**Apache Kafka**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**1. Need for Apache Kafka**

* When the focus was moving towards Data Analytics, we required a framework that can deal with 3 V’s i.e.

1. Volume of data
2. Velocity at which data can be processed
3. Variety of data

* RDBMS is believed to be the most effective tool to store, manage, manipulate & query the data so even today there is no replacement for RDBMS.
* The major flaw with RDBMS is to deal with 3 V’s. RDBMS has a lot of restrictions

1. Capable of dealing only with a few GBs of data.
2. Can deal with only normalized data i.e., data should be structured.
3. Centralized in nature.
4. Only support for vertical scaling (adding power/RAM/Processors) not horizontal scaling (adding nodes)

|  |  |  |
| --- | --- | --- |
| **No.** | **Horizontal Scaling (Scale In/Out/ Across)** | **Vertical Scaling (Scale up/ down)** |
| **1.** | It can be achieved by increasing / decreasing the number of nodes in a cluster to handle an increase or decrease in workload. | It can be achieved by increasing / decreasing the power of a system to handle an increase or decrease in workload. |
| **2.** | E.g., Add / Reduce the no. of VM in a cluster of VMs. | E.g., Add / Reduce the CPU or memory capacity of the existing VM |
| **3.** | Workload is distributed across multiple nodes. Here multiple requests get distributed across multiple machines over the network. | A single node handles the entire workload. It relies on multi – threading on the existing machine to handle multiple requests at the same time. |
| **4.** | High Performance | Low Performance |
| **5.** | There is no downtime because other machines in the cluster offers backup. | High downtime chance since it’s a single source of failure. |
| **6.** | Load balancing is required here to actively distribute workload across the multiple nodes. | Loading balancing not required in the single node. |

* These few limitations brought the need to develop a framework / Architecture that can help to achieve 3 V’s

|  |
| --- |
| * In between 2000 – 2002, Google started research called Google File System (GFS) for driving their business on the data they were receiving (Data Analytics). It internally used 2 important concepts  1. Distributed File System 2. Distributed Processing Engine  * In 2002, Google published a white paper about the concept of distributed framework where they had implemented distributed store & processing engine. * This white paper was picked by a developer named Daug cutting. He developed a framework called HADOOP to manage all the 3 V’s  1. For implementing Distributed Storage (DS), he used Hadoop Distributed File System (HDFS) 2. For implementing Distributed Processing (DP), he used Map Reduce Batch Processing System  * Using HADOOP, it was proved that Distributed way of storing & processing data achieved Massive parallel processing that is far more efficient than the traditional approach (RDBMS). * To overcome the problem of RDBMS vertical scaling, the idea is to have a cluster – based environment where multiple systems are connected over a common network rather having one high end database server. They decided to go with commodity H/W machines that can host this HADOOP framework & can work in a collaborative fashion as cluster – based computing. * **HyperScaler** – Seamlessly extending / downsizing the capacity of a cluster based on the requirement. * So, the main intention behind developing the HADOOP framework was to support OLAP (Online Analytical Processing) kind of transaction not for OLTP (Online Transaction Processing). * Drawbacks of HADOOP  1. The major bottleneck is to do Full – Fledged Map Reduce (MR) coding on the large volume of data stored. 2. A lot of disk input – output operations were involved while writing MR code i.e., the HADOOP framework is designed in such a way that all intermediate results will be buffered on the physical disk & the subsequent interfaces will take it from physical disks & they will further reduce it to the final result. 3. HADOOP framework was specifically designed for batch processing, not for Real – time Data processing.  * Different firms came up the ways to configure interface tools with Distributed Storage & Distributed processing without going in MR coding.  1. **Yahoo** came up with a scripting tool called **PIG** to achieve some kind of ETL things & configure it to HADOOP (Extract, transform, load). This PIG engine will internally reduce script to MR coding. 2. **Facebook** came out with HIVE. It gives an environment where we can use SQL (HQL – High QL) & write any type of complex analytical query to trigger it on a huge volume of data stored in this architecture. So, this abstracts us from writing the Java/Python/R code just write a query.  * At the end of 2008, HADOOP reached its saturation point in terms of business, clients & end user’s expectations. After 2008, the focus started moving towards an even more faster Processing framework irrespective of whether it will do OLAP or OLTP transactions. |

