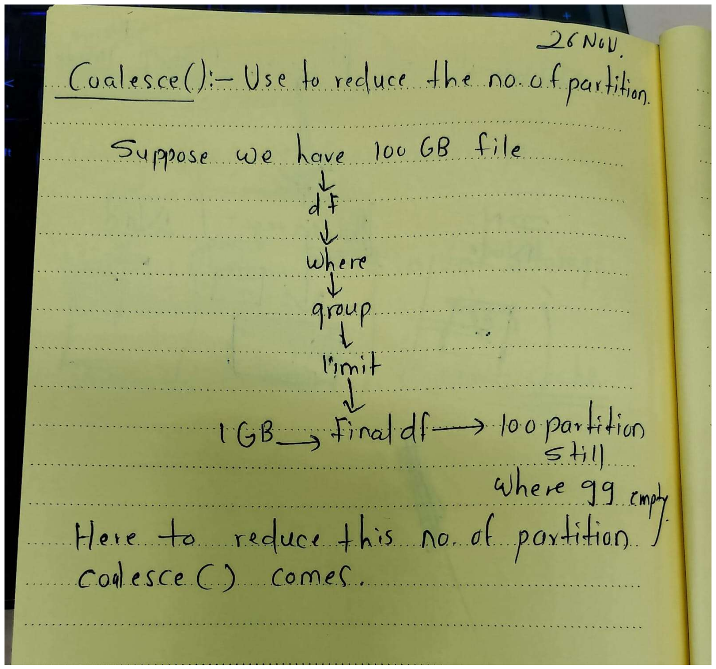
Coalesce

26 November 2025 09:41

This transformation is used to reduce the no. of partitions



26 Nov.

Coalesce():- Use to reduce the no. of partition.

Suppose we have 100 GB file

$$\begin{tikzcd}
& \text{df} \arrow[d] & \\
& \text{where} \arrow[d] & \\
& \text{group} \arrow[d] & \\
& \text{limit} \arrow[d] & \\
\text{1 GB} \arrow[r] & \text{final df} \arrow[r] & \text{100 partition still where 99 empty}
\end{tikzcd}$$

Here to reduce this no. of partition coalesce() comes.

Problem : Suppose we have a 100GB file.
We make it 100 partitions
We want to process it

- While processing, we'll put a lots of transformations on it.
- After where, Group by, order by, limit, transformations ,
- We'll still have 100 partitions of result data.

But the problem is :
 suppose the result will be only 1 GB.
So it can be stored in 1 partition only.

But we have 100 partitions, 99 are waste.
still we have to write those empty 99 partitions in our storage.

To prevent this we use one more transformation i.e. Coalesce()

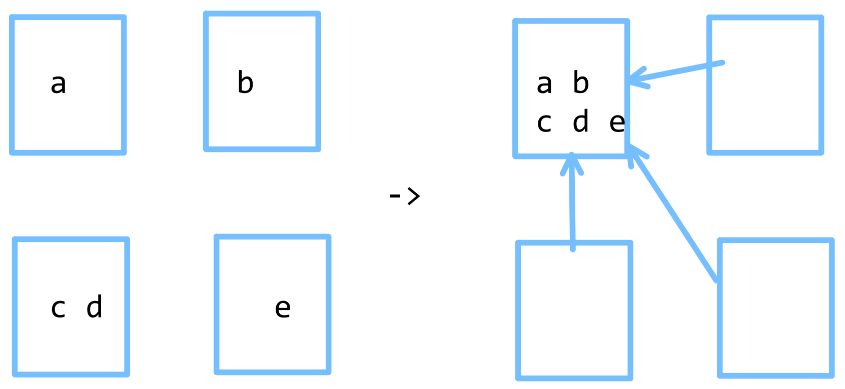
It reduces the number of partitions.

