# Kafka

Introduction

Developed by linked-in

1. **Which version of Kafka does DMP project use?**

0.9.0.0

Latest version is 1.0

1. **What is messaging system?**

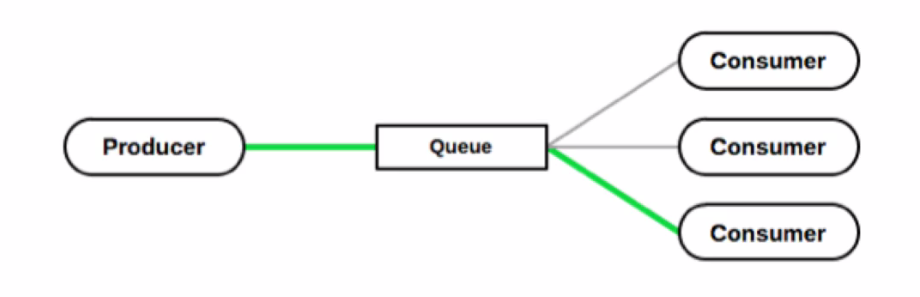
Messaging system is used to transfer the data from application to another.

In such case applications can focus on data no need to worry about how to transfer data

1. **What are types of messaging system?**

**Point to point**: Messages will be written to queue and it can be consumed by multiple consumers but out of them only one consumer can consume a message. Once one consumer reads the message messages will be disappeared from the queue

**Ex**: Order processing system



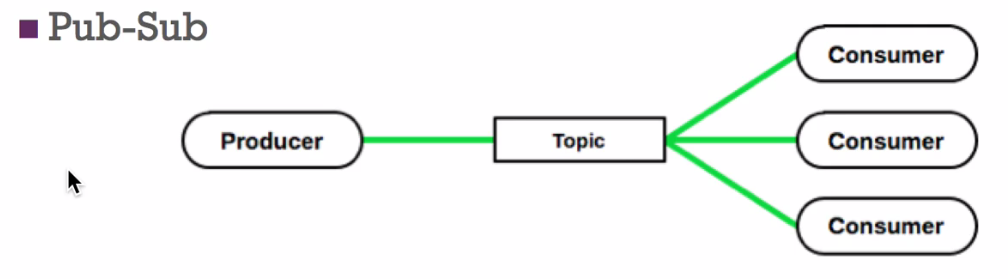
Only the consumer with green line can consume message

**Publish and subscribe:**

Messages will be published to topic and each messages can be consumed by all consumers.

One consumer can be subscribed to multiple topics

Ex: Dish tv where dish tv publishes many channels and consumers can see the channels to which they have subscribed



All consumers are able to consume messages

1. **What is Kafka?**

Distributed messaging system designed to be fast, scalable and durable.

1. **In which language kafka is written?**

Kafka is written using **Scala** and **Java**

1. **Who developed kafka?**

Linked in

1. **What are the advantages of Kafka?**

**Reliability:** It is reliable because it is distributed, partitioned, and supports replication as a result Kafka is reliable

**Performance**: High throughput for both publish and subscribe. It gives stable performance even TBs of data stored.

**Scalability:** can be scaled at any point of time without downtime.

**Durability:** It uses distributed commit logs which means it persists messages to disk as fast as possible hence it is durable.

1. **Where Kafka can be used?**

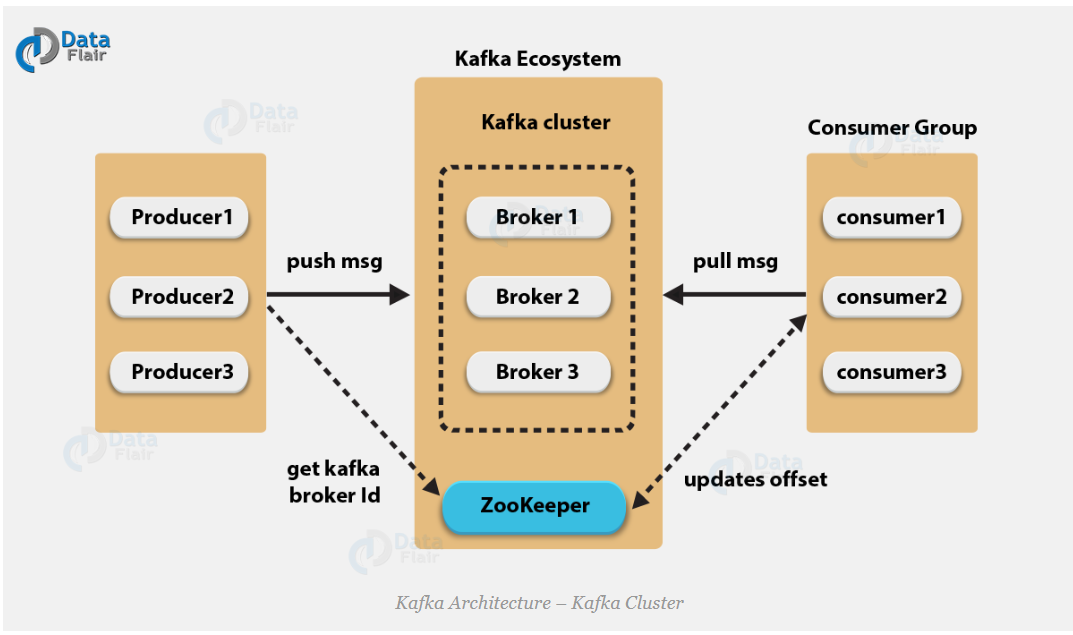
**Streaming:** Can be used in real time streaming application like spark, storm.

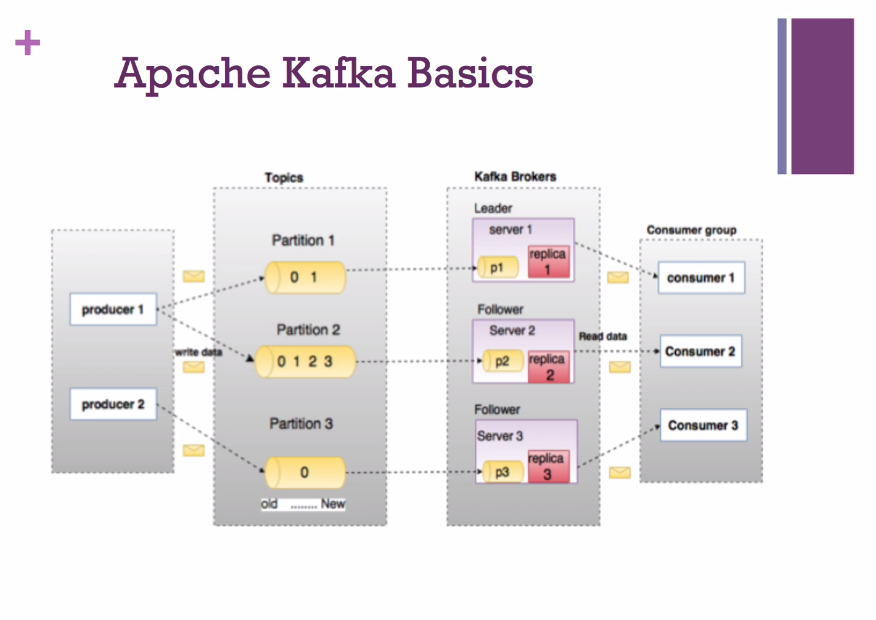
**Log Aggregation solution**:

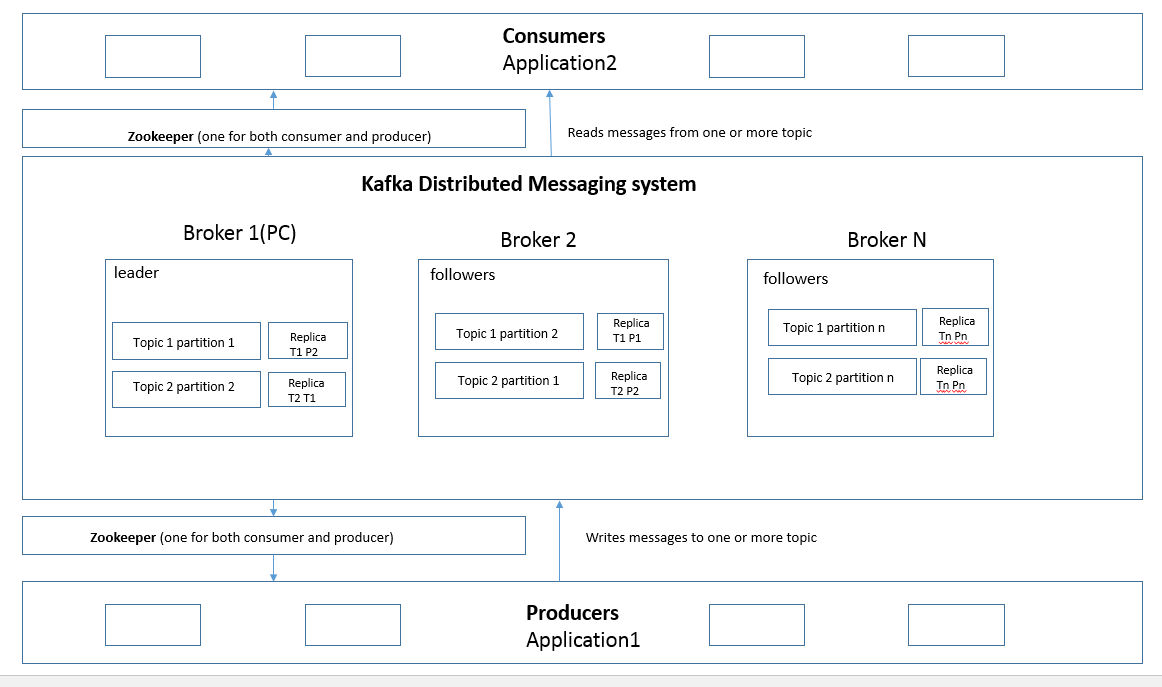
**Metrics**

1. **Explain Architecture of Kafka? IMP**

**Important Terminologies of Kafka**







1. **Kafka architecture- Cluster**
   1. **Kafka Producers: IMP**

Application/process which publish data to a topic of their choice.

