tuning Kafka Brokers

As we know, Topics are divided into partitions. Further, each partition has a leader. Also, with multiple replicas, most partitions are written into leaders. However, if the leaders are not balanced properly, it might be possible that one might be overworked, compared to others.

So, on the basis of our system or how critical our data is, we want to be sure that we have sufficient replication sets to preserve our data. It is recommended that starting with one partition per physical storage disk and one consumer per partition.

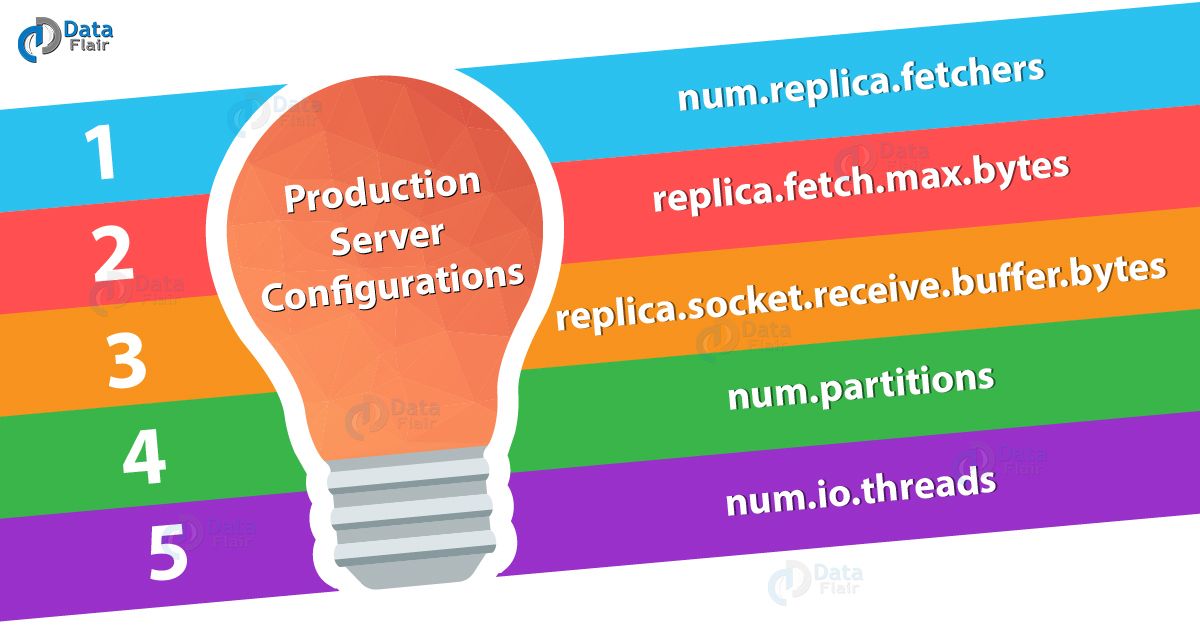
[**Read Apache Kafka Workflow | Kafka Pub-Sub Messaging**](https://data-flair.training/blogs/kafka-workflow/)

### c. Tuning Kafka Consumers

Basically, Kafka Consumers can create throughput issues. It is must that number of consumers for a topic is equal to the number of partitions. Because, to handle all the consumers needed to keep up with the producers, we need enough partitions.

In the same consumer group, consumers split the partitions among them. Hence, adding more consumers to a group can enhance performance, also adding more consumer groups does not affect performance.

Moreover, the way we use the -replica.high.watermark.checkpoint.interval.ms property, can affect throughput. Also, we can mark the last point where we read information while reading from a partition. In this way, we have a checkpoint from which to move forward without having to reread prior data, if we have to go back and locate missing data. So, we will never lose a message, if we set the checkpoint watermark for every event, but it significantly impacts performance. Also, we have a margin of safety with much less impact on throughput, if, instead, we set it to check the offset every hundred messages.



### a. num.replica.fetchers

This parameter defines the number of threads which will be replicating data from leader to the follower. As per availability of thread, we can modify the value of this parameter. It is important to have the number of replica fetchers to complete replication in parallel if we have threads available.

### b. replica.fetch.max.bytes

This parameter is all about how much data we want to fetch from any partition in each fetch request. It’s good to increase value for this parameter so, that it helps to create replica fast in the followers.

### c. replica.socket.receive.buffer.bytes

We can increase the size of a buffer if we have less thread available for creating the replica. Also, if replication thread is slow as compared to the incoming message rate, it will help to hold more data.

### d. num.partitions

While having Kafka in live, we should take care of this configuration. We can have the level of parallelism and write data in parallel, that will automatically increase the throughput.

[**Learn Storm Kafka Integration With Configurations and Code**](https://data-flair.training/blogs/storm-kafka-integration/)

However, if the system configuration is not capable to handle then increasing the number of the partition can slow down our performance and throughput. Basically, if a system does not have sufficient threads or just have single disk then it does not make sense in creating lots of partition for better throughput. So, we can say, the creation of more partition for a topic is directly dependent on available threads and disk.