|  |
| --- |
| * Keeping the drawbacks of HADOOP framework in mind, Apache Spark was developed around 2008, the need for real – time data processing (process data dynamically) came into the picture. * Apache Spark is a lightning – fast distributed data processing engine. * Batch Processing: Process all the data at once. (Data persisted over a period of time pushed to cluster & process it) * 2 aspects were only considered while developing Spark:  1. Completely developed as In-Memory tool i.e., all the data in processing state should be kept on RAM. 2. Support for Batch processing as well as Real – time Data processing.  * Apache Spark has the capability of capturing the data in real-time or near real-time & then respond to the request. **E.g.,** Report Generation for the count of Tweets every 10 minutes. * Limitations of Apache Spark  1. It doesn’t come with its own storage. So, real – time data processing is fine but how batch processing will be done. So, data needs to be stored at NFS (Network file System) **or** integrated with HDFS (HADOOP Distributed File System) **or** pull the data from RDBMS **or** NoSQL – DB, **or** AWS – S3 2. Spark framework doesn’t have the capacity to keep track of what was the data flown when a particular node is down & where should it continue because there is no mediator to manage. 3. Spark framework doesn’t have the capacity to keep on gathering data for a particular interval of time because in this case input & output buffer will be huge & there are likely chances that RAM might overflow.  * To overcome the limitation of Apache Spark, there is a necessity of some mediator in between i.e., A Message Broker so that this broker can take the responsibility of reading the data streams, buffering it for a particular duration, dealing with all fault tolerance & in case some nodes go down, from which offset to continue. |

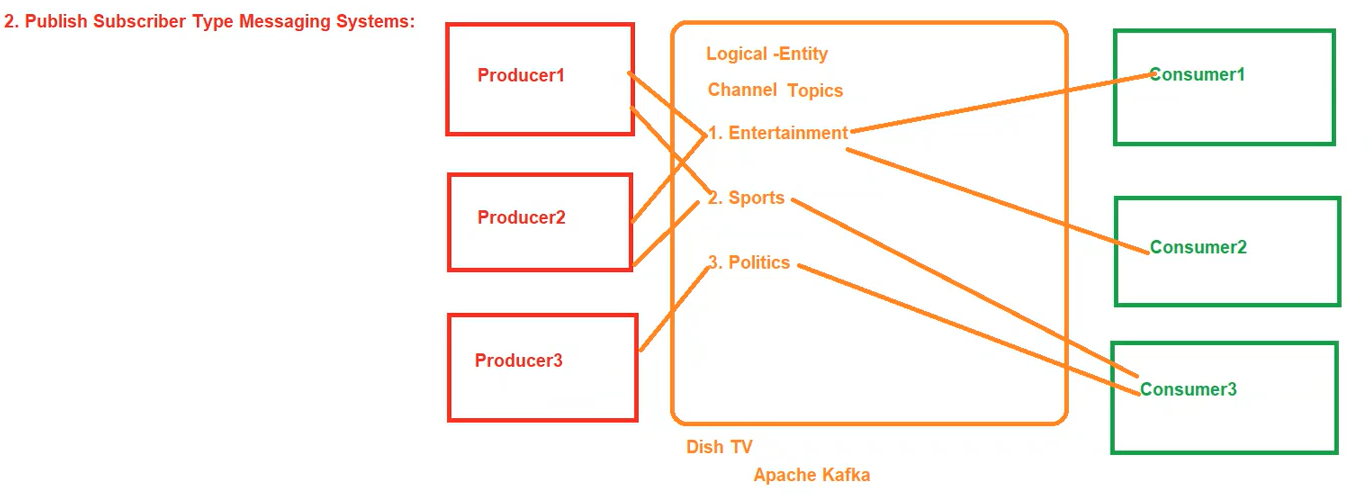
|  |
| --- |
| * Initially Kafka was developed majorly with above intensions. * At the end of 2011, Kafka open – sourced to Apache foundation & lately developers who previously worked on Kafka left LinkedIn & built a new firm called Confluent in 2014. Confluent developers added more components / functionality to the framework along with Kafka to release it as an Enterprise Kafka version & they launched it around 2015 & gained popularity around 2018. * The only aspect we look at Kafka is to act as a Message Broker as long as it was Apache Kafka. On the other end, the consumer use to be mostly spark streaming applications. * Now Kafka ecosystem can itself expand & can accommodate the processing part as well. As Kafka evolves, it might take over Spark. |