Repartition :

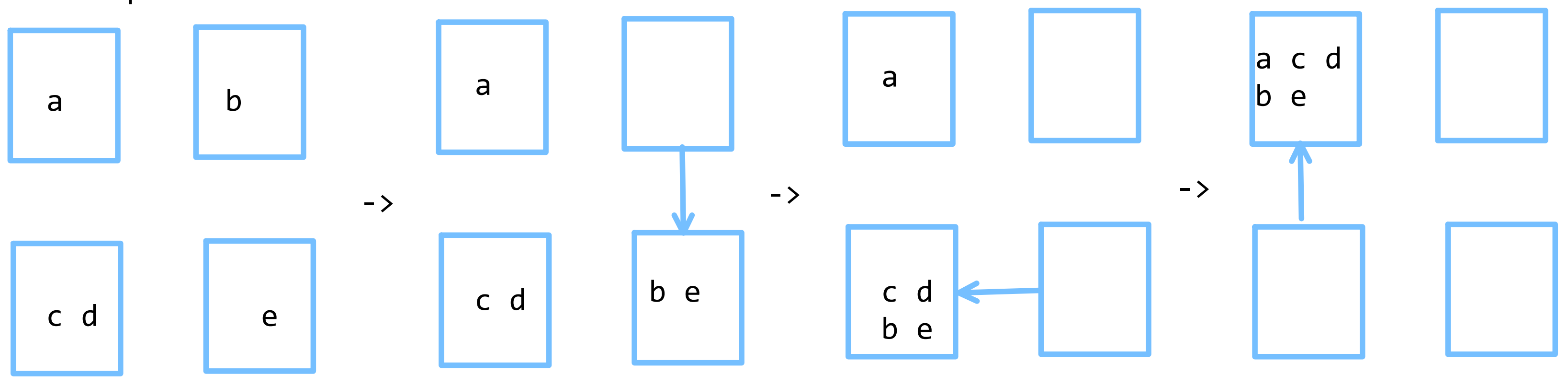
- This transformation can change the number of partitions. (Increase/Decrease)
- But we never use this to decrease this to decrease.
- Reason :

  • Suppose we have 4 partitions :

  • coalesce



- Repartition



Partition repeatedly moves data, making it slower.
Coalesce moves data at one go.

Exercise :

Go to databricks.
Workplace -> LearningSpark folder -> python
Chapter 4 -> 4-2 Spark data sources notebook. Run cells

Now Open 4-1 Example.

<u>Kafka</u>

- Kafka is a message queue.

- It also identified as " pub/sub" (publish-subscribe).

- Apache Kafka is an open-source, distributed, and publish-subscribe messaging system that maintains and manages a stream of real-time data from different applications, websites, different databases, etc.

- In 2011, Apache Kafka became an open sourced Apache project, and in 2012, it was made a first-class Apache project.

- Scala and Java are the languages used by Kafka.

- Kafka delivers high performance, scalability, durability, fault tolerance, and a distributed computing experience by design.

- Streaming applications and real-time pipelines can be built using Kafka.

- All this kafka thing works on cloud

Why Kafka?

Problem before Kafka:

- Many applications needed to communicate directly with each other.
- Example: Web server, App server, Chat server, Security system, PowerBI, Spark Streaming, etc.
- Each app needed a direct connection with every other app.
- This caused:
    - huge network load
    - many point-to-point connections
    - difficult debugging (if message failed: was issue app1? app2? network?)
- System became tightly coupled and hard to manage.

Kafka as a solution:

- Kafka acts as a central data pipeline / messaging backbone.
- Instead of each app talking to every other app:
  - Each app communicates with Kafka

    • Kafka handles delivering messages to consumers
  - Kafka stores incoming data/messages on disk (persistent).
  - Kafka forwards messages reliably to destination applications.

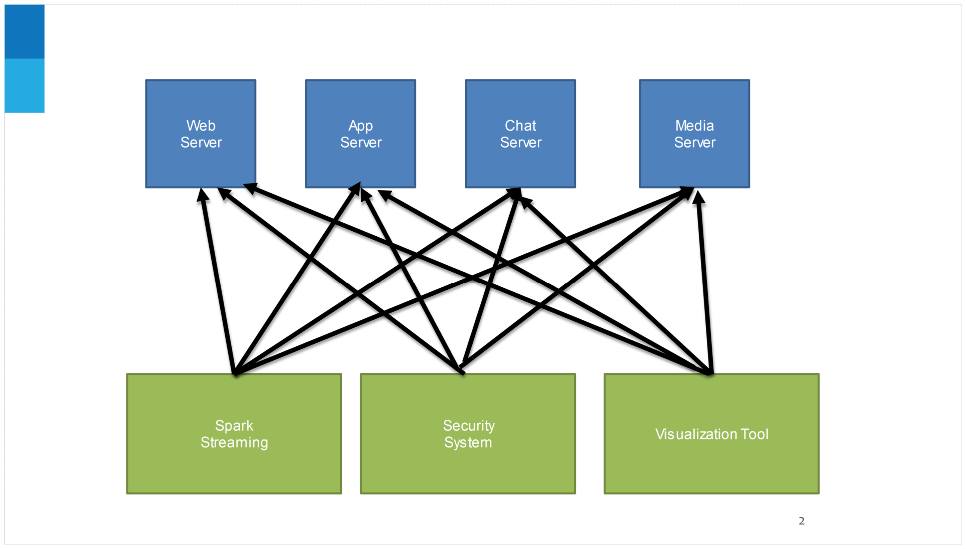
Key Advantages of Kafka:

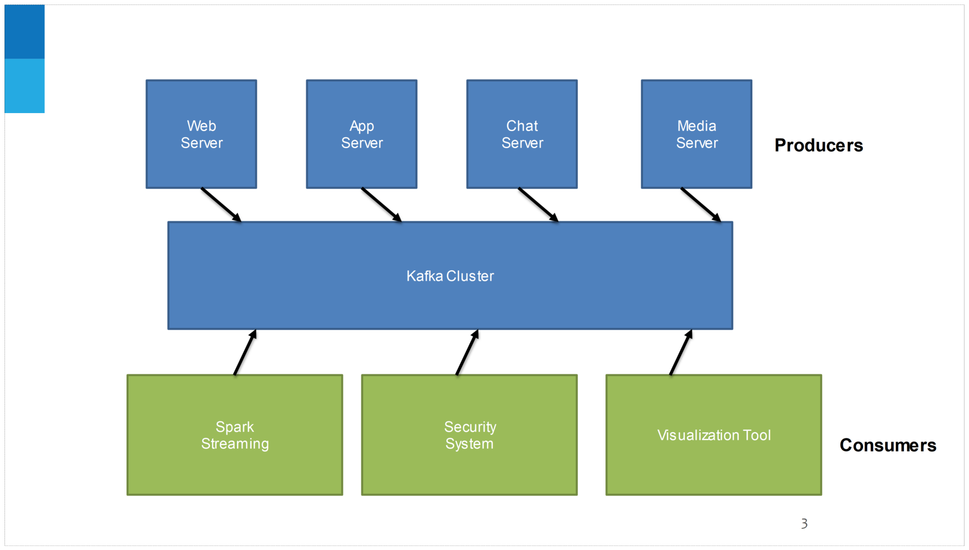
  - Decouples producers and consumers.
  - Reliable message delivery, even if consumer is offline for a while.
  - High performance and low latency.
  - Horizontally scalable using partitions and topic replication.
  - Durable: messages are stored on disk, not lost after delivery.
  - Supports publish-subscribe and streaming patterns.
  - Allows multiple consumers to read the same data independently.
  - Kafka carries out asynchronous communication.

Important Concepts:

  - Producer: sends data/messages to Kafka.
  - Consumer: reads data/messages from Kafka.
  - Topic: a logical channel/category where messages are grouped.
  - Partition: topics are split for parallelism and scalability.
  - Broker: Kafka server that stores messages.
  - Consumer group: group of consumers consuming from a topic in parallel.

In short:
 - Instead of N applications having N^2 direct connections,
 - Kafka provides a hub-based architecture where messages flow efficiently, reliably, and in a scalable manner.





Topic :

- In Kafka, all messages are maintained in topics. And these topics hold, publish, and organize messages.
- A logical channel/category where messages are grouped.
- Before propagating data, we need to create a topic.
- Producer sends the data to topic.
- Consumer subscribes to the topic.

Broker:

- Brokers are servers that manage and mediate the conversation between two different systems and ensure that appropriate messages are delivered to the right parties.
- Brokers replicate the data.