Sends the message as fast as broker can handle.

It does not wait for acknowledgements from the broker.

Producers can decide to which partition the data should be written.

This can be achieved using round robin fashion or by any other semantic partition function.

When producers sends message to Kafka brokers, brokers simply appends message to last segment file.

Actually messages will be added to a partition.

* 1. **Kafka Brokers IMP**

Brokers: Set of servers where the published messages are stored.

Multiple brokers are used in Kafka for load balancing.

Each kafka brokers are stateless that is the reason kafka uses zookeeper.

Each Kafka broker will have capability to handle 1000s of writes and reads per second.

Without impacting the performance each Kafka broker can handle TB ( Tera Bytes) of data.

If there are n partitions in topic and n number of brokers each broker will have 1 partition.

If there are n number of partitions in topic and more than n brokers. In this case Kafka chooses brokers randomly and assigns n partitions to n brokers.

If there are more than n number of partitions in a topic and n number of brokers. In such a case few brokers will have multiple partitions.

* 1. **Kafka Consumers** **IMP**

Application or process, that subscribes to one or more topics and pulls data. It is not like data is produced and consumer received it. We consumer must pull the data. So, it is like producer push the data and consumer pull the data. Push-pull.

Uses partition offset to maintain how many messages have consumed because Kafka broker is stateless.

Once consumer acknowledges a particular offset it assures that it has read all previous data.

Consumer sends an asynchronous pull request to broker so that broker keep buffer of bytes ready to consume.

By sending an offset value consumers can go to previous message or any point in the partition

* 1. **Kafka zookeeper IMP**

Zookeeper is used to manage and coordinate Kafka brokers.

Zookeeper is used to notify producers and consumers about addition of new broker or failure of broker in Kafka system.

As soon as zookeeper send notification to producers and consumers regarding presence or failure of the broker then producers and consumers take decision and start coordinating their task with some other broker

1. **Kafka architecture- Fundamental concept**
   1. **Kafka topic IMP**

Logical channel to which producer produce the message and consumer consumes the message.

Stream of particular type of data in kafka.

In Kafka cluster topic is identified by name and should be unique.

There can be any number of Kafka topics in cluster there is no limitation.

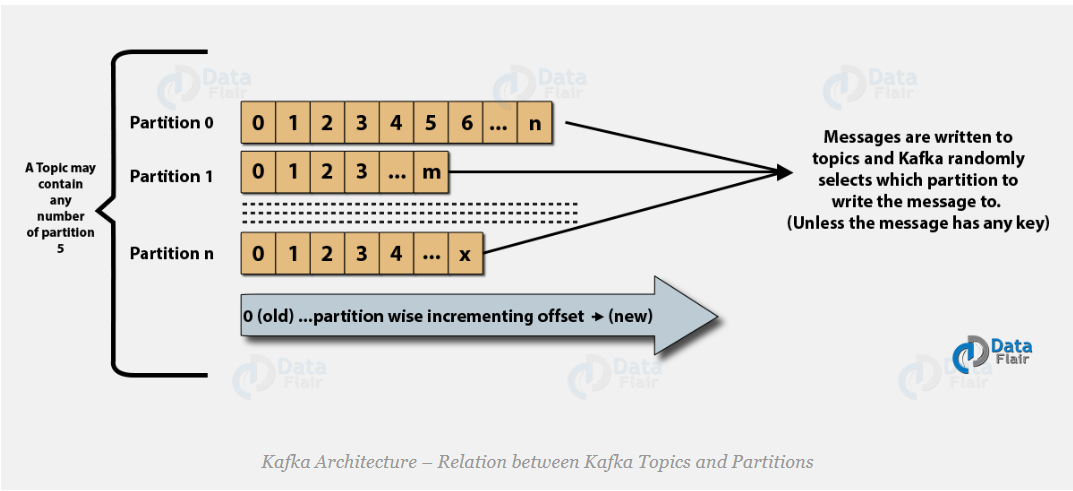
Once any message is published to kafka it cannot be changed or updated.

* 1. **Kafka partition IMP**

Each Kafka topic is divided in to partitions and partitions are replicated across multiple brokers.

By default there will be at least one partition.

Partitions will contain messages in an immutable order of sequence.

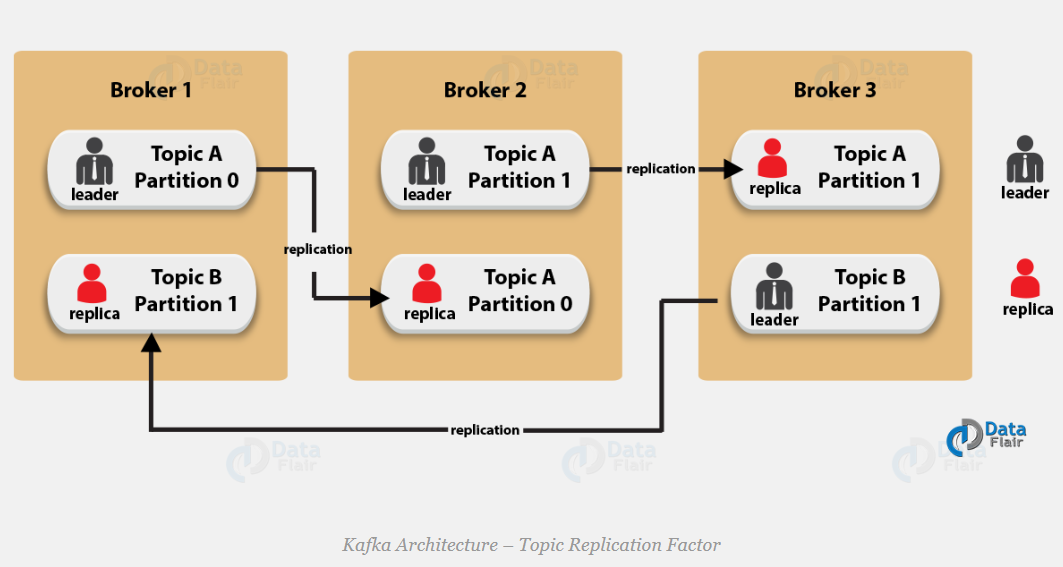


* 1. **Kafka topic replication factor. IMP**

Replication factor is used to achieve fault tolerance in Kafka. If replication factor configuration is configured as 2 then there will be 2 replicas of each partition

Replication takes place in the partition level only.

For a given partition, only one broker can be a leader at a time. Meanwhile, other brokers will have in-sync replica; what we call ISR.



* 1. **Kafka Consumer group. IMP**

We can have group of consumers consume the same topic.

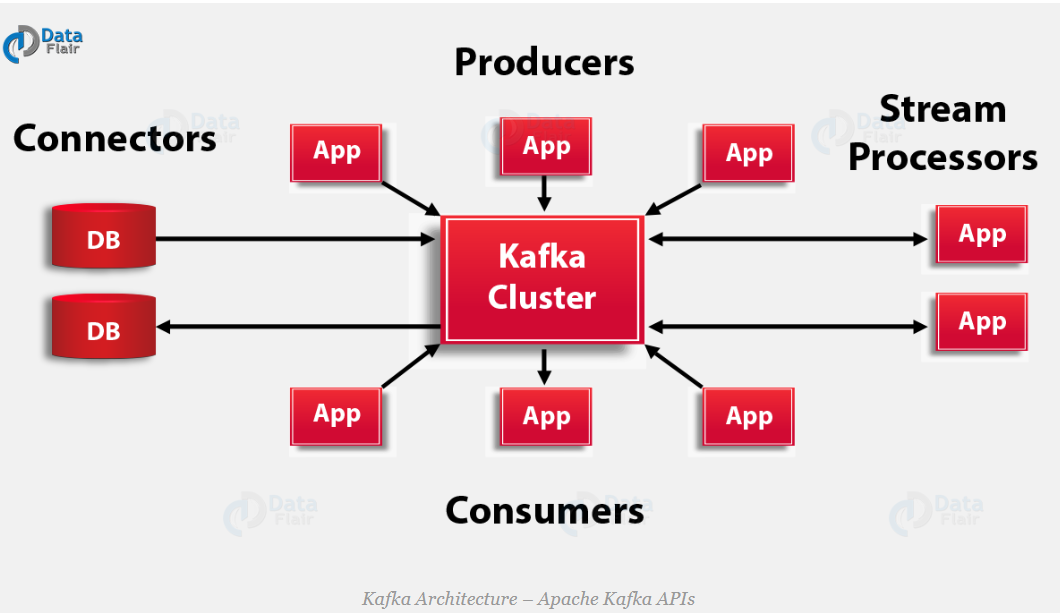
One consumer group is identified by one unique **group-id.**

Exactly one consumer instance from consumer group reads one partition

There can be multiple consumer groups. So one consumer instance from each group reads the message.