### e. num.io.threads

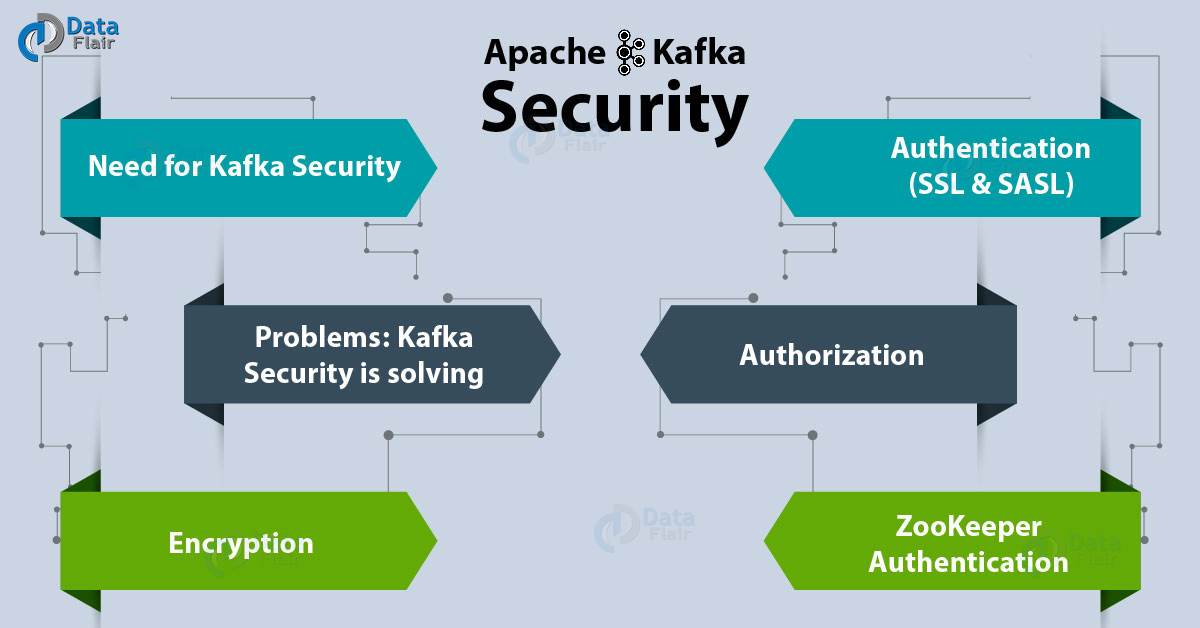
Basically, how much disk we have in our cluster, that decides setting value for I/O threads. Moreover, a server uses these threads for executing the request. Hence, a number of threads must depend on a number of the disk.

So, this was all about Kafka Performance Tuning. Hope you like our explanation.

## Kafka Security

In this Kafka article “Apache Kafka Security”, we will learn the concept of [**Apache Kafka**](https://data-flair.training/blogs/apache-kafka-tutorial/) Security. It includes why we need security, introduction to encryption as well as ZooKeeper authentication in detail. We will also see the list of problems which Kafka Security can solve easily. Moreover, we will see Kafka authentication and authorization. Also, we will look at ZooKeeper Authentication.

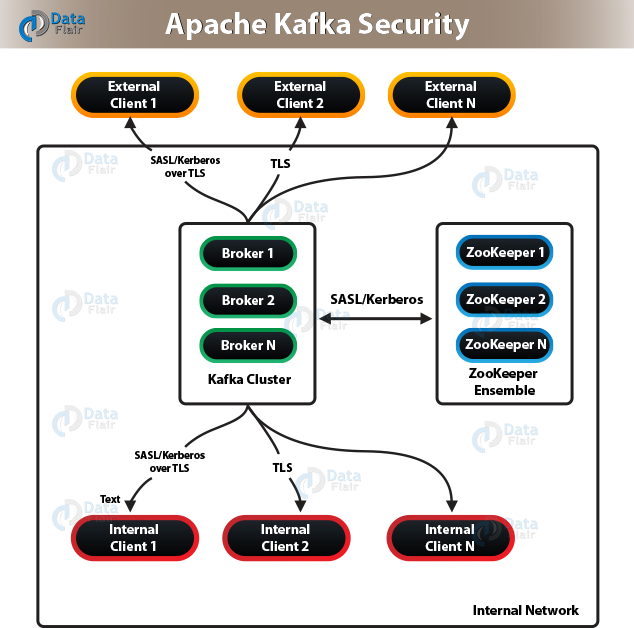
So, let’s begin Apache Kafka Security.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Apache-Kafka-Security-01.jpg)

Apache Kafka Security | Need and Components of Kafka

## 2. Apache Kafka Security

There are a number of features added in Kafka community, in release 0.9.0.0. There is a flexibility for their usage also, like either separately or together, that also enhances security in a [**Kafka cluster**](https://data-flair.training/blogs/kafka-cluster/).

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Apache-Kafka-Security-1.png)

Apache Kafka Security Working

So, the list of currently supported security measures are:

1. By using either SSL or SASL, authentication of connections to [**Kafka Brokers**](https://data-flair.training/blogs/kafka-broker/) from clients, other tools are possible. It supports various SASL mechanisms:

* **SASL/GSSAPI (Kerberos)** – starting at version 0.9.0.0
* **SASL/PLAIN** – starting at version 0.10.0.0
* **SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512** – starting at version 0.10.2.0

2. Also, offers authentication of connections from [**brokers**](https://data-flair.training/blogs/kafka-broker/) to [**ZooKeeper**](https://data-flair.training/blogs/zookeeper-tutorial/).

3. Moreover, it provides encryption of data which is transferring between brokers and [**Kafka clients**](https://data-flair.training/blogs/kafka-clients/) or between brokers and tools using SSL, that includes:

* Authorization of reading/write operations by clients.
* Here, authorization is pluggable and also supports integration with external authorization services.

**Note:** Make sure that security is optional.

## 3. Need for Kafka Security

Basically, Apache Kafka plays the role as an internal middle layer, which enables our back-end systems to share real-time data feeds with each other through Kafka topics. Generally, any user or application can write any messages to any topic, as well as read data from any topics, with a standard Kafka setup. However, it is a required to implement Kafka security when our company moves towards a shared tenancy model while multiple teams and applications use the same Kafka Cluster, or also when Kafka Cluster starts on boarding some critical and confidential information.