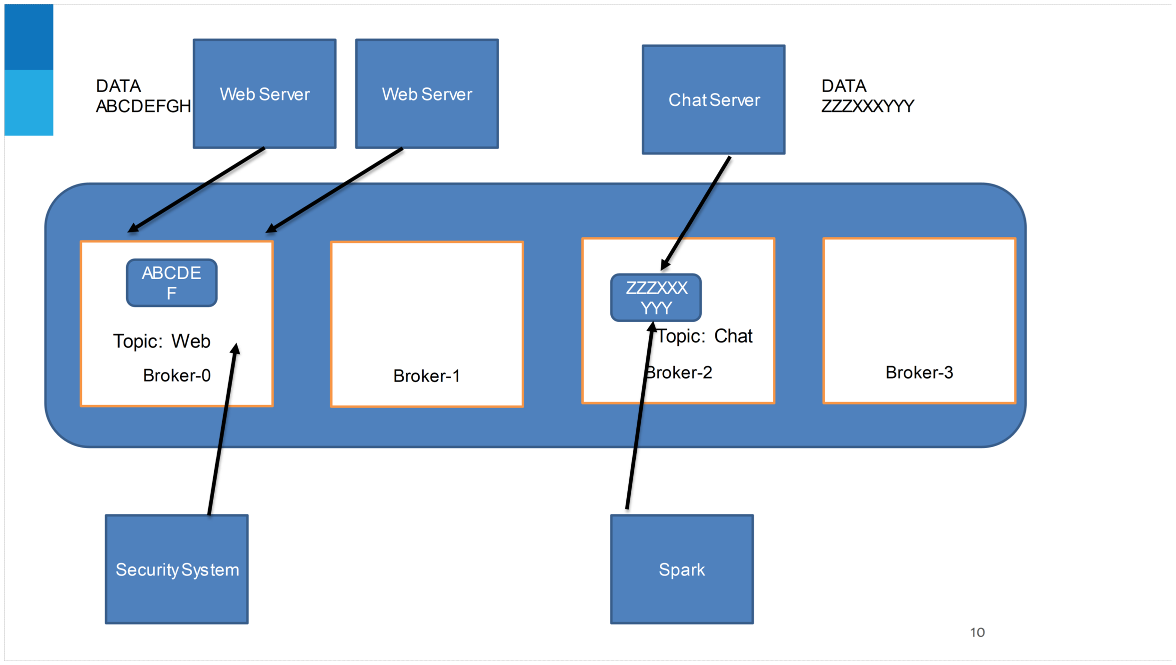
Message:

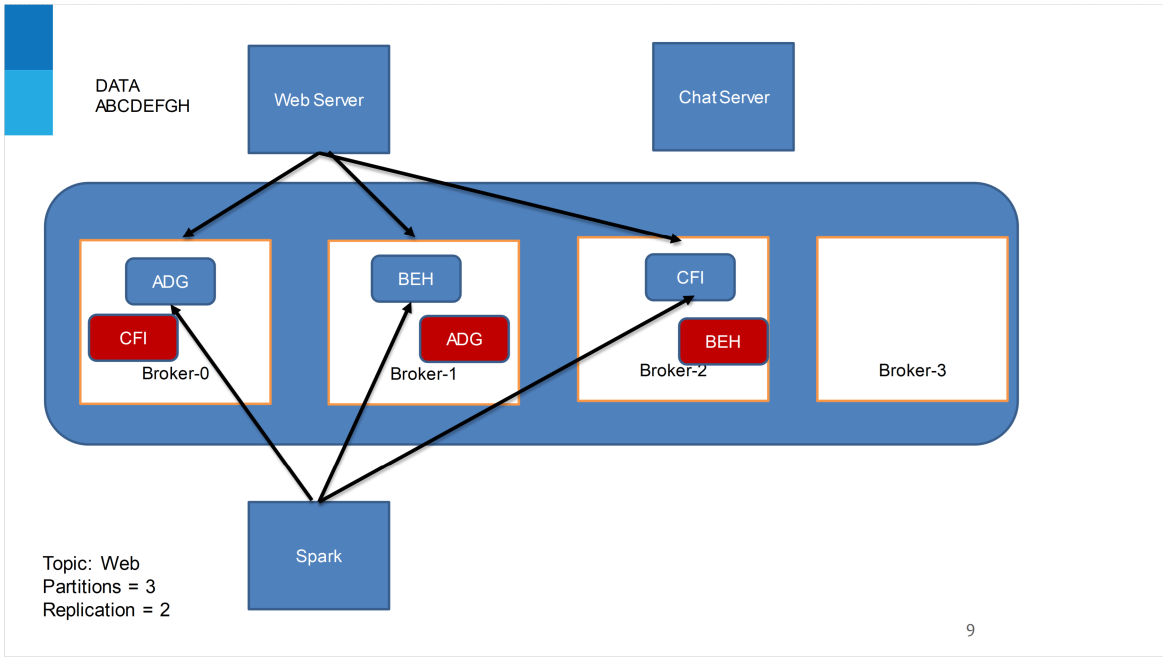
- Messages are simply byte arrays, and developers can store

any object in any format they choose.
- formats include String, JSON, Avro, and many more.

