Kafka

References

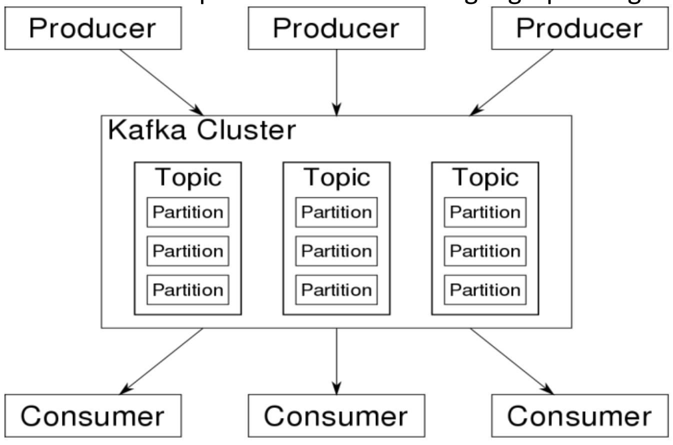
- https://developer.confluent.io/learn-kafka/
- https://www.youtube.com/watch?v=PzPXRmVHMxI
- https://jack-vanlightly.com/blog/2018/9/2/rabbitmq-vs-kafka-part-6-fault-tolerance-and-high-availability-with-kafka

What is it

- Open-source **distributed event streaming platform** used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications
- A framework implementation of Software bus using stream processing
- Platform used to collect, process, store, and integrate data at scale
- Kafka is based on the **abstraction of a distributed commit log**
- Decoupling of data streams and systems
- Used as a transportation mechanism - Really good at making your data move really fast and at scale

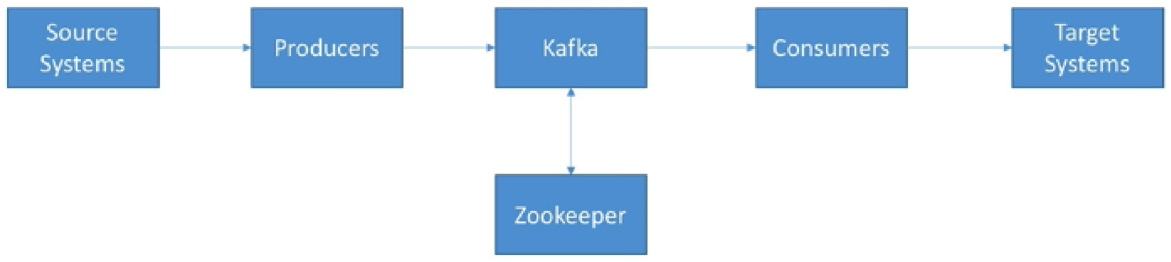
Core capabilities

- **High throughput** - Deliver messages at network limited throughput using a cluster of machines with latencies as low as 2ms
- **Scalable** - Scale production clusters up to a thousand brokers, trillions of messages per day, petabytes of data, hundreds of thousands of partitions. Elastically expand and contract storage and processing
- **Permanent Storage** - Store streams of data safely in a distributed, durable, fault-tolerant cluster
- **High availability** - Stretch clusters efficiently over availability zones or connect separate clusters across geographic regions