## 4. Problems: Kafka Security is solving

There are three components of Kafka Security:

### a. Encryption of data in-flight using SSL / TLS

It keeps data encrypted between our producers and Kafka as well as our consumers and Kafka. However, we can say, it is a very common pattern everyone uses when going on the web.

### b. Authentication using SSL or SASL

To authenticate to our Kafka Cluster, it allows our producers and our consumers, which verifies their identity. It is the very secure way to enable our clients to endorse an identity. That helps well in the authorization.

[**Apache Kafka Use cases | Kafka Applications**](https://data-flair.training/blogs/kafka-use-cases-and-applications/)

### c. Authorization using ACLs

In order to determine whether or not a particular client would be authorized to write or read to some topic, our Kafka brokers can run our clients against access control lists (ACL).

## 5. Encryption (SSL)

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Example-1.png)

Kafka Security- SSL Encryption

Since our packets, while being routed to Kafka cluster, travel network and also hop from machines to machines, this solves the problem of the man in the middle (MITM) attack. Any of these routers could read the content of the data if our data is PLAINTEXT.

Our data is encrypted and securely transmitted over the network with enabled encryption and carefully setup SSL certificates. Only the first and the final machine possess the ability to decrypt the packet being sent, with SSL.

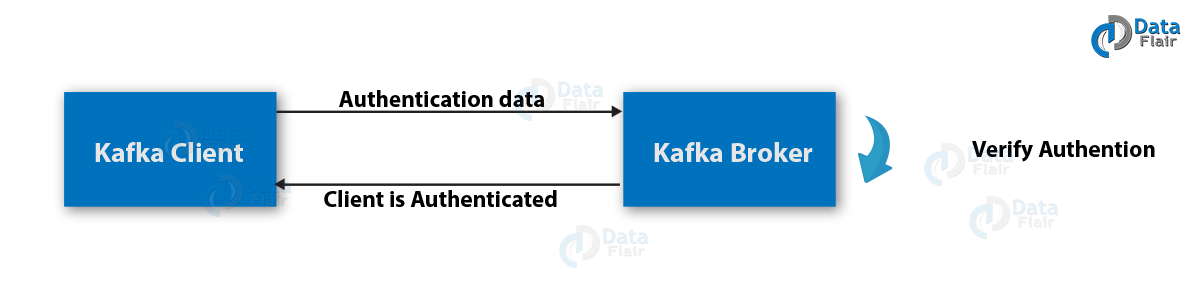
However this encryption comes at a cost, that means in order to encrypt and decrypt packets CPU is now leveraged for both the Kafka Clients and the Kafka Brokers. Although, SSL Security comes at the negligible cost of performance.

**Note:** The encryption is only in-flight and the data still sits un-encrypted on our broker’s disk.

[**Read Kafka serialization and deserialization**](https://data-flair.training/blogs/kafka-serialization-and-deserialization/)

## 6. Kafka Authentication (SSL & SASL)

Basically, authentication of Kafka clients to our brokers is possible in two ways. SSL and SASL

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Example-2.png)

Kafka Security- SSL & SASL Authentication

### a. SSL Authentication in Kafka

It is leveraging a capability from SSL, what we also call two ways authentication. Basically, it issues a certificate to our clients, signed by a certificate authority that allows our Kafka brokers to verify the identity of the clients.

However, it is the most common setup, especially when we are leveraging a managed Kafka clusters from a provider like Heroku, Confluent Cloud or CloudKarafka.

[**Apache Kafka Architecture and its fundamental concepts**](https://data-flair.training/blogs/kafka-architecture/)

### **b.**SASL Authentication in Kafka

SASL refers to Simple Authorization Service Layer. The basic concept here is that the authentication mechanism and Kafka protocol are separate from each other. It is very popular with [**Big Data**](https://data-flair.training/blogs/what-is-big-data/) systems as well as [**Hadoop setup**](https://data-flair.training/blogs/configure-hadoop-cdh5-on-ubuntu/).￼

Kafka supports the following shapes and **forms of SASL**:

#### **i.**SASL PLAINTEXT

SASL PLAINTEXT is a classic username/password combination. However, make sure, we need to store these usernames and passwords on the Kafka brokers in advance because each change needs to trigger a rolling restart. However, it’s less recommended security. Also, make sure to enable SSL encryption while using SASL/PLAINTEXT, hence that credentials aren’t sent as PLAINTEXT on the network.

#### **ii.**SASL SCRAM

It is very secure combination alongside a challenge. Basically, password and Zookeeperhashes are stored in Zookeeper here, hence that permits us to scale security even without rebooting brokers. Make sure to enable SSL encryption, while using SASL/SCRAM, hence that credentials aren’t sent as PLAINTEXT on the network.

#### **iii.**SASL GSSAPI (Kerberos)

It is also one of a very secure way of providing authentication. Because it works on the basis of Kerberos ticket mechanism. The most common implementation of Kerberos is Microsoft Active Directory. Since it allows the companies to manage security from within their Kerberos Server, hence we can say SASL/GSSAPI is a great choice for big enterprises. Also, communications which are encrypted to SSL encryption is optional with SASL/GSSAPI. However, setting up Kafka with Kerberos is the most difficult option, but worth it in the end.

[**Learn Apache Kafka-Load Test with JMeter**](https://data-flair.training/blogs/kafka-load-testing/)

* (WIP) SASL Extension (KIP-86 in progress)

To make it easier to configure new or custom SASL mechanisms that are not implemented in Kafka, we use it.

