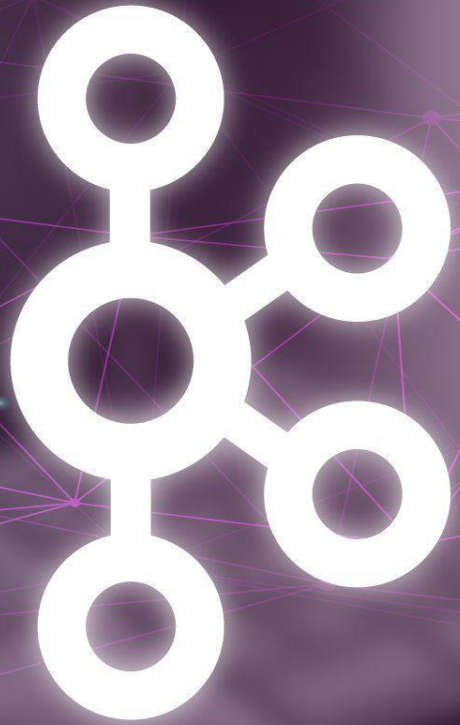
# Apache Kafka



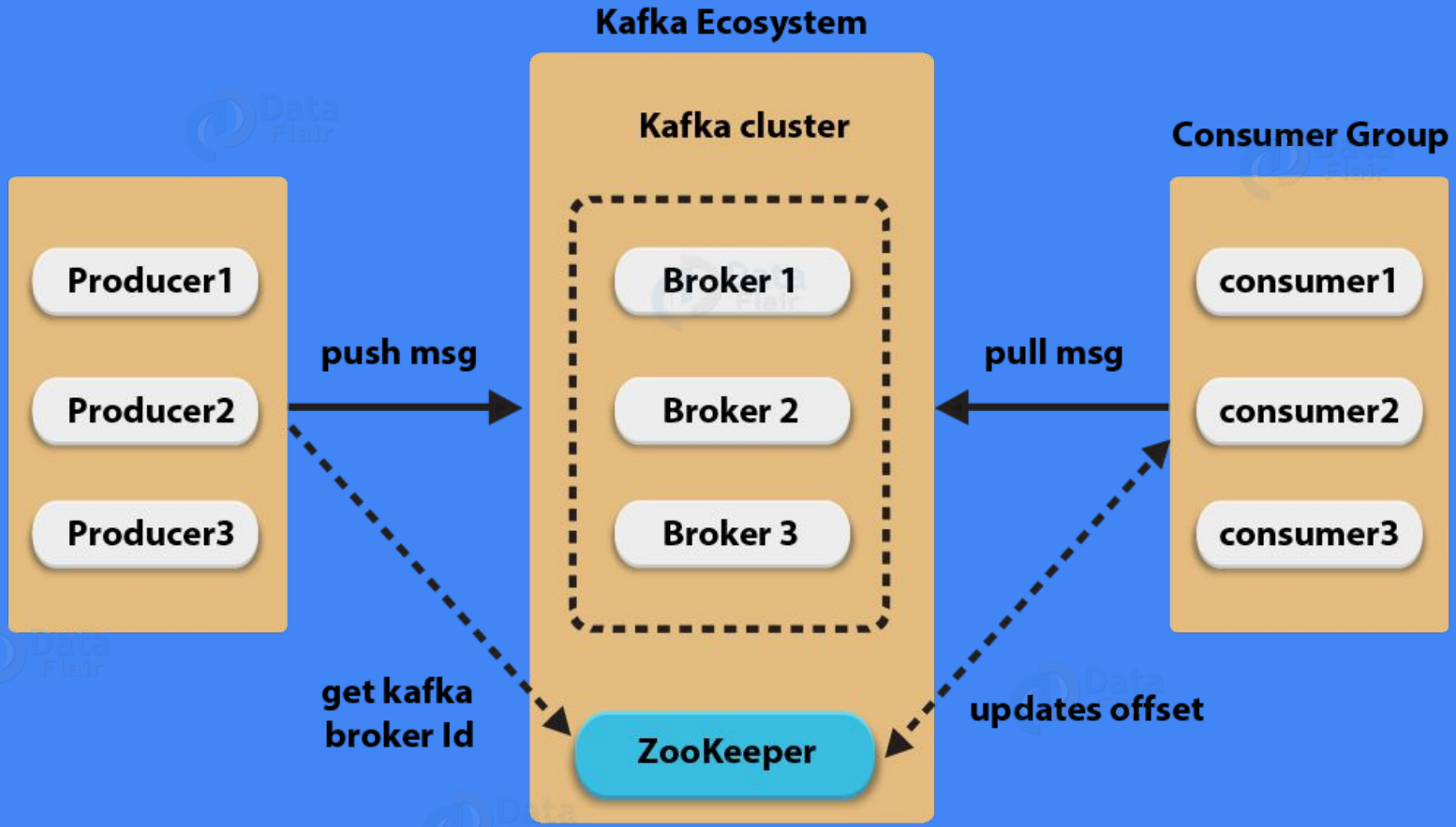
# What is **Kafka**

**Apache Kafka** is a fast, scalable, fault-tolerant messaging system which enables communication between producers and consumers using message-based topics.

