Apache Kafka

Dhruvil Dave - AU1841003
Supan Shah - AU2044011

What is Kafka

Apache Kafka is an open source distributed event streaming platform used by thousands of companies for high performance data pipelines, streaming analytics, data integration and mission critical applications.

Who uses Kafka

- 1/3rd of all Fortune 500 companies
- LinkedIn: Activity data and operational metrics
- Twitter: Uses it as part of Storm – stream processing infrastructure
- Spotify, Uber, Tumbler, Goldman Sachs, PayPal, Box, Cisco, CloudFlare, DataDog, NetFlix, etc

Why Kafka is needed?

- Real time streaming data processed for real time analytics
- Apache Kafka is a fast, scalable, durable, and fault tolerant publish-subscribe messaging system
- Higher throughput, reliability and replication
- Feed your data lakes with data streams

Kafka use-cases

- Works in combination with HBase and Spark for real time analysis and processing of streaming data.
- Kafka brokers support massive message streams for follow up analysis in Hadoop or Spark
- Kafka Streaming (subproject) can be used for real-time analytics

Kafka use-cases

- Stream Processing
- Website Activity tracking
- Metrics Collection and Monitoring
- Log Aggregation
- CRQS, replay, error recovery
- Guaranteed distributed commit log for in-memory computing

Why is Kafka popular?

- Great performance
- Operational Simplicity, easy to setup and use, easy to reason
- Robust Replication
- Flexible Publish-subscribe/queue (scales with N-number of consumer groups)
- Producer Tunable Consistency Guarantees
- Ordering Preserved at shard level (Topic Partition)
- Works well with systems that have data streams to process, aggregate, transform & load into other stores

Why is Kafka so fast?

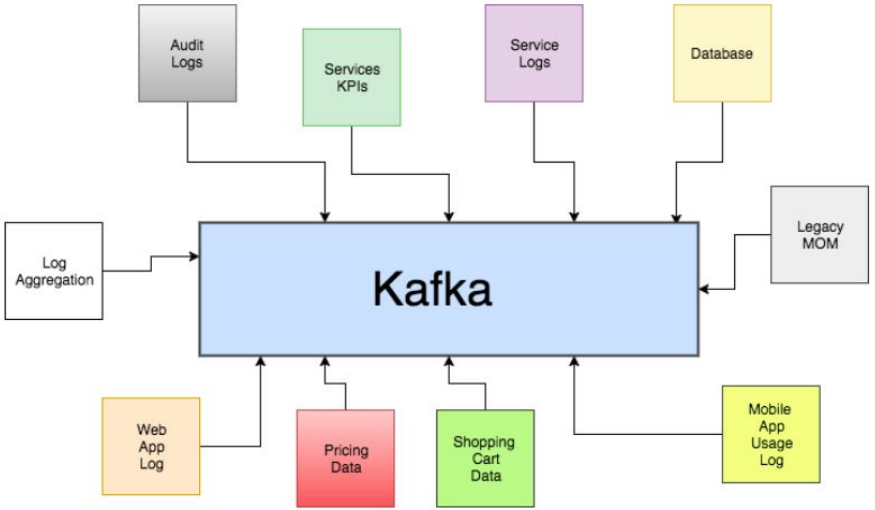
- Zero Copy - calls the OS kernel direct rather to move data fast
- Batch Data in Chunks - Batches data into chunks which reduces I/O latency
- Sequential Disk Writes - Avoids Random Disk Access
- Horizontal Scale - uses 100s to thousands of partitions for a single topic

Streaming Architecture

Kafka gets used most often for real-time streaming of data into other systems. Kafka is a middle layer to decouple your real-time data pipelines. Kafka core is not good for direct computations such as data aggregations, or CEP. Kafka Streaming which is part of the Kafka ecosystem does provide the ability to do real-time analytics. Kafka can be used to feed fast lane systems (real-time, and operational data systems) like Storm, Flink, Spark Streaming and your services and CEP systems. Kafka is also used to stream data for batch data analysis. Kafka feeds Hadoop. It streams data into your Big Data platform or into RDBMS, Cassandra, Spark, or even S3 for some future data analysis. These data stores often support data analysis, reporting, data science crunching, compliance auditing, and backups.