* (WIP) SASL OAUTHBEARER (KIP-255 in progress)

This will allow us to leverage OAUTH2 token for authentication.

However, to perform it in easier way use SASL/SCRAM or SASL/GSSAPI (Kerberos) for authentication layer.

## 7. Kafka Authorization (ACL)

Kafka needs to be able to decide what they can and cannot do, as soon as our Kafka clients are authenticated. This is where Authorization comes in, controlled by Access Control Lists (ACL).

Since ACL can help us prevent disasters, they are very helpful. Let’s understand it with an example, we have a topic that needs to be writeable from only a subset of clients or hosts. Also, we want to prevent our average user from writing anything to these topics, thus it prevents any data corruption or deserialization errors. ACLs are also great if we have some sensitive data and we need to prove to regulators that only certain applications or users can access that data.

we can use the kafka-acls command, to adds ACLs. It also even has some facilities and shortcuts to add producers or consumers.

1. kafka-acl --topic test --producer --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal User:alice

The result being:

**Adding ACLs for resource `Topic:test`:**

**User:alice has Allow permission for operations: Describe from hosts: \***

**User:alice has Allow permission for operations: Write from hosts: \***

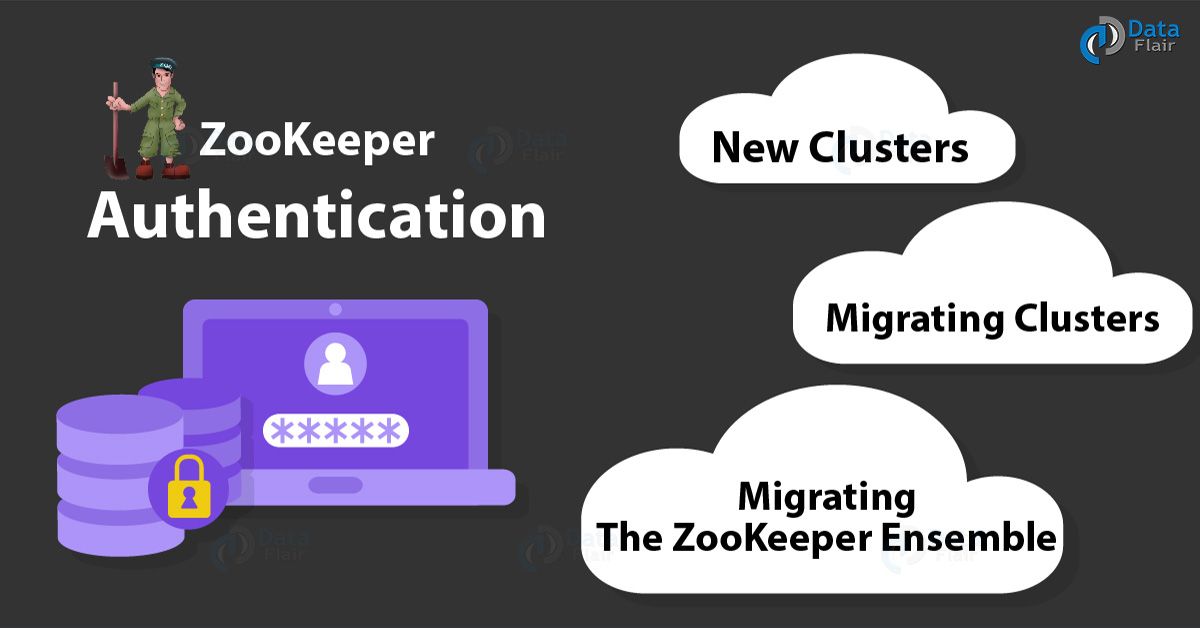
**Adding ACLs for resource `Cluster:kafka-cluster`:**

**User:alice has Allow permission for operations: Create from hosts: \***

**Note:** Store ACL in Zookeeper by using the default SimpleAclAuthorizer, only. Also, ensure only Kafka brokers may write to Zookeeper (zookeeper.set.acl=true). Else, any user could come in and edit ACLs, thus defeating the point of security.

[**Advantages and Disadvantages of Kafka**](https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/)

## 8. ZooKeeper Authentication

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/ZooKeeper-Authentication-01.jpg)

Zookeeper Authentication

### a. New Clusters

There are two necessary steps in order to enable ZooKeeper authentication on brokers:

1. At first, set the appropriate system property just after creating a JAAS login file and to point to it.
2. Set the configuration property zookeeper.set.acl in each broker to true.

Basically, the ZooKeeper’s metadata for the Kafka cluster is world-readable, but only brokers can modify it because inappropriate manipulation of that data can cause cluster disruption. Also, we recommend limiting the access to ZooKeeper via network segmentation.

[**Read Role of Apache ZooKeeper in Kafka – Monitoring & Configuration**](https://data-flair.training/blogs/zookeeper-in-kafka/)

### **b.**Migrating Clusters

We need to execute the several steps to enable ZooKeeper authentication with minimal disruption to our operations, if we are running a version of Kafka that does not support security or simply with security disabled, and if we want to make the cluster secure:

1. At first, perform a rolling restart setting the JAAS login file, which enables brokers to authenticate. At the end of the rolling restart, brokers are able to manipulate znodes with strict ACLs, but they will not create znodes with those ACLs
2. Now, do it the second time, and make sure this time set the configuration parameter zookeeper.set.acl to true, hence as a result, that can enable the use of secure ACLs at the time of creating znodes.
3. Moreover, execute the ZkSecurityMigrator tool. So, in order to execute the tool, use this script: **./bin/zookeeper-security-migration.sh** with zookeeper.acl set to secure. This tool traverses the corresponding sub-trees changing the ACLs of the znodes.

with these following steps we can turn off authentication in a secure cluster:

1. Perform a rolling restart of brokers setting the JAAS login file, which enables brokers to authenticate, but setting zookeeper.set.acl to false. However, brokers stop creating znodes with secure ACLs, at the end of the rolling restart. Although they are still able to authenticate and manipulate all znodes.
2. Also, execute the tool ZkSecurityMigrator tool with this script ./bin/zookeeper-security-migration.sh with zookeeper.acl set to unsecure. It traverses the corresponding sub-trees changing the ACLs of the znodes.
3. Further, do perform it a second time as well. Make sure this time omitting the system property which sets the JAAS login file.