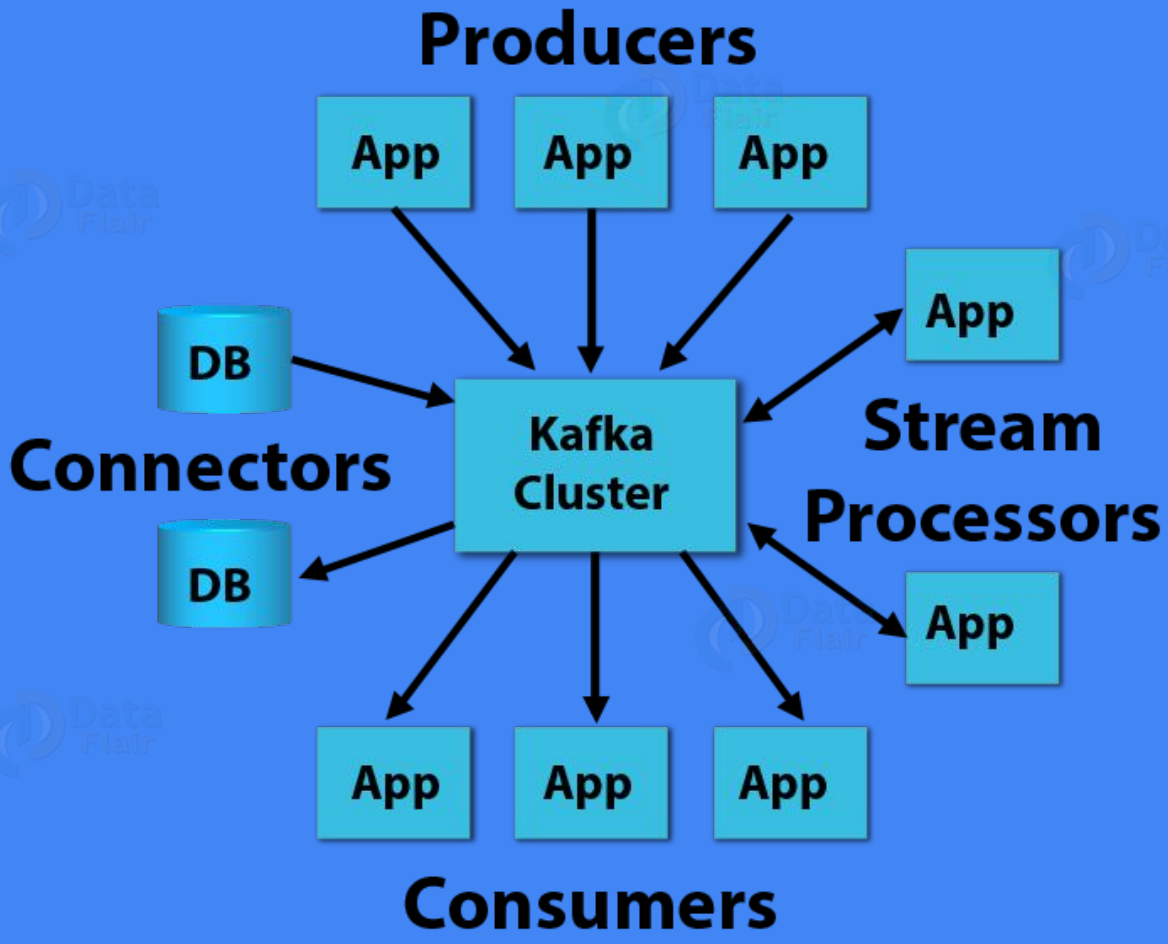
In simple words, it designs a platform for high-end new generation distributed applications.

One of the best features of **Kafka** is, it is highly available and resilient to node failures and supports automatic recovery.