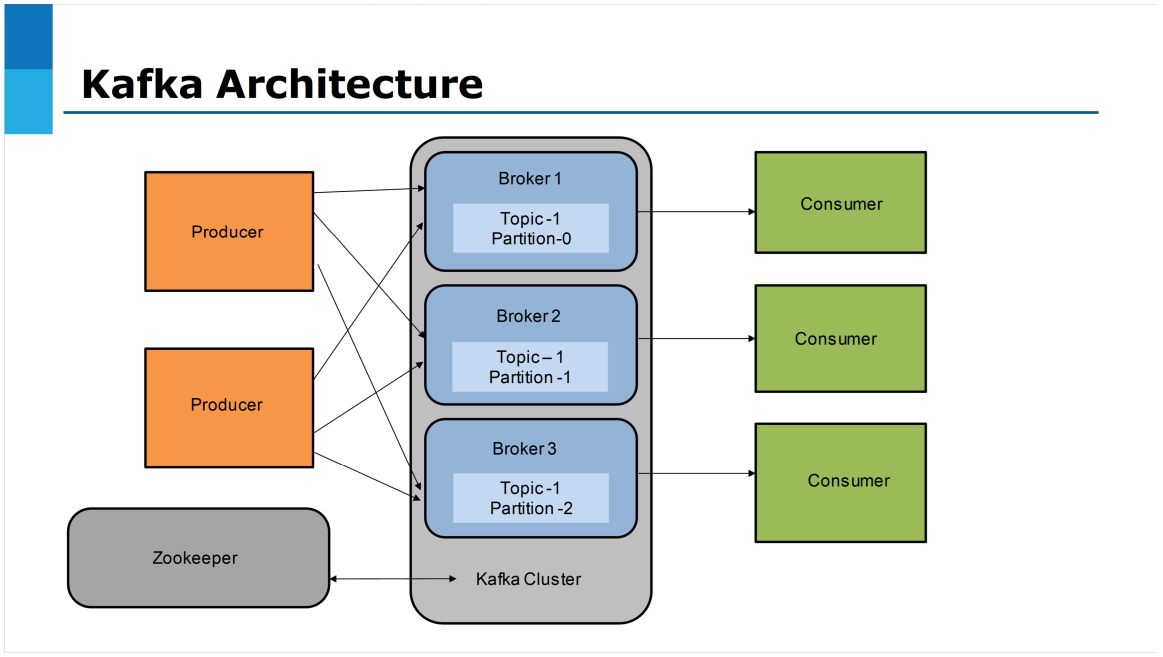
Cluster:

- Apache Kafka clusters are made up of multiple brokers, i.e., a set of servers.
- Also, it consists of multiple computers, each of which has one Kafka broker.





- Red blocks are backup messages.
- Here, broker-0 is leader for ADG. Means it has primary copy of ADG.
- And broker-1 is backup for ADG, because it has backup copy of ADG.
- Every time primary copy is sent to the receiver.



Kafka Architecture

Zookeeper is a co-ordination system.

It keeps the track of brokers' availability.

If any broker is down, zookeeper will know that.

Zookeeper is installed in the cluster.

We'll always see zookeeper in the cluster of odd no. of machines, Because it has to achieve quorum.

Quorum means majority votes.

Why Zookeeper runs on an odd number of machines

Zookeeper uses a consensus protocol where decisions are made if a majority (quorum) of nodes agree.
This majority rule ensures consistency and prevents split-brain.

Key idea:
- A quorum = more than half of the nodes
- Only the quorum can elect a leader and make updates

Example:
- 3 nodes → quorum is 2
- 5 nodes → quorum is 3
- 7 nodes → quorum is 4

Why odd number?
- With an odd count, the quorum is always a clear majority.
- With an even number, you waste one server without improving fault tolerance.

Example of inefficiency:
- 4 nodes → quorum is 3
  Failure tolerance: still only 1 node can fail
- Compare to 3 nodes → quorum is 2
  Failure tolerance: still 1 node can fail
  → so adding the 4th node gives no benefit.

Better to have 5 nodes instead:
- 5 nodes → quorum 3
- Now you can tolerate 2 node failures.

Summary:
- Zookeeper needs majority consensus.
- Odd node count guarantees clear majority.
- Even counts don’t improve fault tolerance and are inefficient.

Go to getting started with kafka, rich text format file.
Install those things.