[**Let’s explore Kafka-docker: Steps to run Apache Kafka using Docker**](https://data-flair.training/blogs/kafka-docker/)

**Example** **of how to run the migration tool**:

For Example,

**./bin/zookeeper-security-migration.sh –zookeeper.acl=secure –zookeeper.connect=localhost:2181**

Run this to see the full list of parameters:

**./bin/zookeeper-security-migration.sh –help**

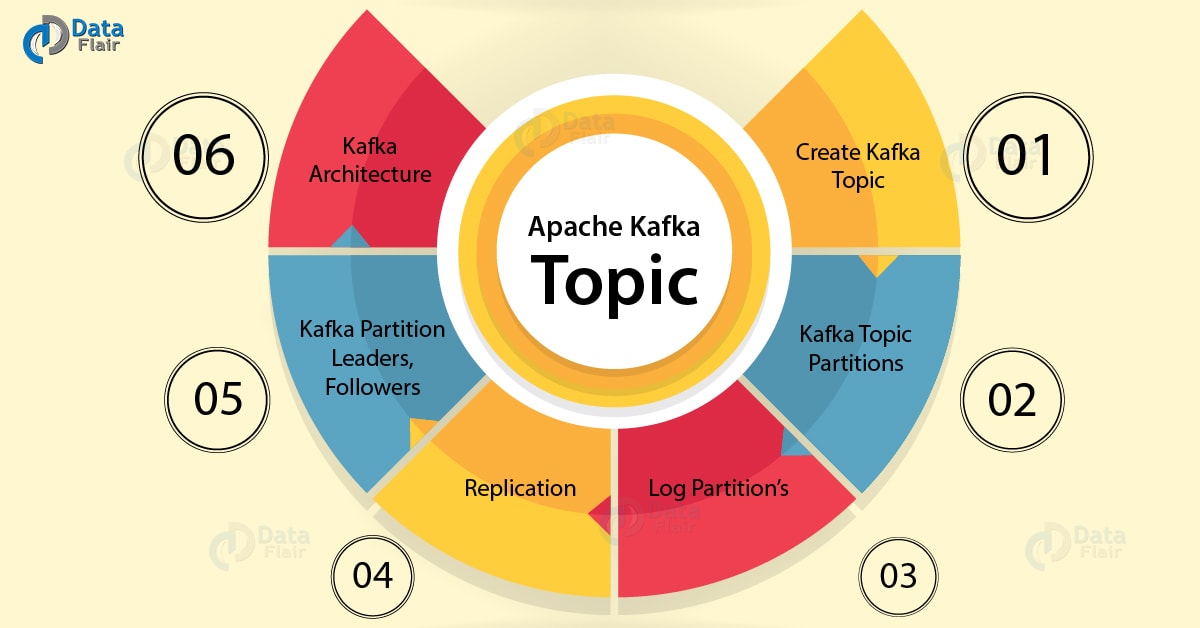
### c.Migrating the ZooKeeper Ensemble

We need to enable authentication on the ZooKeeper ensemble. Hence, we need to perform a rolling restart of the server and set a few properties, to do it.

Kafka Topic

In this [**Kafka**](https://data-flair.training/blogs/apache-kafka-tutorial/) article, we will learn the whole concept of a Kafka Topic along with Kafka Architecture. Where [**architecture in Kafka**](https://data-flair.training/blogs/kafka-architecture/) includes replication, Failover as well as Parallel Processing. In addition, we will also see the way to create a Kafka topic and example of Apache Kafka Topic to understand Kafka well. Moreover, we will see Kafka partitioning and Kafka log partitioning.

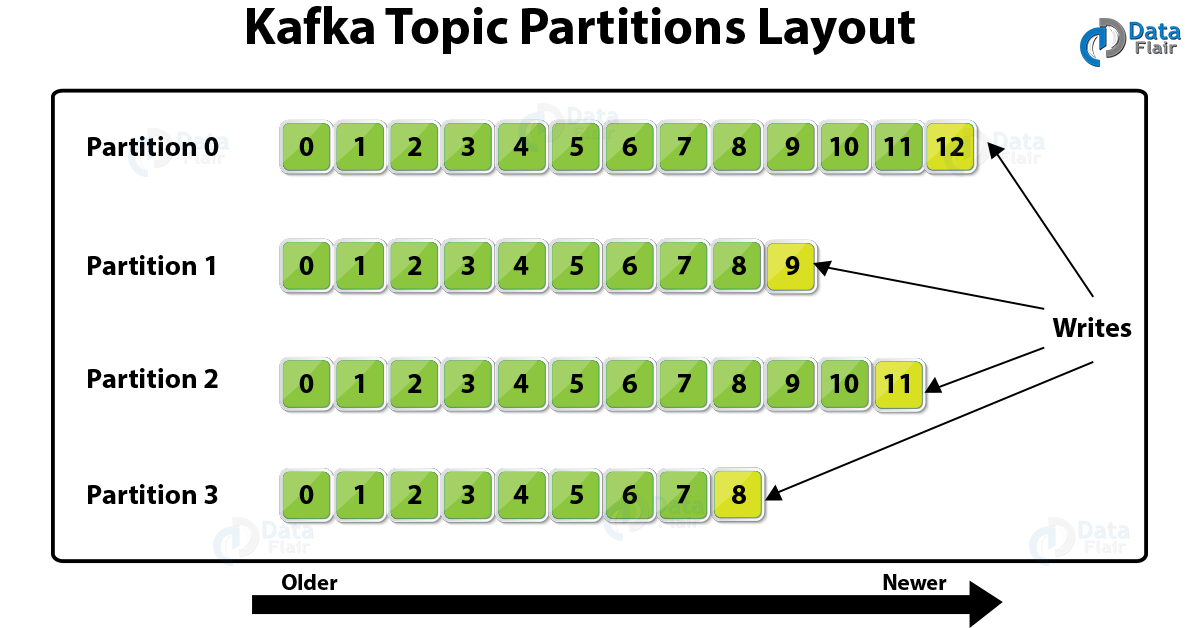
So, let’s begin with the Kafka Topic.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Apache-Kafka-Topic-01.jpg)