If the number of consumers exceeds number of partitions, then there will be inactive consumers. Let’s understand it with an example if there are 8 consumers and 6 partitions in a single consumer group, that means there will be 2 inactive consumers.

1. **Kafka architecture-Apache Kafka API**



1. **Producer API**

Using producer API application can publish stream of message to one or more topics

1. **Consumer API**

Using consumer API application can subscribe to one or more topics and process the stream of data produced.

1. **Stream API**

Using streaming API we can consume input stream from one or more topic and producing output streams to one or more topics. Stream API effectively transforms input stream to output stream.

1. **Connector API**

Using connector API we can connect to data bases say relational databases and we can capture every changes that happens to table.

## Broker

### What will happen if a broker fails? IMP

Say we have 3 brokers in that case partitions are divided and among all brokers and each partition will have leaders. Leader partitions are the one who actively involved in reading and writing of data and each replicas will have followers.

Once one of the broker goes down zookeeper will immediately elect one of the follower from other broker as leader without affecting operation.

<https://dzone.com/articles/understanding-kafka-failover>

1. **When a broker is down how producer will know that broker/leader is down and how producer write data to another leader? IMP**

Every broker in the cluster has information about list of topics, their partitions and their leaders which is kept up to date by zookeeper. Zookeeper update this information whenever a new leader is elected or when the number of partition changes.

So when producer makes a call to one of the broker, broker returns all this information to producer, producer cache this data and uses it to connect to leader. Producer uses this cached information each time it writes data.

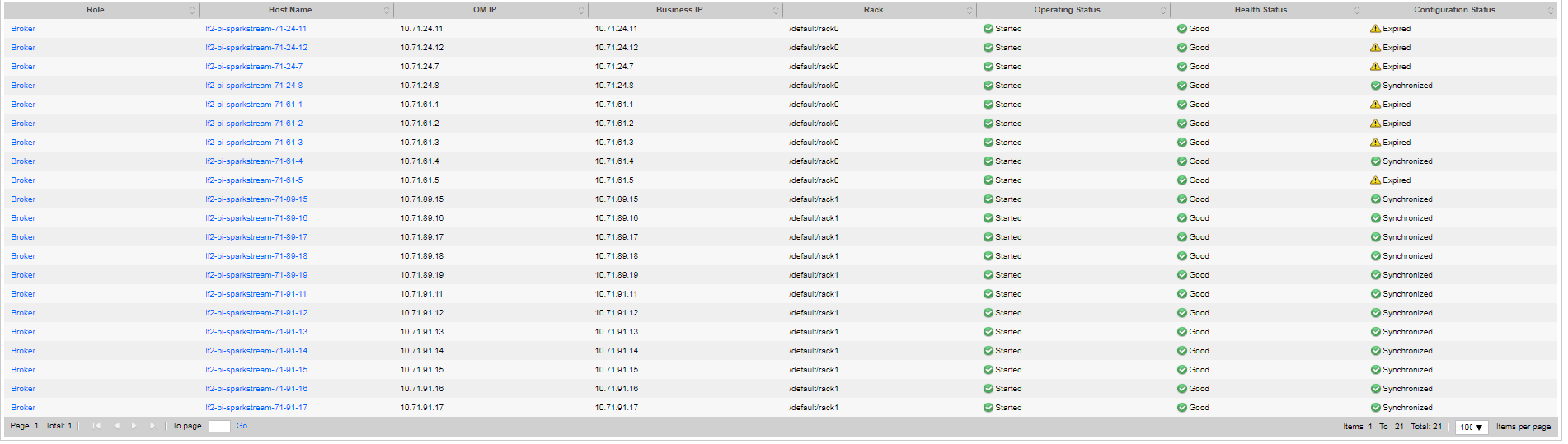
If one of the broker/leader is crashed producer try to connect to the leader and it fails, In that case producer tries to connect to other brokers to check if there is new leader elected to this partition. If exists, then data is produced to that leader. If there is no leader exists, then producer tries to connect to leader to which it has crashed and after reaching maximum number of retries it will throw an exception.

1. **How consumer behaves when broker is down? IMP**

1. **How many brokers were there in DMP Project production?**

21. It is shared among multiple components.

*Note: These brokers are shared among many applications in shared streaming cluster*



1. **What is Kafka Cluster:**

Kafka having more than one broker is called as Kafka cluster.

Kafka cluster can be scaled without any downtime.

Using cluster we can manage persistence (storing) and replicating data.

## Partition

### What is Partition in Kafka?

Each Kafka topic is divided into partitions and partitions are replicated across multiple brokers.

By default, there will be at least one partition.

Partitions will contain messages in an immutable order of sequence.

### How parallelism is archived in Kafka?

Parallelism is achieved using partitions in Kafka.

1. **When I send a message to kafka topic where does in reside? IMP**

Logically the message will side in topic’s partition.

Physically data is stored in disk.

1. **How it is decided that data is to be written to which topic partition? IMP**

Actually in Kafka data is key-value pair.

Based on the key Partition to which data is to be written will be decide.

Using Key, we can distribute the data intelligently and efficiently.

If there is no key then Kafka decides partitions randomly and write the data.

In one partition messages are stored in sequenced fashion.

There can be any number of partitions there are no limitations.

1. **By default, which is the algorithm used by Kafka for partitioning (Data distribution)?** **IMP**

Consistent **hashing** algorithm

1. **How data is stored in case key is null?** **IMP**

In case key is null the data is stored randomly on any of the partition

1. **How Custom Partitioner can be created? IMP**

Custom partitioner can be created

* 1. by implementing **Partitioner** interface

*org.apache.kafka.clients.producer.Partitioner*

* 1. Once custom **Partitioner** is implemented we need to configure so that kafka producer will use it.

Sample Patitioner

https://simplydistributed.wordpress.com/2016/12/13/kafka-partitioning/

(fully qualified class name)RandomKafkaPartitioner

1. **Is it possible to have number of replication factors more than number of available brokers?**

No.

1. **How many partitions RTS input topic has?**

RTS input topic has 200 partitions currently

1. **Is the message ordering is maintained in Kafka topic?**

Message ordering is maintained with in the partition only not in the entire kafka topic.

1. **Explain the concept of producing message synchronously and asynchronously?**

We can choose in our code whether we need to produce message synchronously or asynchronously.

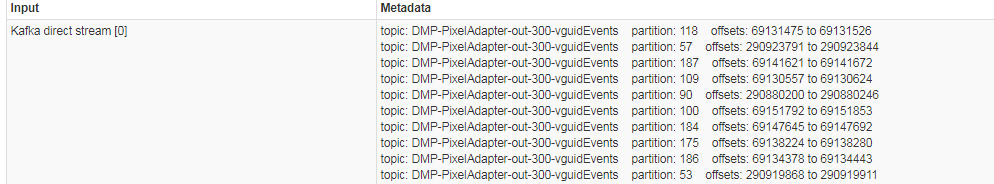
The configuration producer.type can be used. By default is **sync** can use **async**

When configuration is async producer just publish the message and forget it. It mean the data will be stored in buffer and it will be written once batch is full.

In case of sync each message will be written immediately without batching?

1. **What is Partition offset:**

Partition offset is used to uniquely identify messages in the partition



1. **How do you define a Partitioning Key?**

Partition key is used to decide the partition to which the data is to be written.

By default**, a hashing-based Partitioner** is used to determine the partition ID given the key.

Alternatively, users can also use customized Partitions.

1. **What is Replication of partition?**

Replication is simply a copy of data to avoid data loss. Using Replication only high availability in the Kafka is achieved.

1. **What is the replication factor of DMP Project kafka cluster?**

DMP Project uses default replication factor that is 2.

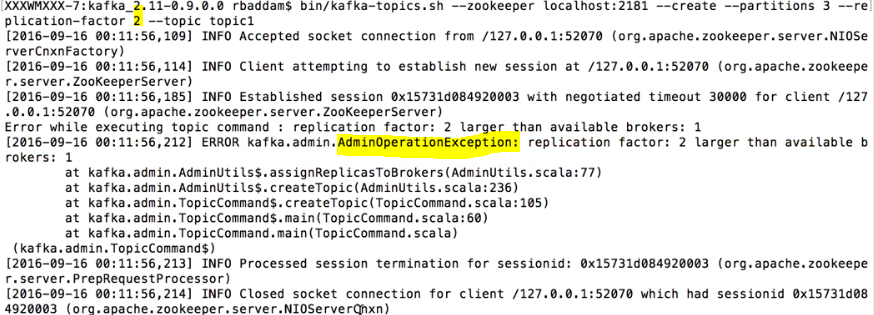
1. **Can replication factor be more than number of brokers you have?**

No. It is of no use to keep replica of same partition 2 times in same broker.