Kafka decoupling data streams

Kafka decouple streams of data by allowing multiple consumer groups that can each control where in the topic partition they are. The producers don’t know about the consumers. Since the Kafka broker delegates the log partition offset (where the consumer is in the record stream) to the clients (Consumers), the message consumption is flexible. This allows you to feed your high-latency daily or hourly data analysis in Spark and Hadoop and the same time you are feeding microservices real-time messages, sending events to your CEP system and feeding data to your real-time analytic systems.



Wire Protocol

Kafka communication from clients and servers uses a wire protocol over TCP that is versioned and documented. Kafka promises to maintain backward compatibility with older clients, and many languages are supported. The Kafka ecosystem also provides REST proxy allows easy integration via HTTP and JSON. Kafka also supports Avro via the Confluent Schema Registry for Kafka. Avro and the Schema Registry allows complex records to be produced and read by clients in many programming languages and allows for the evolution of the records. Kafka is truly polyglot.

Kafka Record Retention

Kafka cluster retains all published records and if you don't set a limit, it will keep records until it runs out of disk space. You can set time-based limits (configurable retention period), size-based limits (configurable based on size), or use compaction (keeps the latest version of record using key). You can, for example, set a retention policy of three days or two weeks or a month. The records in the topic log are available for consumption until discarded by time, size or compaction. The consumption speed not impacted by size as Kafka always writes to the end of the topic log.

- Kafka cluster retains all published records
- Retention policy of three days or two weeks or a month
- It is available for consumption until discarded by time, size or compaction
- Consumption speed not impacted by size

Kafka Scalable Message Storage

Kafka is a good storage system for records/messages. Kafka acts like high-speed file system for commit log storage and replication. These characteristics make Kafka useful for all manners of applications. Records written to Kafka topics are persisted to disk and replicated to other servers for fault-tolerance. Since modern drives are fast and quite large, this fits well and is very useful. Kafka Producers can wait on acknowledgment, so messages are durable as the producer write not complete until the message replicates. The Kafka disk structure scales well. Modern disk drives have very high throughput when writing in large streaming batches. Also, Kafka Clients/ Consumers can control read position (offset) which allows for use cases like replaying the log if there was a critical bug (fix the bug and the replay). And since offsets are tracked per consumer group, the consumers can be quite flexible (like replaying the log).

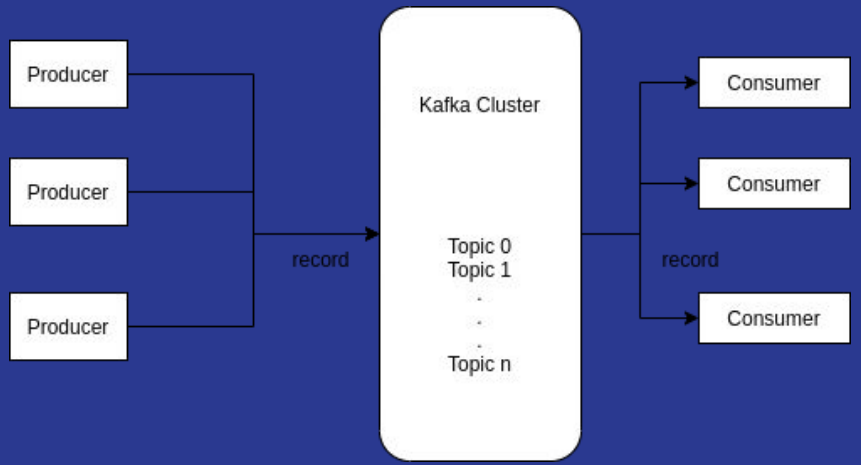
- Kafka acts as a good storage system for records/messages
- Records written to Kafka topics are persisted to disk and replicated to other servers for fault-tolerance
- Kafka Producers can wait on acknowledgment
- Kafka disk structures scales well as writing large batches is fast
- Clients/Consumers can control read position

Kafka Fundamentals

- Records have a key (optional), value and timestamp; Immutable
- Topic a stream of records (“/orders”, “/user-signups”), feed name
  - Log topic storage on disk
  - Partition / Segments (parts of Topic Log)
- Producer API to produce a streams or records
- Consumer API to consume a stream of records
- Broker: Kafka server that runs in a Kafka Cluster. Brokers form a cluster.
- ZooKeeper: Does coordination of brokers/cluster topology.