Apache Kafka Topic – Architecture & Partitions

2. What is Kafka Topic?

Simply put, a named stream of records is what we call Kafka Topic. Basically, in logs Kafka stores topics. However, a topic log in Apache Kafka is broken up into several partitions. And, further, Kafka spreads those log’s partitions across multiple servers or disks. In other words, we can say a topic in Kafka is a category, stream name or a feed.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Kafka-Topic-Partitions-Layout.png)

What is the Topic in Kafka

In addition, we can say Topics in Apache Kafka are inherently published as well as subscribe style messaging. Moreover, there can be zero or many subscribers called **Kafka Consumer Groups** in a Kafka Topic. Basically, these Topics in Kafka are broken up into partitions for speed, scalability, as well as size.

[**Learn More about Kafka Pub-Sub Messaging System**](https://data-flair.training/blogs/kafka-workflow/)

3. How to Create a Kafka Topic

At very first, run kafka-topics.sh and specify the topic name, replication factor, and other attributes, to create a topic in Kafka:

1. /bin/kafka-topics.sh --create \
2. --zookeeper <hostname>:<port> \
3. --topic <topic-name> \
4. --partitions <number-of-partitions> \
5. --replication-factor <number-of-replicating-servers>

Now, with one partition and one replica, below example creates a topic named “test1”:

[**Read Kafka Monitoring**](https://data-flair.training/blogs/kafka-monitoring/)

1. bin/kafka-topics.sh --create \
2. --zookeeper localhost:2181 \
3. --replication-factor 1 \
4. --partitions 1 \
5. --topic text

Further, run the list topic command, to view the topic:

1. > bin/kafka-topics.sh --list --zookeeper localhost:2181
2. test1

Make sure, when applications attempt to produce, consume, or fetch metadata for a nonexistent topic, the auto.create.topics.enable property, when set to true, automatically creates topics.

4. Kafka Topic Partitions

Further, Kafka breaks topic logs up into several partitions. So, usually by record key if the key is present and round-robin, a record is stored on a partition while the key is missing (default behavior). By default, the key which helps to determines that which partition a [**Kafka Producer**](https://data-flair.training/blogs/kafka-producer/) sends the record is the Record Key.

Basically, to scale a topic across many servers for producer writes, Kafka uses partitions. Also, in order to facilitate parallel consumers, Kafka uses partitions. Moreover, while it comes to failover, Kafka can replicate partitions to multiple [**Kafka Brokers**](https://data-flair.training/blogs/kafka-broker/).

5. Kafka Topic Log Partition’s Ordering and Cardinality

Well, we can say, only in a single partition, Kafka does maintain record order. As a partition is also an ordered, immutable record sequence. And, by using the partition as a structured commit log, Kafka continually appended to partitions. In partitions, all records are assigned one sequential id number which we further call an offset. That offset further identifies each record location within the partition.

[**Have a look at Kafka vs RabbitMQ**](https://data-flair.training/blogs/kafka-vs-rabbitmq/)

In addition, in order to scale beyond a size that will fit on a single server, Topic partitions permits to Kafka log. As topics can span many partitions hosted on many servers but Topic partitions must fit on servers which host it. Moreover, topic partitions in Apache Kafka are a unit of parallelism. That says, at a time, a partition can only be worked on by one [**Kafka Consumer**](https://data-flair.training/blogs/kafka-consumer/) in a consumer group. Basically, a Consumer in Kafka can only run in their own process or their own thread. Although, Kafka spreads partitions across the remaining consumer in the same consumer group, if a consumer stops.

6. Kafka Topic Partition Replication

For the purpose of fault tolerance, Kafka can perform replication of partitions across a configurable number of Kafka servers. Basically, there is a leader server and zero or more follower servers in each partition. Also, for a partition, leaders are those who handle all read and write requests.