So replication factor value should not be greater than number of available brokers.

1. **What will happen if you try to create topic with replication factor more that number of brokers?**

Kafka will not allow to create like that it throws an exception



1. **What will be the effect if you configure more replication?**

More the replication it impacts performance more.

1. **What is Leader in kafka?**

**Leader:**

The node responsible for all read and write for the given partition. Every partition will have one server acting as leader and replicas of the partition will acts as followers.

1. **What are followers in Kafka?**

**Followers:** The nodes which follows the instructions of the leader are called as followers.

If a leader fails one of the followers will automatically become leader

1. **Explain the concept of Leader and Follower.**

Every partition in Kafka has one server which plays the role of a Leader, and zero or more servers that act as Followers.

The Leader performs all read and write operations for the partition, at the same time follower will replicate the leader.

In case of the Leader failure, one of the Followers will take on the role of the Leader. This ensures load balancing of the server.

1. **What roles do *Replicas* and the *ISR* play?**

**Replicas**: List of nodes that replicate particular partition even though they do not play the role of leader.

*Note: for assumption purpose I can consider replicas as broker/follower.*

**ISR:** Set of messages that are synced to leader partition and yet to sync to followers partition.

*Note: I think in sync replica is like keeping messages that are to be replicated to followers once it is replicated that will be deleted. This is*

1. **Why are Replications critical in Kafka?**

If we replicate data we will not lose messages there will be high availability.

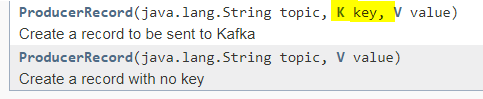
These messages can be consumed even in case of machine error, programs errors or frequent software upgrades.

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

It means that the Follower is unable to fetch data as fast as data accumulated by the Leader.

1. **How to achieve round robin partitioning in Kafka?**

If we want the keys to be written to all partition in round robin fashion do not pass key. If we do not pass key default partitioner will do job for us



1. **In the Producer, when does QueueFullException occur? IMP**

Producer sending messages at a pace that the Broker cannot handle.

In order to handle this we need to add enough brokers so that increased load is handled.

1. **What is concept of batching in producer?**

We know that in Kafka messages are written to file.

Each message causes some IO within the broker.

If the message size is going to be small and huge volume of messages comes into the cluster – there would be spike in IO which could impact performance.

To avoid that – We can store messages in the producer thread, and send message to broker only when message reach the configured size or no of messages. There are configurations to achieve this.

We should not keep more messages in producer thread also as it consumes more memory. We need to configure this to an ideal size.

**batch.size:** this is the producer configuration for to decide batch size.

1. Read below link and fit learnings in the doc

<https://community.hortonworks.com/articles/49789/kafka-best-practices.html>

## Consumers

1. **What is consumer group?**

Group of one or more consumers who jointly consumes messages from set of subscribed topics.

1. **What will happen when consumer goes down from group of consumers? IMP**

Say when 3 consumers are running with same group id then messages produced are divided across 3 consumers. Suppose if one of the consumer goes down the messages will be consumed by other 2 consumers even though one is down.

1. **How will you identify messaging model of consumer?**

**How kafka decides point to point and publish subscribe?**

1. Point to point: If all consumer have same group id then messages will be distributed to those consumers in point to point manner. That is one message can be read only by one consumer.
2. Publish and subscribe: If consumers have different group id then they will read all messages.

### If I have 2 partition for 1 topic and only one consumer what happens? If I have 2 what happens? If I have 3 what happens?

In case of 1 consumer both partitions will be read from single consumer

In case of 2 consumers one consumer will read from one and another consumer read from other.

In case of 3 consumers one consumer will be idle and other 2 consumers will read from one partition each.

1. **Work flow in Kafka?**

There are 2 kinds of work flow in Kafka.

1. **Queue messaging system: point to point (Consumer group)**

It is also called as consumer group.

In this case instead of single consumer a group of consumers with same id will be subscribed to a topic at any point of time message will be processed by one consumer.

Number of consumers should be less than or equal to number of partitions. If number of consumer becomes more (even 1) then the newly subscribed consumer to the group will not receive message.

**Note:** Kafka provides both publish and subscribe and point to point messaging system in an efficient way. We should use the feature as per our requirement.

1. **Topic messaging system: Pub-sub system:**

Producer sends messages to topic at some intervals, Kafka broker stores messages in to the partitions configured for that topic. It is responsibility of the broker to distribute the messages equally between configured partitions.

For ex if producer sends 2 message and if there are two partitions then Kafka stores one message in one partition and another on in other partition.

Consumer subscribes to a specific topic, once consumer subscribes to specific topic kafka provides current offset of the topic to the consumer also saves the offset in the zookeepr assembly.

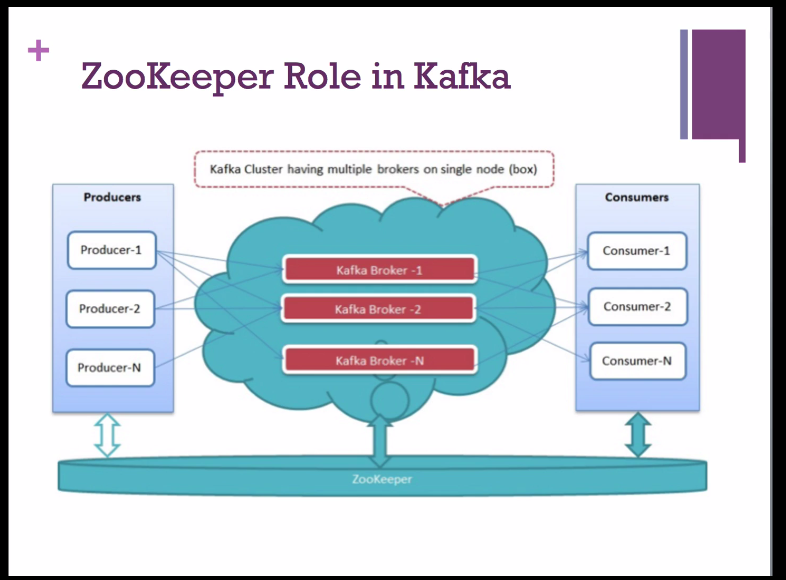
Consumer sends request to kafka at regular intervals like 100miliseconds for new messages once kafka receives messages from producers It forwards message to consumers. Consumers receives messages and process it.

Once the messages are processed consumer will send an acknowledgement to kafka brokers saying we got messages.once kafka receives acknowledgement it changes its offset value to the new value and updates zookeeper since offsets are maintained in zookeeper.

As offsets are maintained in zookeeper server can read correct messages even server went down. The same process will be repeated till consumer requests for the message

Consumer have an option to skip or rewind to desired offset of the topic in anytime and read all subsequent messages.

1. **What is the Role of Zookeeper in Kafka? IMP**



**Zookeeper:**

Kafka uses zookeeper to store offsets of the messages consumed from specific partition and topic by a specific consumer group.

1. Managing and coordinating Kafka broker and it is like interface between broker and consumer.
2. Kafka stores basic metadata in zookeeper such as information about the topics, number of partitions, brokers, consumer offsets and so on. Since all the critical information is stored in zookeeper it normally replicates its data across its assembly.
3. The leader election between the kafka broker is also done by zookeeper in the event of leader failure.
4. Failure of the kafka broker or zookeeper does not affect kafka cluster, kafka will restore the state once the zookeeper restarts this gives zero downtime for the kafka.
5. It is used to inform producer and consumer if any new broker is added to cluster or if any broker is failed.

Based on this producer and consumer takes decision and starts coordinating their task with some other brokers

The brokers depend on Zookeeper for:

a. Server failure detection.

b. Data partitioning.

c. In-sync data replication ISR.

d. Consumer membership management.

1. **Is it possible to use Kafka without Zookeeper? IMP**

Kafka will not work without Zookeeper.

Kafka uses Zookeeper for the following:

**Electing a controller**. The controller is one of the brokers and is responsible for maintaining the leader/follower relationship for all the partitions. When a node shuts down, it is the controller that tells other replicas to become partition leaders to replace the partition leaders on the node that is going away. Zookeeper is used to elect a controller, make sure there is only one and elect a new one it if it crashes.

**Cluster membership** - which brokers are alive and part of the cluster? This is also managed through ZooKeeper

**Topic configuration** - which topics exist, how many partitions each has, where are the replicas, who is the preferred leader, what configuration overrides are set for each topic

**(0.9.0) - Quotas** - how much data is each client allowed to read and write

**(0.9.0) - ACLs** - who is allowed to read and write to which topic (old high level consumer) - Which consumer groups exist, who are their members and what is the latest offset each group got from each partition.

1. **How many node zookeeper clusters does DMP Project has?**

5 node zookeepers.

Note: In streaming cluster where d project applications running there are many other applications it is shared cluster.