# **Kafka** Architecture


Kafka Ecosystem
Kafka cluster
Consumer Group
Producer1
Producer2
Producer3
push msg
Broker 1
Broker 2
Broker 3
pull msg
consumer1
consumer2
consumer3
get kafka
broker Id
ZooKeeper
updates offset

# Apache Kafka 4 core APIs


Producers
App
App
App
DB
Connectors
DB
Kafka
Cluster
App
Stream
Processors
App
App
App
App
Consumers

# 1. Kafka Producer API

This Kafka Producer API permits an application to publish a stream of records to one or more Kafka topics.

# 2. Kafka Consumer API

The Consumer API permits an application to take one or more topics and process the continuous flow of records produced to them.

# 3. Kafka Streams API

The Streams API permits an application to behave as a stream processor, consuming an input stream from one or more topics and generating an output stream to one or more output topics, efficiently modifying the input streams to output streams.

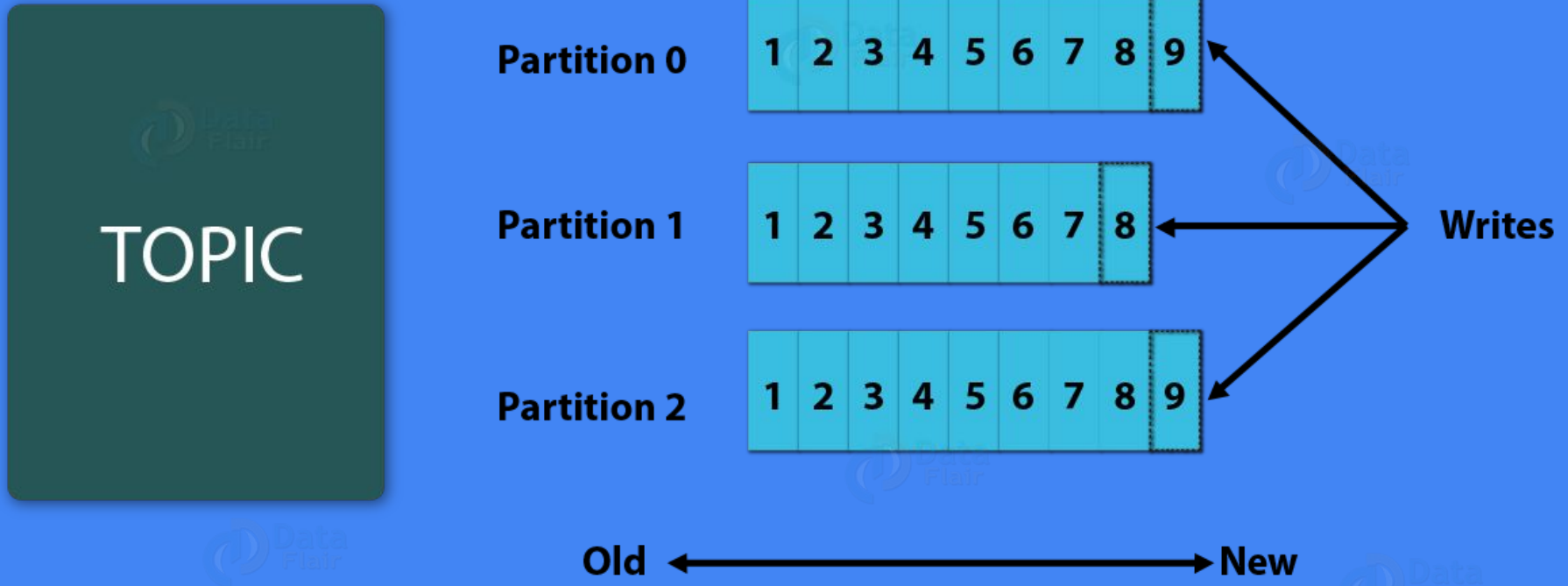
# 4. Kafka Connector API

The Connector API permits creating and running reusable producers or consumers that enables connection between Kafka topics and existing applications or data systems.

# **Kafka** Components

# Kafka Topic

A bunch of messages that belong to a particular category is known as a Topic. Data stores in topics. In addition, we can replicate and partition Topics.
Here, replicate refers to copies and partition refers to the division. Also, visualize them as logs wherein, Kafka stores messages.
However, this ability to replicate and partitioning topics is one of the factors that enable Kafka's fault tolerance and scalability.


TOPIC
Partition 0
1 2 3 4 5 6 7 8 9
Partition 1
1 2 3 4 5 6 7 8
Partition 2
1 2 3 4 5 6 7 8 9
Writes
Old
New

# Kafka Producer

The producers publish the messages on one or more Kafka topics.

# Kafka Consumer

Consumers take one or more topics and consume messages that are already published through extracting data from the brokers.