**2. Message Brokers – (Point to Point, Publish – Subscriber)**



**a) Point to Point Messaging System (P2P)**

* In this messaging system, each message produced by the producer will be intended for a specific consumer & once the message is consumed it will no longer be available on the queue i.e., Consumer c1 will receive only those messages specified only by c1.
* E.g., Orders placed on Zomato, Swiggy will be specifically for a particular consumer & once you consume it, it will no longer exist in the queue.



**b) Publish Subscriber Messaging System**

* In this messaging system, producers are publishing their messages to the specific entity/identifier & the consumer can subscribe to any of these entities/channel to receive the data.
* Here, each producer can produce to multiple consumers & each consumer can consume data from multiple producers.
* E.g., Dish TV where multiple consumers can subscribe to multiple packages. That way subscribers to a particular package will receive all channels in that particular package.
* Kafka falls under Publish subscriber messaging system.
* In terms of Kafka, we call this logical entity as a Topic. So, the topic is the one thing that will relate the respective producer to the respective consumer.

**3. Apache Kafka Intro**

* Kafka is an opensource distributed publish – subscriber messaging system. It is fast, durable, scalable & fault tolerant.
* It acts as a stream processing platform using KStream library & can connect to external system for data import/export by Kafka Connect.
* It’s suitable for handling large – scale streaming data.
* The design is highly influenced by Transaction logs.