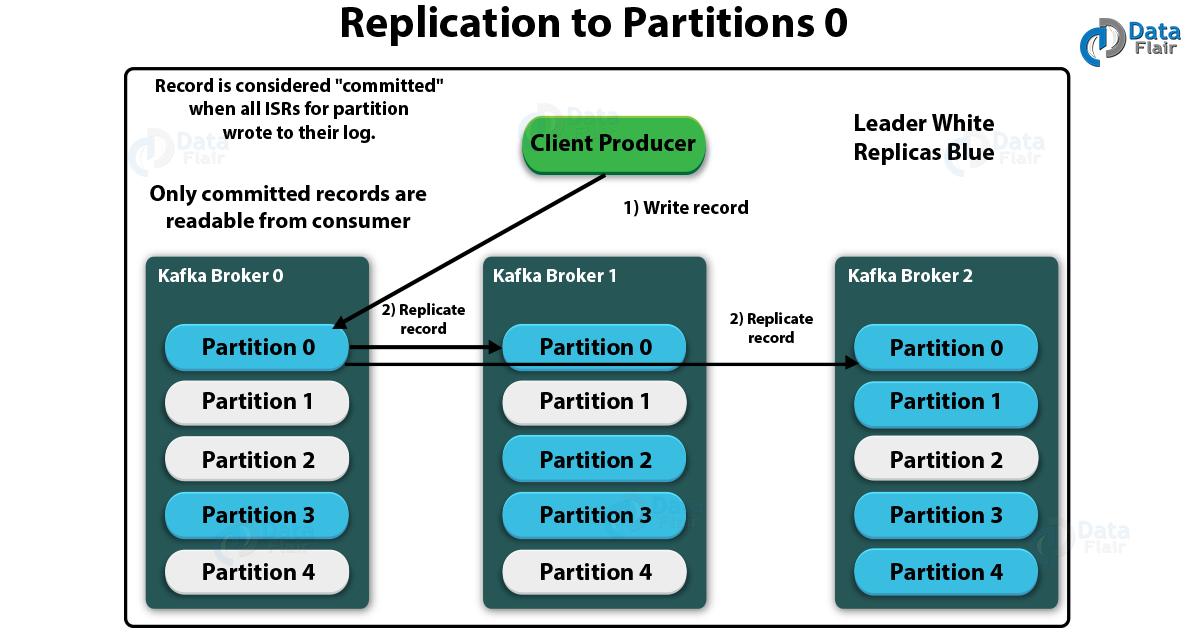
However, if the leader dies, the followers replicate leaders and take over. Additionally, for parallel consumer handling within a group, Kafka uses also uses partitions.

[**Let’s discuss Kafka Schema**](https://data-flair.training/blogs/kafka-schema-registry/)

7. Replication: Kafka Partition Leaders, Followers, and ISRs.

However, by using [**ZooKeeper**](https://data-flair.training/blogs/zookeeper-tutorial/), Kafka chooses one broker’s partition’s replicas as the leader. Also, we can say, for the partition, the broker which has the partition leader handles all reads and writes of records. Moreover, to the leader partition to followers (node/partition pair), Kafka replicates writes. On defining the term ISR, a follower which is in-sync is what we call an ISR (in-sync replica). Although, Kafka chooses a new *ISR* as the new leader if a partition leader fails.

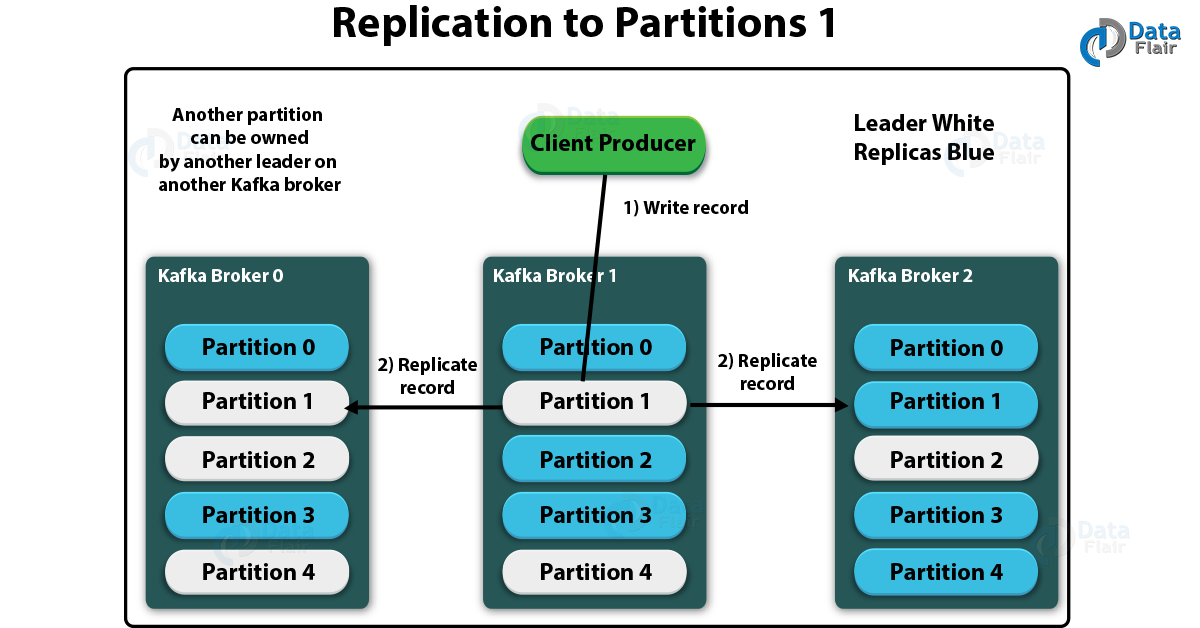
**8. Kafka Architecture: Kafka Replication – Replicating to Partition 0**

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Replication-to-Partitions-0.png)

Kafka Architecture: Kafka Replication – Replicating to Partition 0

[**Let’s discuss the role of ZooKeeper in Kafka**](https://data-flair.training/blogs/zookeeper-in-kafka/)

Although, when all ISRs for partition wrote to their log, the record is considered “committed”. However, we can only read the committed records from the consumer.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Replication-to-Partitions-1.png)

Kafka Topic – Replication to Partition 1

So, this was all about Kafka Topic. Hope you like our explanation.