1. **What is the industry standard for zookeeper cluster?**

It is suggested to have at least 3 node zookeeper cluster.

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

We know that kafka uses zookeeper so we need to start the zookeeper server first and then start kafka server.

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

**Kafka server basic operations and commands?**

Single node single broker configuration

Single node multiple brokers configuration.

Topics and Its commands

1. **How to create kafka topic?**

Topics can be created using **kafka-topics.sh**

kafka-topics.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --create --topic CHANDRA-TEST --partitions 16 --replication-factor 2

Note: localhost:2181 is zookeeper server.

1. **How to see list of topics in kafka server?**

kafka-topics.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --list

1. **How to check details of created topic?**

**Using Describe command**

kafka-topics.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --topic CHANDRA\_TEST --describe

1. **How to modify already existing topics?**

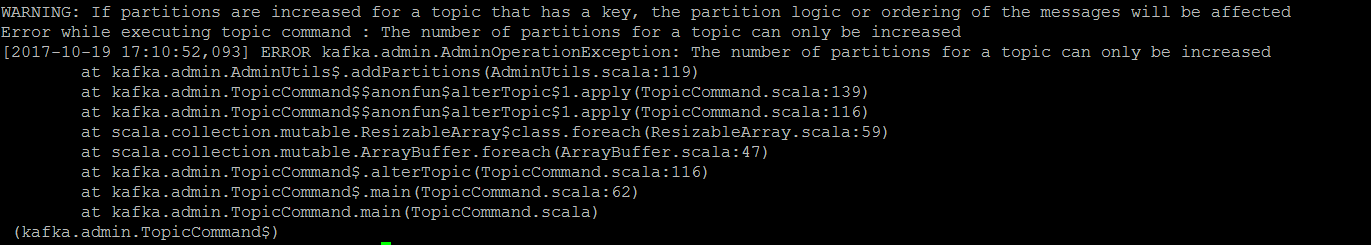
Using alter command.

kafka-topics.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --alter --topic CHANDRA\_TEST --partitions 4

1. **Can I decrease number of partitions of topic once it is created?**

**No**, we cannot decrease the number of partitions once created we can only increase the count.

If we try we will get exception



1. **Can I decrease number of replicas once it is created?**

No

1. **How to delete a topic?**

kafka-topics.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --delete --topic CHANDRA\_TEST

1. **How to restrict deleting a topic?**

If we set **delete.topic.enable = false** topics will not be deleted

1. **How to start console producer to send messages?**

kafka-console-producer.sh --broker-list 10.19.150.43:21005,10.19.150.44:21005 --topic CHANDRA-TEST

1. **How to start console consumer to receive messages?**

kafka-console-consumer.sh --zookeeper 10.19.150.21:24002,10.19.150.22:24002,10.19.150.23:24002/kafka --bootstrap-server 10.19.150.43:21005,10.19.150.44:21005 --topic CHANDRA-TEST

When we execute above command we usually receives messages which are produced after starting kafka consumer. There is option using which we can read the messages from the beginning

1. **How to configure Single node multi broker kafka set up?**

Can refer

D:\Technology\prudhvi videos\6.The Complete Apache Kafka course for beginners\09 Apache Kafka -Multi Node Cluster\ 027 Configuring Multi node Kafka.mp4

1. **Producer APIs?**

Can see video

<https://kafka.apache.org/090/javadoc/index.html>

1. **Explain the role of the Kafka Producer API? IMP**

It wrap two producers – kafka.producer.SyncProducer and the kafka.producer.async.AsyncProducer.

The goal is to expose all the producer functionality through a single API to the client.

1. **How producer works?**
   1. Messages from multiple producers are buffered/queued and batched data will be dispatched
   2. It handles serialization of data through user-specified encoder
   3. It provides load balancing through an optionally user-specified Partitioner
2. **Sample Producer?**

For writing a sample producer we can import all jars in $KAFKA\_HOME/libs and start writing project

Or Simply we can use below 2 Jars

kafka\_2.10-0.9.0.0.jar

kafka-clients-0.9.0.0.jar

1. **Consumer APIs?**

Can see video

<https://kafka.apache.org/090/javadoc/index.html>

1. **Explain High level Consumer API?**

Sometimes the logic to read messages from Kafka doesn’t care about handling the message offsets, it just wants the data.

High Level Consumer stores the last offset read from a specific partition in Zookeeper. This offset is stored based on the group id.

1. **Sample consumer?**

Simple consumer provides greater control over partition consumption than Consumer Groups give you.

We can have below controls,

Read a message multiple times.

Consume only a subset of the partitions in a topic in a process.

Manage transactions to make sure a message is processed once and only once.

Actually the high level consumer is good enough for most of the cases. Only applications require features which is not there in high level consumer (for ex: set initial offset when restarting the consumer) can use simple consumer

1. **What is the concept of compression in Kafka?**

We know that Kafka involves movement of huge messages across the network and to avoid any network bandwidth bottleneck kafka allows compression of the topics.

1. **Which are the compression codec supported by Kafka?**

*Gzip* compression

*Snappy* compression

*LZ4* compression

1. **Which is the configuration used to configure compression in Kafka?**

offsets.topic.compression.codec default 0

0: No compression, 1: GZIP compression, 2: Snappy compression, 3: LZ4 compression

1. **Which compression is used in D Project?**

D Project uses no compression the configuration is disabled in cluster

1. **Security of Kafka?**

Kafka 0.8 version does not provide any security authentication methods in their API. The security features will be implemented in coming versions

1. **Monitoring Kafka?**

Kafka writes lot of metrics, we can browse these metrics with any monitoring solution that uses JMX.

There are already some monitoring tools exists in the community like **Ganglina** and **Graphite**.

The major monitoring items for kafka are **Garbage collection** **cycles,CPU,IO, message transfer rate,broker state change and zookeeper logs.**

1. **Administrating Kafka?**

Most of the Kafka administration work is done using kafka utilities.

Kafka has not provided any UI interface but still there are few projects in the community which provide UI kind of utility for broker cluster.

Apart from that – lot of utilities are available to mirror the Kafka cluster across data center, work with zookeeper offsets, JMX tool, migration tool etc.

1. **Compare kafka and flume? IMP**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Kafka** | **Flume** |
| Data flow | Pull | Push |
| Hadoop Integration | Loose | Tight |
| Functionality | Publish-subscribe model messaging system | System for data collection, aggregation & movement |

## Kafka important features

### What are the delivery guarantees offered by Kafka?

https://dzone.com/articles/interpreting-kafkas-exactly-once-semantics

At most once delivery, at least once delivery and exactly once delivery.

### What is **at most once** delivery?

Message can either be delivered once or not at all. There can be a loss of messages, but a message can never be delivered more than once.

### What is **at least once** delivery?

Message will always be delivered. It can be delivered multiple times but there will never be any messages lost.

### What is **exactly once** delivery?

It guarantees that all messages will always be delivered exactly once. Exactly once does not mean that there will be no failures or no retries. These are inevitable. The important thing is that the retries succeed.

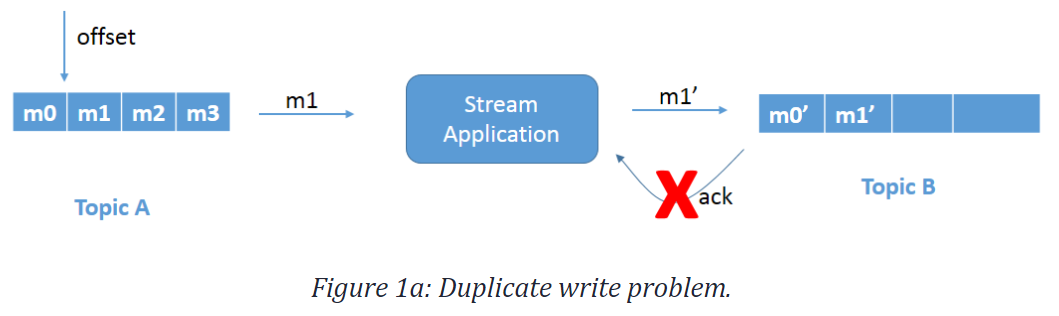
### Why exactly once is important?

There are certain use cases (like financial applications, IoT applications, and other streaming applications) which cannot afford anything less than exactly-once. You cannot afford to have duplicates or lose messages when depositing or withdrawing money from a bank account. It needs exactly-once as an outcome.

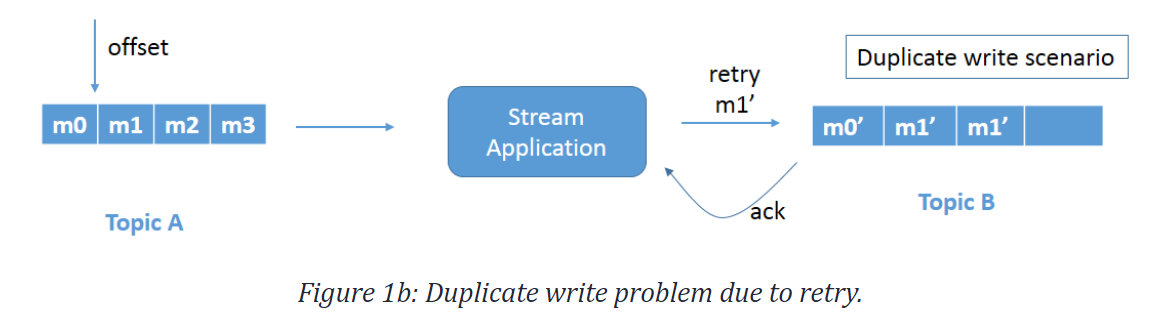
### Why is it difficult to achieve Exactly once?