# Kafka Broker

These are basically systems which maintain the published data.

A single broker can have zero or more partitions per topic.

# Kafka Zookeeper

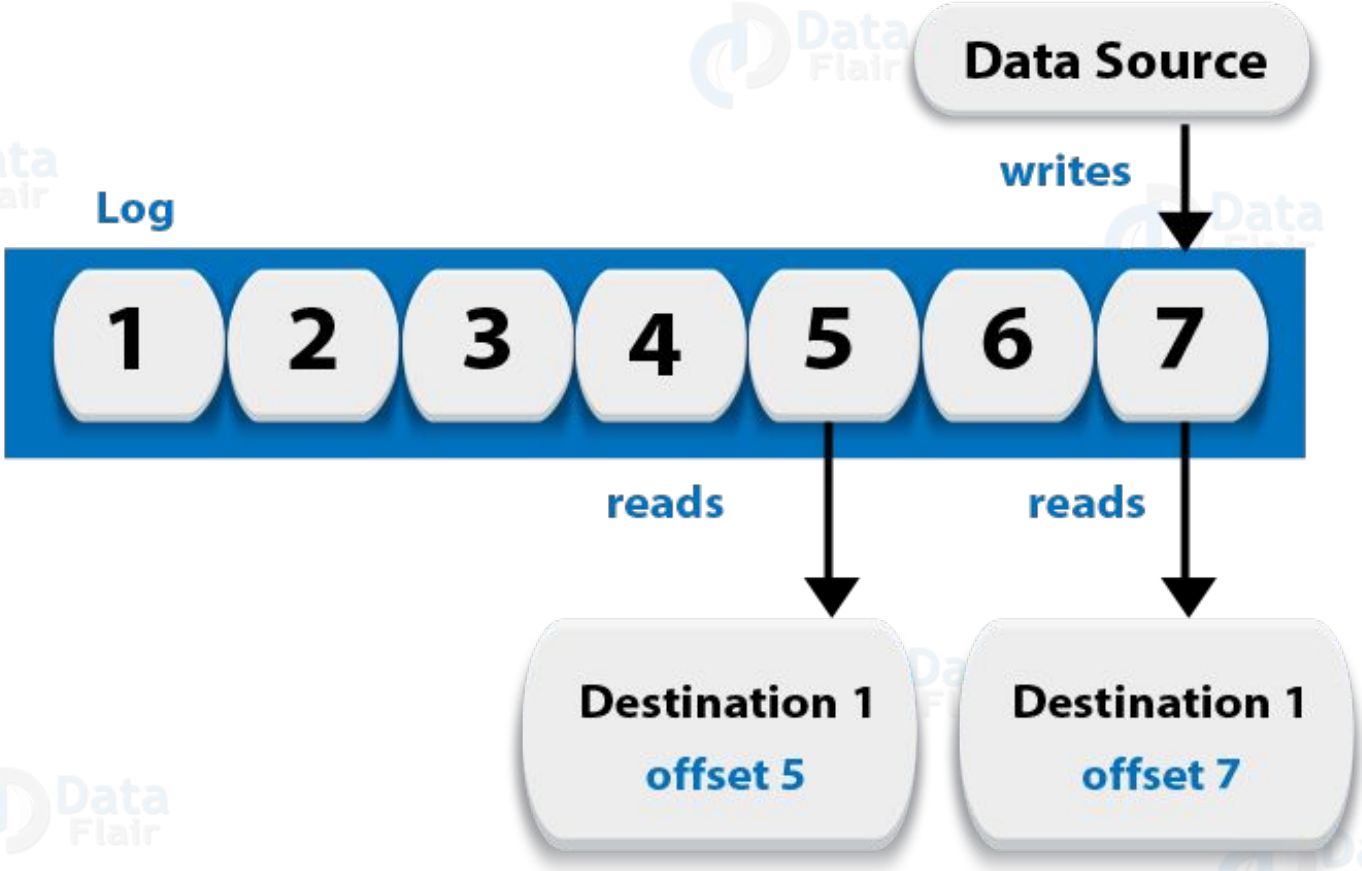
With the help of zookeeper, Kafka provides the brokers with metadata regarding the processes running in the system and grants health checking and broker leadership election.

# Kafka Log Anatomy

Basically, a data source writes messages to the log.

One of the advantages is, at any time one or more consumers can read from the log they select.

Here, the below diagram shows a log is being written by the data source and read by consumers at different offsets.


Data Source
writes
Log
1
2
3
4
5
6
7
reads
reads
Destination 1
offset 5
Destination 1
offset 7

# **Kafka** Offsets

The offset is a position within a partition for the next message to be sent to a consumer.