**4. Use cases of Apache Kafka**

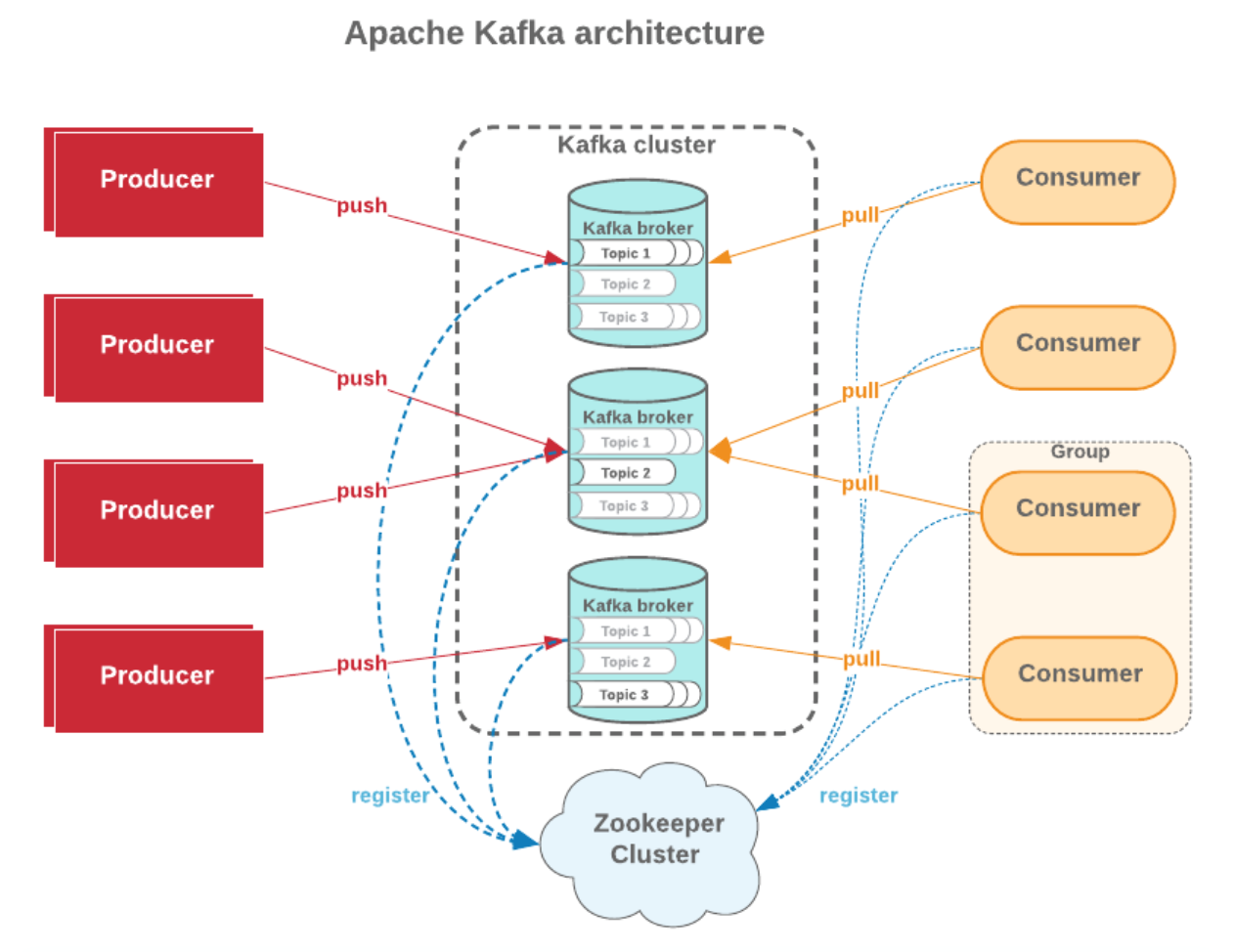
* Building real – time streaming data pipeline
* Building real – time streaming application
* CDC (Change Data Capture)

**5. Key features of Apache Camel**

1. Scalability f) Real – time stream processing
2. Durability g) Data Retention policies
3. Fault tolerant h) Dynamic Reconfiguration
4. Partitioned i) Community & Support
5. High Throughput

**6. Apache Kafka Architecture**

* In this architecture, we have Producer Group (containing n number of producers), consumer Group (containing n number of consumers) & in between comes Kafka ecosystem.



* This Kafka ecosystem primarily consists of:

1. **Kafka Cluster** – Primary component of kafka ecosystem
2. **Kafka brokers/Servers/Nodes** – Cluster composed of n number of brokers.
3. **Zookeeper** – Cluster coordinator service. It helps in interaction & coordination b/w producer group, consumer group & kafka ecosystem.
4. **Topic** – Topics are logical channels or categories to which messages are published.

**Apache Kafka Terminology**

**1. Producer** – A producer can be any application who can publish message to a topic.

**2. Consumer** – A consumer can be any application that subscribes to a topic & consuming the messages. A consumer can subscribe to multiple topics.

**3. Consumer Group**

* Multiple consumers can be grouped under one label called Consumer Group.
* Each record published to a topic is delivered to one consumer instance within the subscribed consumer group.
* Each consumer group processes a subset of partition, allowing for parallel processing & load distribution.

**4. Zookeeper**

* It is used for managing & coordinating kafka cluster.
* It maintains metadata, leader election & manages the overall state of the cluster.

**5. Kafka Broker**

* Kafka brokers are individual servers / machines within the kafka cluster.
* They store & manage data, handle producer & consumer requests & participate in the replication & distribution of data.

**6. Topics**

* Topics are logical channels or categories to which messages are published.
* Topics can be divided into partitions for scalability & parallelism.
* Each record consists of Key, Value & timestamp.