**Kafka Connect:**

Kafka Connect is part of Apache Kafka® and is a powerful framework for building streaming pipelines between Kafka and other technologies. It can be used for streaming data into Kafka from numerous places including databases, message queues and flat files, as well as streaming data from Kafka out to targets such as document stores, NoSQL, databases, object storage and so on.

Check it Dead letter queue:

Here, we’ll look at several common patterns for handling problems and examine how they can be implemented.

## “Fail fast”

Sometimes you may want to stop processing as soon as an error occurs. Perhaps encountering bad data is a symptom of problems upstream that must be resolved, and there’s no point in continuing to try processing other messages.

To fix the pipeline, we need to resolve the issue with the message on the source topic. Kafka Connect will not simply “skip” the bad message unless we tell it to.

This is the default behavior of Kafka Connect

errors.tolerance = none

## Silently ignore bad messages

## No logs will be printed.

## Route messages to a dead letter queue

Kafka Connect can be configured to send messages that it cannot process (such as a deserialization error as seen in “fail fast” above) to a dead letter queue, which is a separate Kafka topic. Valid messages are processed as normal, and the pipeline keeps on running. Invalid messages can then be inspected from the *dead letter queue*, and ignored or fixed and reprocessed as required.

**Avoiding message losses, duplication and lost / multiple processing in Kafka**

Kafka misbehaving? So what? Increase RAM, add new broker machine. Nope, it won’t work that way!

**Various failure scenarios**

There are many scenarios because such things can happen and each scenario needs to be handled carefully. we need to configure and implement it carefully

First, there are the producer side scenarios. It deals with mainly two things:

* Ensuring the message does indeed gets logged to Kafka
* Ensuring the message is not getting logged multiple times to Kafka

Second, there are consumer side scenarios. Again, it deals with mainly two things:

* Ensuring the message does gets consumed and processed (yeah, those two things are quite different and needs extra care during implementation)
* Ensuring the message gets consumed and processed exactly once and not multiple times

**Producer side – Why a message won't get logged to the Kafka**

So, first thing, why a message won't get logged to the Kafka, even when we do make the required API call? Following can be some of reasons:

* Leader of the partition is down
* All brokers are down

Message simply gets destroyed in the transit from producer to leader

**Producer side – Ensuring a message gets logged to the Kafka – producer acks property**

We simply need to configure `acks` property in producer to `all`. Meaning of different values of acks property is as follows:

acks=0 : The default is `acks=0`, which means producer will send the message and forget, it won't wait for any acknowledgement from the leader of the partition to which the message is to be produced.

acks=1 : Means leader sends acknowledgement to the producer after it writes the message to its own replica without waiting for followers to replicate the messages. So, if the leader fails before any follower replicates the message, the record is lost.

acks=all : With this, once the leader receives acknowledgements from in-sync replicas, telling they have replicated the message, it will send back the acknowledgement to the producer. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive.

**Producer side – why a message may get logged multiple times**

This happens when producer fails to receive acknowledgement before resending the message. So, the message is already persisted in leader replica, but producer did not receive any acknowledgement from leader or the acknowledgement is received after the producer resend timer is expired and hence producer resends the message causing duplication. Its easy to avoid this kind of message duplication. We just have to set producer property `enable.idempotence=true`.

**Producer side – what if message fails to get logged or any exception occurs**

As stated above, a message may get failed to log after max re attempts or some other exception may occur. In this scenario, we may opt to persist the message to some other data store (which is what we opted for). This can be done by specifying a callback in send(), which gets invoked in such error scenarios.

**Consumer side – committing message offset**

When consumer consumes a message, it commits its offset to Kafka. Committing the message offset makes next message to be returned when poll() is called again. If consumer does not commit the message, every call to poll() will always return the same message. By default, commit occurs automatically periodically in the background. If we want to manually commit, then we should set enable.auto.commit property of consumer to false and then call Consumer.commit().

**Consumer side – what leaves messages unprocessed?**

Consider consumer consumes message, commits its offset and then starts processing it. But while processing, some failure occurs and processing is not completed. Now, next call to poll() will return next message (as earlier message offset is committed) and the processing for earlier message will never be completed.

**Why Kafka is so fast:**

Kafka’s speed is like the Flash of the data world. It’s all thanks to its unique design and architecture:

**1. Zero Copy**

In traditional data transfer methods, data is moved from the disk to the kernel space and then to the user space. From there, it’s moved back to the kernel space and finally to the disk. This back-and-forth movement of data is quite CPU-intensive and slows down the data transfer process.

Kafka, however, uses a technique called “zero copy”. This technique allows data to be transferred directly from the file system cache (kernel space) to the network buffer without being copied to the user space. This reduces the CPU overhead and makes data transfer significantly faster.

**2. Batch Processing**

Kafka groups multiple messages together into batches. This batch-processing approach reduces the number of network calls and disk I/O operations, thereby increasing the throughput.

Imagine you’re sending letters through the post. Instead of sending each letter individually, you group several letters together and send them in one large envelope. This is more efficient and faster than sending each letter separately. That’s essentially what Kafka does with batch processing.

**3. Sequential I/O**

Disk I/O operations can be a major bottleneck in data transfer. Random disk I/O operations, where data is written to or read from different locations on the disk, are particularly slow.

Kafka writes data to disk sequentially, meaning data is written to or read from contiguous locations on the disk. Sequential disk I/O operations are much faster than random disk I/O operations because they reduce the amount of time the disk spends seeking the correct place to read or write data.