Kafka: Topics, Producers, and Consumers

This shows Producers sending records to a Kafka Topic and Consumers consuming records from the Kafka cluster.



Kafka and Zookeeper

- Zookeeper helps with leadership election of Kafka Broker and Topic Partition pairs
- Zookeeper manages service discovery for Kafka Brokers that form the cluster
- Zookeeper sends changes to Kafka
  - New Broker join, Broker died, etc
  - Topic removed, Topic added, etc.
- Zookeeper provides in-sync view of Kafka Cluster configuration

Kafka Producer/Consumer details

Producers write at their cadence so the order of Records cannot be guaranteed across partitions. The producers get to configure their consistency/durability level (ack=0, ack=all, ack=1). Producers pick the partition such that Record/messages go to a given partition based on the data. Consumers are grouped into “Consumer Groups”. A consumer group has a unique id. Each consumer group is a subscriber to one or more Kafka topics. Each consumer group maintains its offset per topic partition. If you need multiple subscribers, then you have multiple consumer groups. A record gets delivered to only one consumer in a consumer group.

- Producers write to and Consumers read from Topic(s)
- Topic associated with a log which is data structure on disk
- Producer(s) append Records at end of Topic log
- Topic Log consist of Partitions and spread multiple files to multiple nodes
- Consumers read from Kafka at their own cadence
    - Each Consumer (Consumer Group) tracks offset from where they left off reading
- Partitions can be distributed on different machines in a cluster
    - High performance with horizontal scalability and failover with replication

Kafka Speed and Scale

First Kafka is fast, Kafka writes to filesystem sequentially which is fast. On a modern fast drive, Kafka can easily write up to 700 MB or more bytes of data a second. Kafka scales writes and reads by sharding topic logs into partitions (parts of a Topic log). Recall Topics logs can be split into multiple partitions which can be stored on multiple different servers, and those servers can use multiple disks. Multiple producers can write to different partitions of the same Topic. Multiple consumers from multiple consumer groups can read from different partitions efficiently.

Kafka Brokers

- Kafka Cluster is made up of multiple Kafka Brokers
- Each Broker has an ID (number)
- Brokers contain topic log partitions
- Connecting to one broker bootstraps client to entire cluster
- Start with at least three brokers, cluster can have, 10, 100, 1000 brokers if needed

Kafka Clusters, Failover, In Sync Replicas (ISR)

- Topic Partitions can be replicated across multiple nodes for failover
- Topic should have a replication factor greater than 1
- Failover - if one Kafka Broker goes down then Kafka Broker with ISR (in-sync replica) can serve data

Failover and Disaster Recovery

Kafka uses replication for failover. Replication of Kafka topic log partitions allows for failure of a rack or AWS availability zone (AZ). You need a replication factor of at least 3 to survive a single AZ failure. You need to use Mirror Maker, a Kafka utility that ships with Kafka core, for disaster recovery. Mirror Maker replicates a Kafka cluster to another data-center or AWS region. They call what Mirror Maker does mirroring as not to be confused with replication.

Kafka vs AWS Kinesis

- Data is stored in Kinesis for default 24 hours, and you can increase that up to 7 days
- Kafka records default stored for 7 days
- With Kinesis data can be analyzed by lambda before it gets sent to S3 or RedShift
- With Kinesis you pay for use, by buying read and write units.
- Kafka is more flexible than Kinesis but you have to manage your own clusters, and requires some dedicated DevOps resources to keep it going
- Kinesis is sold as a service and does not require a DevOps team to keep it going (can be more expensive and less flexible, but much easier to setup and run)

Thank You

Dhruvil Dave - AU1841003
Supan Shah - AU2044011