Assuming you have a small Kafka, stream application which reads from some topic partition and writes to some topic partition. The expectation from the application is to receive data from input partitions, process the data, and write the same to output partitions. This is where one wants to achieve exactly-once as a guarantee. There are scenarios due to network glitches, system crashes, and other errors where duplicates get introduced during the process.

* Problem 1: Duplicate or Multiple Writes:

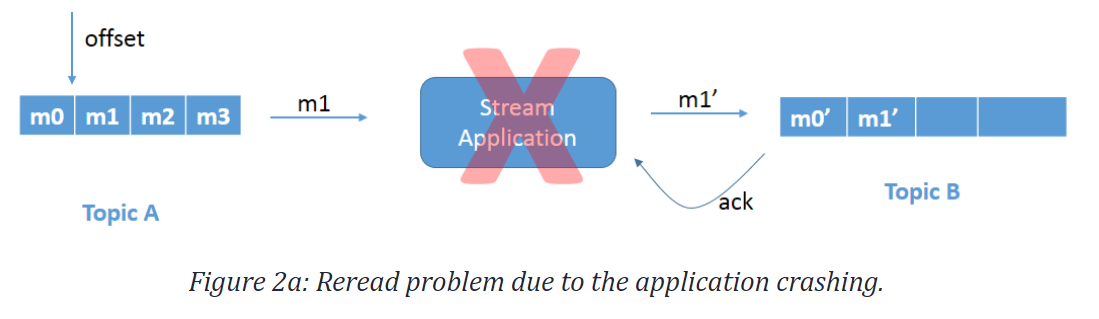


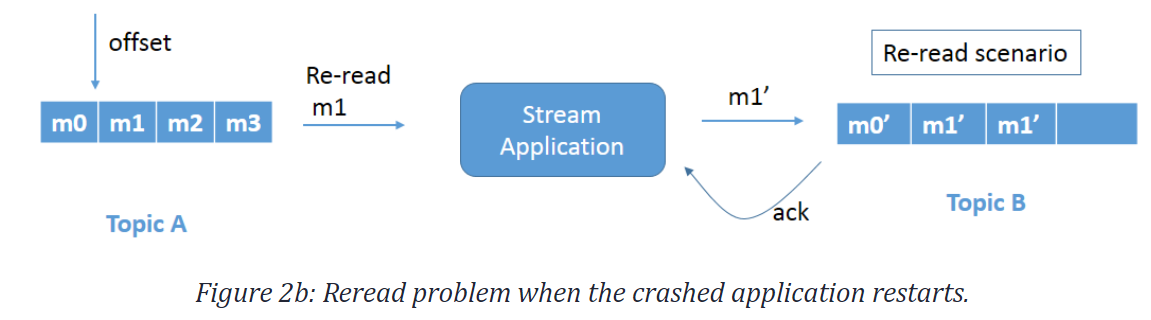
Refer to Figure 1a. Message m1 is being processed and being written to Topic B. Message m1 gets successfully written to Topic B (as m1') but the acknowledgment is not received. The reason could be, let's say, network delay and this eventually gets timed out.



Since the application does not know that the message is already successfully written, as it never received the acknowledgment, it retries and leads to a duplicate write. Refer to Figure 1b. Message m1' gets rewritten to Topic B. This is a duplicate write issue and needs to be fixed.

* Problem 2: Reread Input Record





Refer to Figure 2a. We have the same scenario as above, but, in this case, the stream application crashes just before committing the offset. Since the offset is not committed, when the stream application comes up again it rereads message m1 and processes the data again (Figure 2b). This again leads to duplicate writes of message m1 in Topic B.

*Apache Kafka solves these problems using exactly once semantics.*

### How Kafka achieves exactly once semantics? **IMP**

* **Idempotent Producer**

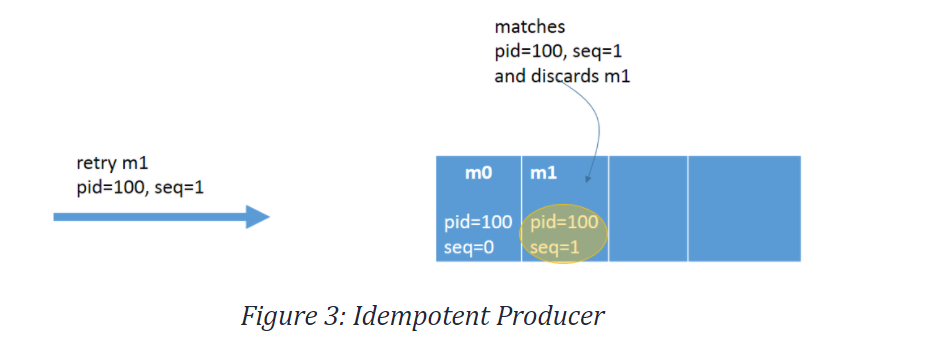
Idempotence means that applying an operation once or applying it multiple times has the same effect.

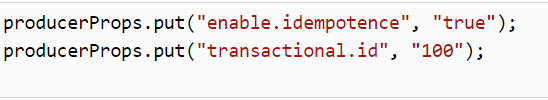
We must make sure producer will not write same messages to output Kafka topic. So we need to set producer configuration **enable.idempotence** to **true.**

When we enable this configuration each kafka message will get 2 things.

producer id (PID)

sequence number (seq)





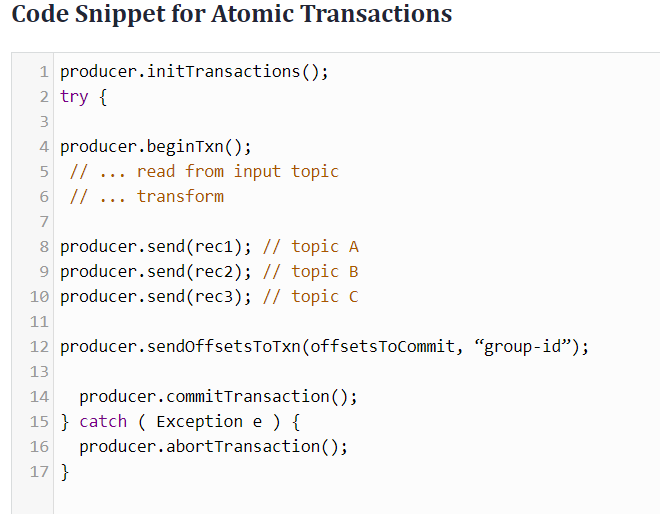
In the case of broker failure or client failure, there will be retried to send message in that case topic will only accept messages that have a new unique sequence number and producer id. If it receives same sequence number for same producer id (PID) the broker automatically deduplicates any message(s) sent by this producer, ensuring idempotency. No additional code changes are required.

* **Transactions Across Partitions**

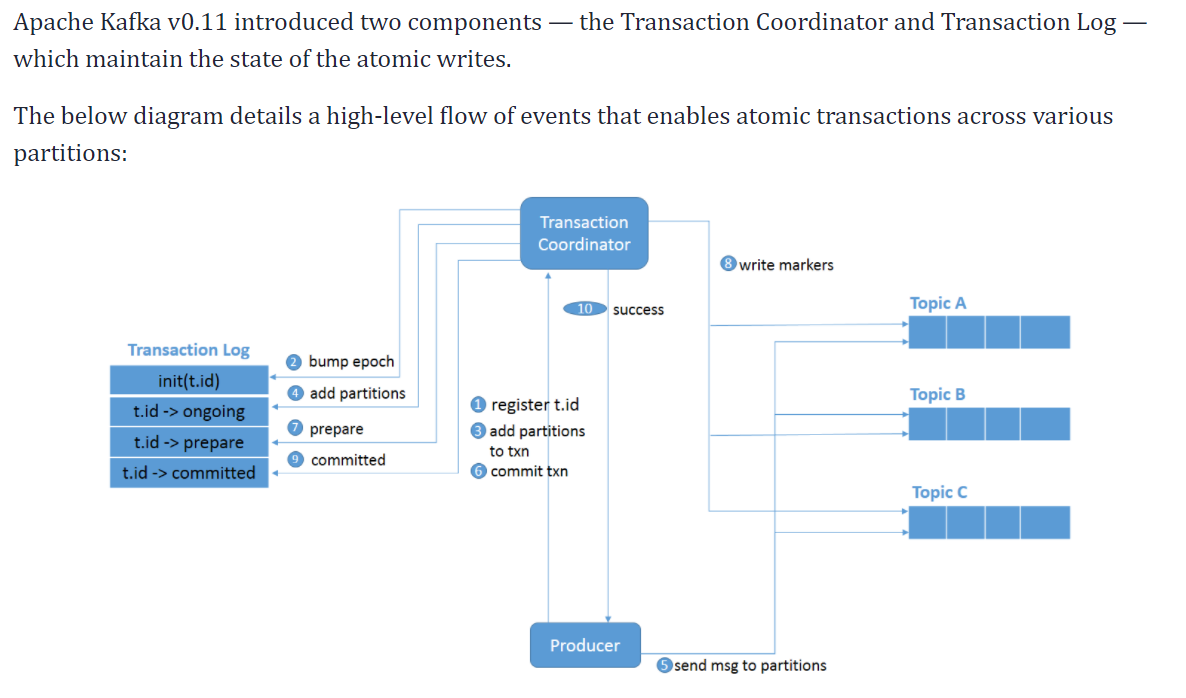
To ensure that each message gets processed exactly-once, transactions can be used.