**7. Partition**

* Topics are broken into order commit logs called Partitions. They are the basic unit of Parallelism.
* Each partition can be placed on other brokers to allow multiple consumers to read it parallelly.
* Partitioning is the concept of segregating data into subsets & increasing the parallelism of writing & reading the data. On the other hand, if one of the brokers goes down, we need more copies of that topic.
* So, we need to define while creating a topic is how many replications should happen for that particular topic i.e., Replication Factor.

Note: (Ideal cases) [N – No. of nodes/brokers & P – No. of Partitions]

Case 1: If N == P for Topic T1 then each node will contain 1 Partition.

Case 2: If N > P for Topic T1 then each N nodes will have 1 Partition & other (N – P) nodes will be empty.

Case 3: If N < P for Topic T1 then there may be a possibility that Nodes will have more than 1 partition.

**8. Offset Management**

* Offset represents the position of a consumer within a partition.
* Consumers commit offsets to Kafka for tracking their progress. This ensures that they can resume processing from the last committed offset in case of failure or restart.

**9. Topic Partition Replication**

* Each partition has multiple replicas for fault tolerance. Replicas are distributed across different brokers.
* One replica is designated as the Leader & others are followers.

**10. Log Compaction**

* Kafka supports log compaction i.e., retaining only the latest message for each key in a partition.
* This is useful for scenarios where maintaining the latest state for a set of keys is critical.

**11. Kafka Connect**

* Kafka connect is a framework for integrating Kafka with external system.
* It simplifies the development of connectors for ingesting data from or delivering data to various sources & sinks.

**12. Kafka Streams**

* Kafka Stream is a stream processing library that allows developers to build real – time applications & microservices using kafka as underlying infrastructure.

**High level Architecture of Kafka**

* On high level overview, there is a cluster containing multiple brokers where one is leader & rests are followers having replica of data.
* Now the leader broker hosts a leader partition where the producer writes & consumer reads.
* Rest broker hosts follower partitions & hence will replicate the same data to different disk location for fault tolerance.

**Architecture Flow**

|  |
| --- |
| **1. Producer writes data to a Leader** i.e., Producer sends message to the leader partition of the specified topic. The leader appends messages to its local log. |

|  |
| --- |
| **2. Replica synchronization** i.e., Replication of data is always done at partition level. Leader replicates messages to followers, ensuring they have the same set of messages.  ISR (In – Sync Replica) represents replicas that are up to date with the leader |

|  |
| --- |
| **3. Consumer reads from the Leader** i.e., consumers always read from the leader partition. Once read, it acknowledges the broker that it has read the message successfully & also commits the offset so that in case of failure, it can come back & read from the same offset (the leader ensures followers are kept in sync) |

|  |
| --- |
| **Fault Tolerance** – In the event of a leader or broken failure, kafka ensures quick leader election & data replication from in-sync replicas.  **Scalability** – Kafka scales horizontally by distributing partitions across multiple brokers.  **Zookeeper** coordinates leader election, maintains metadata & manages the overall state of the kafka cluster. |

**7. Role of Zookeeper in Kafka**

* It acts as a centralized & reliable coordination service for Kafka, ensuring that the distributed components of a Kafka cluster can work together seamlessly.
* It helps in managing the dynamic nature of kafka clusters, providing fault tolerance, & enabling coordinating among different components.
* For interaction b/w producers & brokers, Zookeeper helps the producer in term of Broker id i.e., to which broker producers are supposed to publish the message.
* Similarly, for interaction b/w brokers & consumers, Zookeeper helps the consumer in terms of Broker id & offset of the data.
* Cluster Coordination – Leader Election, Broker registration & management.
* Topic & Partition information
* Consumer Group management
* Broker & Topic health monitoring

Note: From Kafka 2.8.0, there have been efforts to reduce Kafka’s dependency on Zookeeper, with ongoing work to replace it with a self – managed metadata store.