Kafka maintains two types of offsets.

1. Current offset
2. Committed offset

# Current Offset

When we call a poll method, Kafka sends some messages to us. Let us assume we have 100 records in the partition. The initial position of the current offset is 0. We made our first call and received 20 messages. Now Kafka will move the current offset to 20. When we make our next request, it will send some more messages starting from 20 and again move the current offset forward. The offset is a simple integer number that is used by Kafka to maintain the current position of a consumer. That's it. The current offset is a pointer to the last record that Kafka has already sent to a consumer in the most recent poll. So, the consumer doesn't get the same record twice because of the current offset.

# Committed Offset

this offset is the position that a consumer has confirmed about processing. What does that mean? After receiving a list of messages, we want to process it. Right? This processing may be just storing them into HDFS. Once we are sure that we have successfully processed the record, we may want to commit the offset. So, the committed offset is a pointer to the last record that a consumer has successfully processed. For example, the consumer received 20 records. It is processing them one by one, and after processing each record, it is committing the offset.

# Offsets Summary

1. Current offset -> Sent Records -> This is used to avoid resending same records again to the same consumer.
2. Committed offset -> Processed Records -> It is used to avoid resending same records to a new consumer in the event of partition rebalance.

The committed offset is critical in the case of partition rebalance.

In the event of rebalancing. When a new consumer is assigned a new partition, it should ask a question. Where to start? What is already processed by the previous owner? The answer to the question is the committed offset.

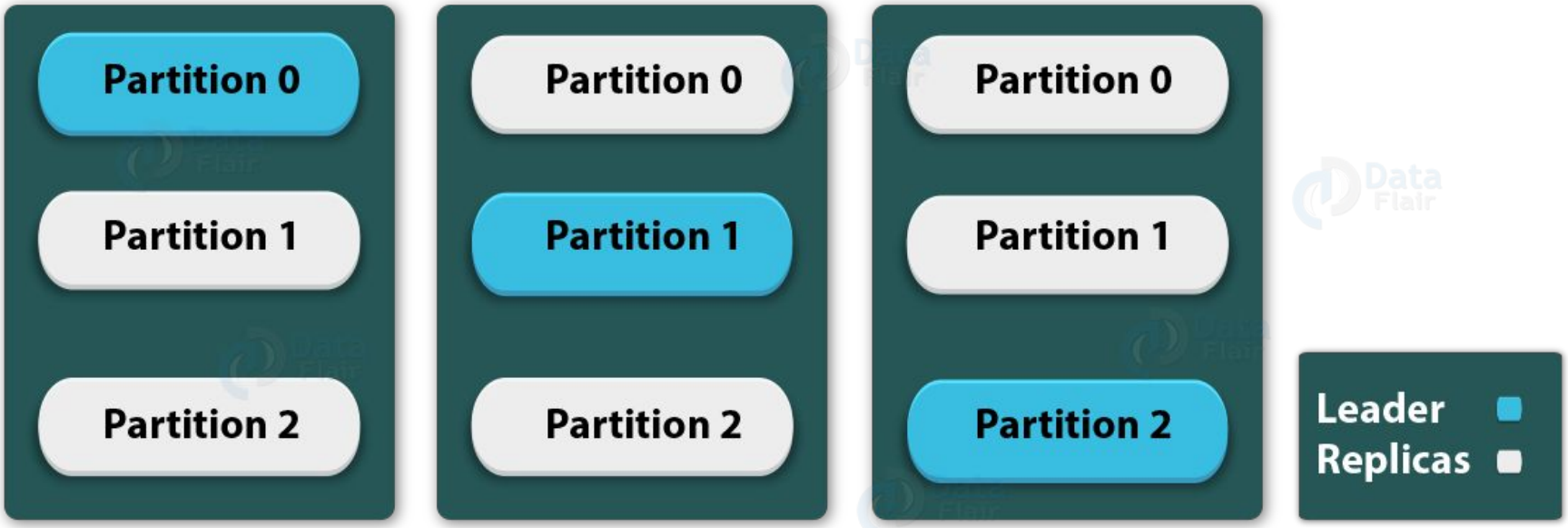
# **Kafka** Partition

There are few partitions in every Kafka broker.

Each partition can be either a leader or a replica of a topic.

In addition, along with updating of replicas with new data, Leader is responsible for all writes and reads to a topic.

The replica takes over as the new leader if somehow the leader fails.


Partition 0
Partition 1
Partition 2
Partition 0
Partition 1
Partition 2
Partition 0
Partition 1
Partition 2
Leader
Replicas


Questions?