Transactions works on principal of either process all messages or no messages.

Transactions ensure that after picking a message, that the message can be transformed and atomically written to multiple topics/partitions along with an offset of the consumed message.



Apache Kafka v0.11 introduced two components — the **Transaction Coordinator** and **Transaction Log** — which maintain the state of the atomic writes.



1. initTransactions() registers a transactional.id with the coordinator.
2. The coordinator bumps up the epoch of the PID so that previous instance of that PID is considered a zombie and fenced off. No writes in the future are accepted from these zombies.
3. The producer adds a partition with the coordinator when the producer is about to send data to a partition.
4. The transaction coordinator keeps the state of each transaction it owns in memory, and also writes that state to the transaction log (partition information, in this case).
5. The producer sends data to the actual partitions.
6. The producer initiates a commit transaction and, as a result, the coordinator begins the two-phase commit protocol.
7. This is where the first phase begins and the coordinator updates the transaction log to “prepare\_commit”.
8. The coordinator then begins Phase 2, where it writes the transaction commit markers to the topic-partitions which are part of the transaction.
9. After writing the markers, the transaction coordinator marks the transaction as “committed.”

* **Transactional Consumer**

If a consumer is transactional, we should use the isolation level, **read\_committed**.

This ensures that consumers read only messages that are committed.

The default value of **isolation.level** is **read\_uncommitted**

# Kafka Streams:

The easiest way to write mission-critical real-time applications and microservices

<https://www.youtube.com/watch?v=Z3JKCLG3VP4&feature=youtu.be>

## References

<https://kafka.apache.org/28/documentation/streams/>

## Basics

### What is Kafka stream?

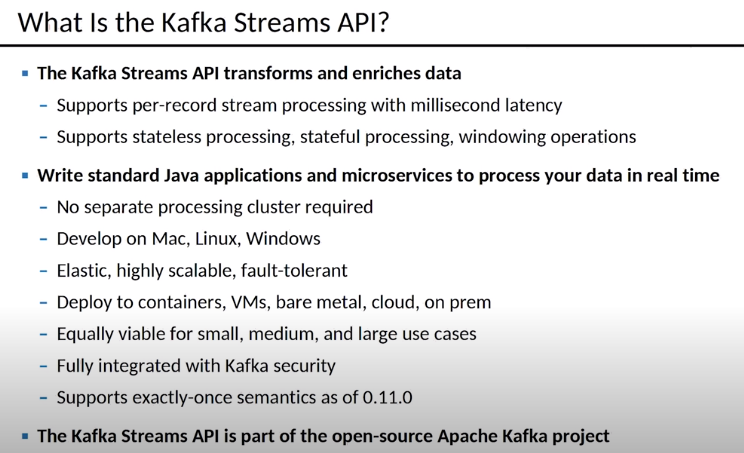
client library for processing and analyzing data stored in Kafka.

Using Kafka stream, we can build applications and microservices.

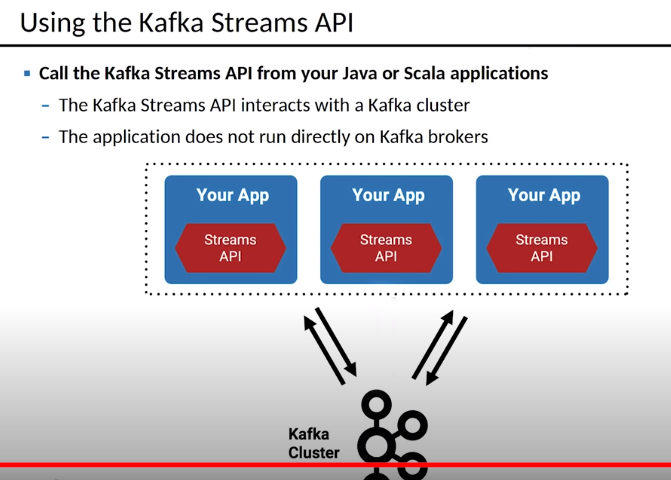
### Highlights or important information of Kafka Stream?

* Designed as a **simple and lightweight client library**, which can be easily embedded in any Java application and integrated with any existing packaging, deployment and operational tools that users have for their streaming applications.
* Has **no external dependencies on systems other than Apache Kafka** **itself** as the internal messaging layer; notably, it uses Kafka's partitioning model to horizontally scale processing while maintaining strong ordering guarantees.
* Supports **fault-tolerant local state**, which enables very fast and efficient stateful operations like windowed joins and aggregations
* Supports **exactly once** processing semantics to guarantee that each record will be processed once and only once even when there is a failure on either Streams clients or Kafka brokers in the middle of processing.
* Employs **one-record-at-a-time processing** to achieve millisecond processing latency and supports **event-time based windowing operations** with out-of-order arrival of records.
* Offers necessary stream processing primitives, along with a high-level Streams DSL and a low-level Processor API.

### Kafka stream API?

’

### Using Kafka Stream API?



### Scalability in Kafka stream?

<https://www.confluent.io/blog/elastic-scaling-in-kafka-streams/>

consumer group supports scalability in Kafka stream.

scalability is achieved in streams by deploying multiple instances of the same application. Stream api takes care of everything.

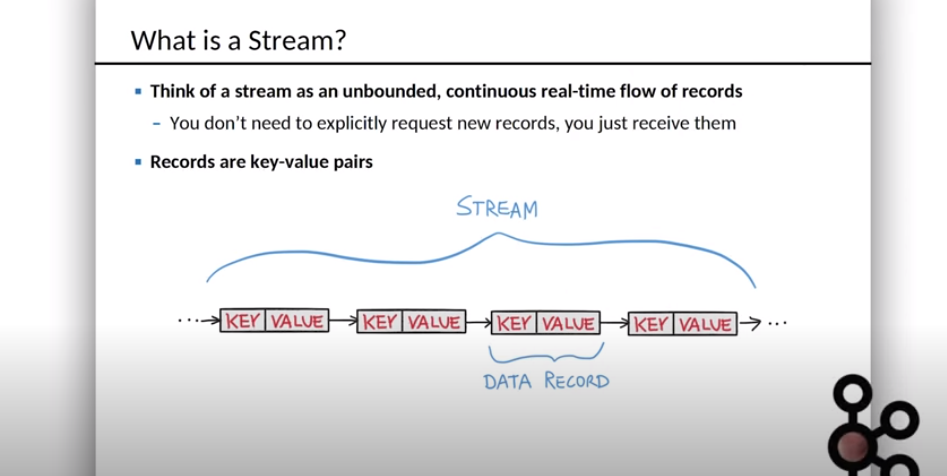
## Stream Processing Topology

https://kafka.apache.org/28/documentation/streams/core-concepts

### What is Stream?

unbounded, continuous real time flow of records.

A stream is an *ordered*, *replayable*, and *fault-tolerant sequence of immutable data records*, where a data record is defined as a key-value pair.



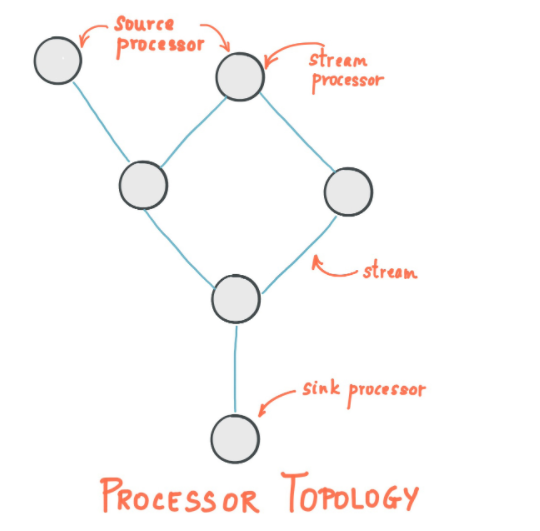
### What is Stream processing application?

program that makes use of the Kafka Streams library for example Preprocessor.

Streaming applications defines its computational logic through one or more processor topologies

### What is processor topology? **IMP**

graph of stream processors (nodes) that are connected by streams (edges).



### What is Source Processor? **IMP**

Source processor consumes records from one or more topics and forwards it to its down-stream processors.

A source processor is a special type of stream processor that does not have any upstream processors.

### Can source processor read from multiple topics?

Yes

### What is Sink processor?

It receives records from its up-stream processors and sends it to a specified Kafka topic.

A sink processor is a special type of stream processor that does not have down-stream processors.

### What is Stream processor? **IMP**

Stream processor receives one record at a time from its upstream processors in the topology and transforms stream data by applying the logic associated with it and produces one or more output records to its downstream processors.

A stream processor node can access other remote systems while processing the current record. Therefore, the processed results can either be streamed back into Kafka or written to an external system.

### What are all the ways in which you can define Preocessor topology?

* **Processor API**

allows developers define and connect custom processors as well as to interact with state stores.

* **Kafka Streams DSL (D**omain **S**pecific **L**anguage**)**

provides the most common data transformation operations such as map, filter, join and aggregations out of the box.

processor topology is a logical abstraction for your stream processing code,

At runtime, the logical topology is instantiated and replicated inside the application for parallel processing

## Kafka Stream Architecture

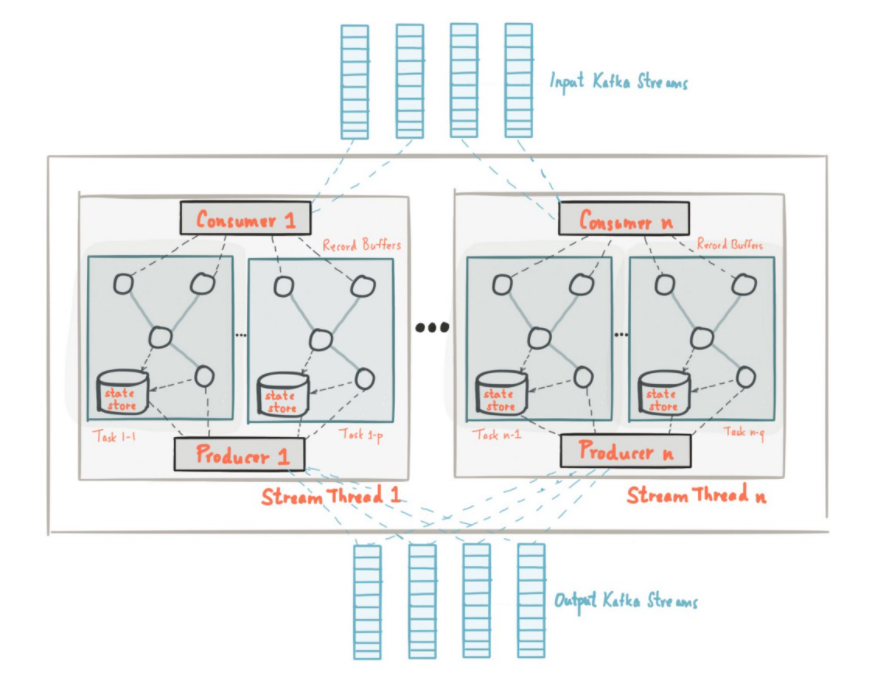
https://kafka.apache.org/21/documentation/streams/architecture

### What is Stream thread in Kafka stream?

In the beginning of this post we mentioned that Kafka Streams library is built on top of consumer/producer APIs and data processing is organized in exactly same way as a standard Kafka solution. In Kafka Streams there’s notion of application.id configuration which is equivalent to group.id in the vanilla consumer API.

The Streams library creates pre-defined number of Stream Threads and each of these does data processing from one or more partitions of the input topic(s

Stream threads are the main way of scaling data processing in Kafka Streams, this can be done vertically, by increasing the number of threads for each Kafka Streams application on a single machine, or horizontally by adding an additional machine with the same application.id.



### Kafka Stream stateless processing?

### Kafka stream stateful processing?

### Kafkastream windowing operations?

### Why we should use kafka streams instead of spark streaming?

### In your project kafka stream is stateless or stateful?

### How Kafka stream is fault tolerant?

### Does Kafka stream support stateful and stateless operations?

Yes, Kafka stream can do stateful and stateless processing on real time data.

In stream processing, there is a notion of stateless and stateful operations. State is anything your application needs to “remember” beyond the scope of the single record currently being processed.

Stateless operations (filter, map, transform, etc.) are very simple, since there is no need to keep the previous state and a function is evaluated for each record in the stream individually.

Stateful operations such as basic count, any type of aggregation, joins, etc. are much more complex. This is because with only one record you can’t determine the latest state (let’s say count) for the given key, thus you need to hold the state of your stream in your application. As we have discussed in the Kafka: Data Partitioning section, each thread in Kafka Streams handles set of unique partitions, therefore the thread handles only a subset of the entire data stream. What it means is that, if needed, each thread of a Kafka Streams application with the same application.id maintains its own, isolated state. We won't go into details on how state is handled in Kafka Streams, but it's important to understand that state is backed-up as a change-log topic and is saved not only on the local disk, but on Kafka Broker as well.

### How will you achieve high availability in Kafka Stream?

stateful Kafka Stream

https://medium.com/transferwise-engineering/achieving-high-availability-with-stateful-kafka-streams-applications-cba429ca7238

### Why achieving high availability in Kafka stream is problematic?

### Explain how to achieve high availability using stateful applications?

## Consumer

### What is Consumer Group Rebalancing?

As we said earlier, each consumer group instance gets set of unique partitions from which it consumes the data. Whenever a new consumer instance joins the group, rebalancing should happen for the new instance to get its partition assignments. The same thing happens when a consumer instance dies, the remaining instances should get a new assignment to ensure all partitions are being processed.

# RabbitMQ

## What is rabbit MQ?

open-source distributed message broker

It helps in efficient message delivery in complex routing scenarios.

RabbitMQ employs a push model and prevents overwhelming users via the consumer configured prefetch limit. This model is an ideal approach for low-latency messaging. It also functions well with the RabbitMQ queue-based architecture. Think of RabbitMQ as a post office, which receives, stores, and delivers mail, whereas RabbitMQ accepts, stores, and transmits binary data messages.

RabbitMQ natively implements AMQP 0.9.1 and uses plug-ins to offer additional protocols like AMQP 1.0, HTTP, STOMP, and MQTT. RabbitMQ officially supports Elixir, Go, Java, JavaScript, .NET, PHP, Python, Ruby, Objective-C, Spring, and Swift. It also supports various dev tools and clients using community plug-ins.

## What is Kafka used for?

Kafka can be used real-time processing and analyzing data.

Use Kafka if you need a framework for storing, reading, re-reading, and analyzing streaming data.

Use when you do not have complex routing, but with maximum throughput.

It’s also ideal stream processing, and carrying out modeling changes to a system as a sequence of events

Kafka is also suitable for processing data in multi-stage pipelines.

## What is RabbitMQ is used for?

RabbitMQ to perform complex routing to consumers

Integrate multiple applications and services with non-trivial routing logic.

RabbitMQ with long-running tasks, reliably running background jobs, and communication/integration between and within applications.

|  |  |  |
| --- | --- | --- |
| Type | **Kafka** | **RabbitMQ** |
| **Performance** | Kafka performs better than Rabbit MQ | Rabbit MQ is slow compared to kafka. |
| **Use cases** | Massive data/high throughput cases | Simple use cases |
| **Payload Size** | Default 1MB limit | No constraints |
| **Message Retention** | Policy-based (e.g., 30 days) | Acknowledgment based |
| **Topology** | Publish/subscribe based | Exchange type: Direct, Fan out, Topic, Header-based |
| **Data Type** | Transactional | Operational |
| **Consumer Mode** | Consumer Mode | Dumb broker/smart consumer |

Data Flow

RabbitMQ uses a distinct, bounded data flow. Messages are created and sent by the producer and received by the consumer. Apache Kafka uses an unbounded data flow, with the key-value pairs continuously streaming to the assigned topic.

Data Usage

RabbitMQ is best for transactional data, such as order formation and placement, and user requests. Kafka works best with operational data like process operations, auditing and logging statistics, and system activity.

Messaging

RabbitMQ sends messages to users. These messages are removed from the queue once they are processed and acknowledged. Kafka is a log. It uses continuous messages, which stay in the queue until the retention time expires.

Design Model

RabbitMQ employs the smart broker/dumb consumer model. The broker consistently delivers messages to consumers and keeps track of their status. Kafka uses the dumb broker/smart consumer model. Kafka doesn’t monitor the messages each user has read. Rather, it retains unread messages only, preserving all messages for a set amount of time. Consumers must monitor their position in each log.

Topology

RabbitMQ uses the exchange queue topology — sending messages to an exchange where they are in turn routed to various queue bindings for the consumer’s use. Kafka employs the publish/subscribe topology, sending messages across the stream to the correct topics, and then consumed by users in the different authorized groups.

## In What use cases we should use Kafka?

* Streaming and processing data in multiple-stage pipelines.
* Streams with a throughput of at least 110K/sec events.
* Applications that need a stream history, delivered in “at least once” partitioned order.
* Event sourcing or system modeling changes as a sequence of events

## In what use cases we should use RabbitMQ?

* Complex routing to users/consumers
* Applications requiring a variety of publish/subscribe, or point-to-point request/reply messaging capabilities
* Applications that must support legacy protocols, like STOMP, MQTT, AMQP, 0-9-1