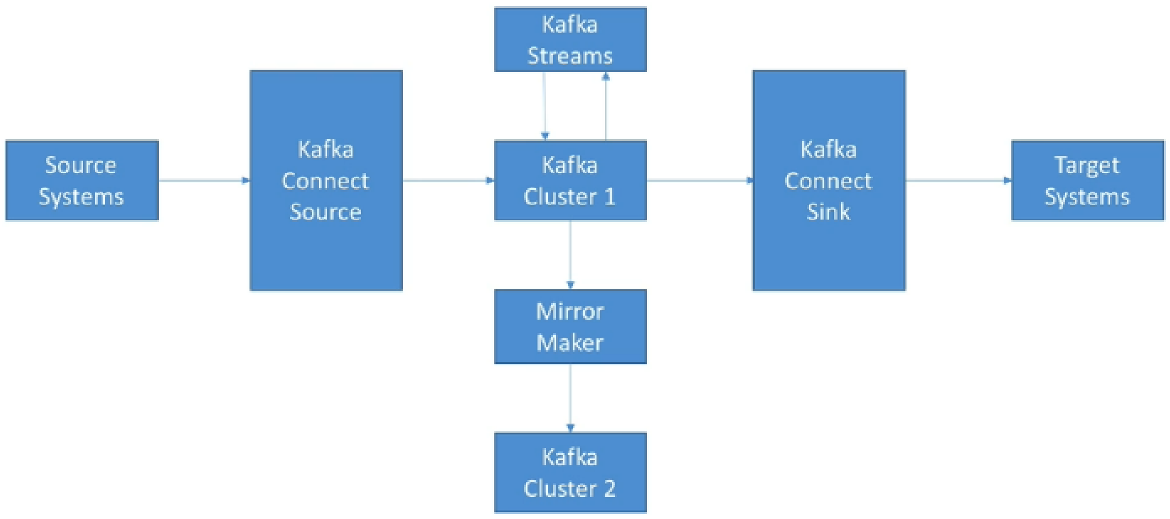


Kafka Core Ecosystem

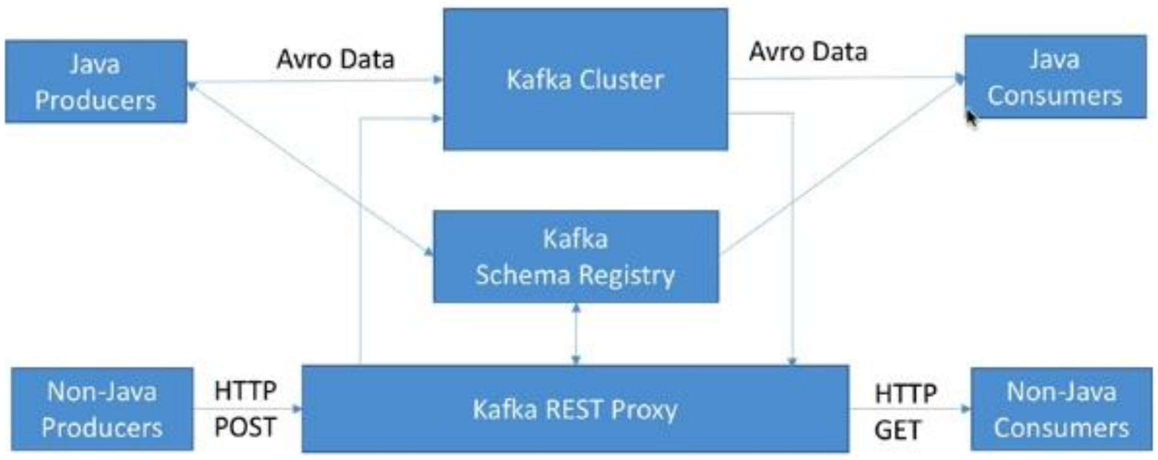
- BUILT-IN STREAM PROCESSING
- CONNECT TO ALMOST ANYTHING
- CLIENT LIBRARIES
- LARGE ECOSYSTEM OPEN SOURCE TOOLS



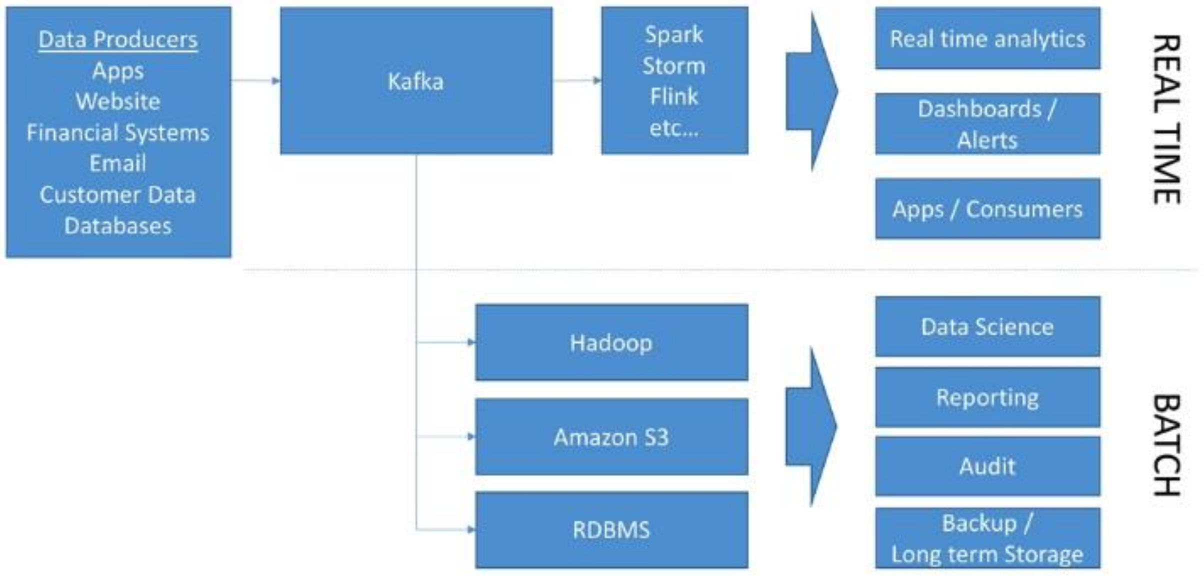
Kafka Extended API



Confluent Components Schema Registry and REST proxy

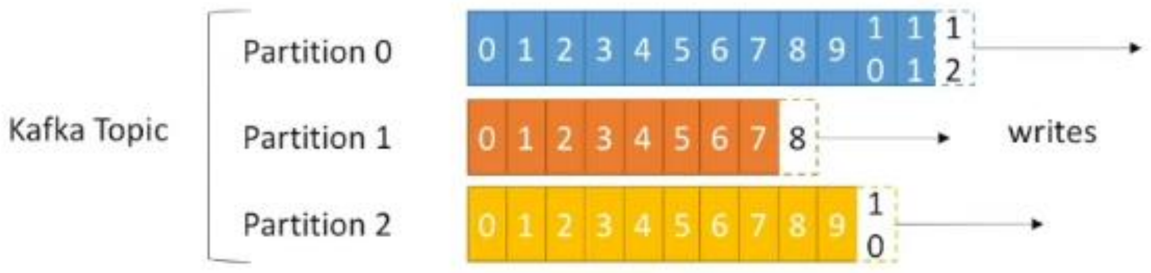


Kafka in the Enterprise Architecture



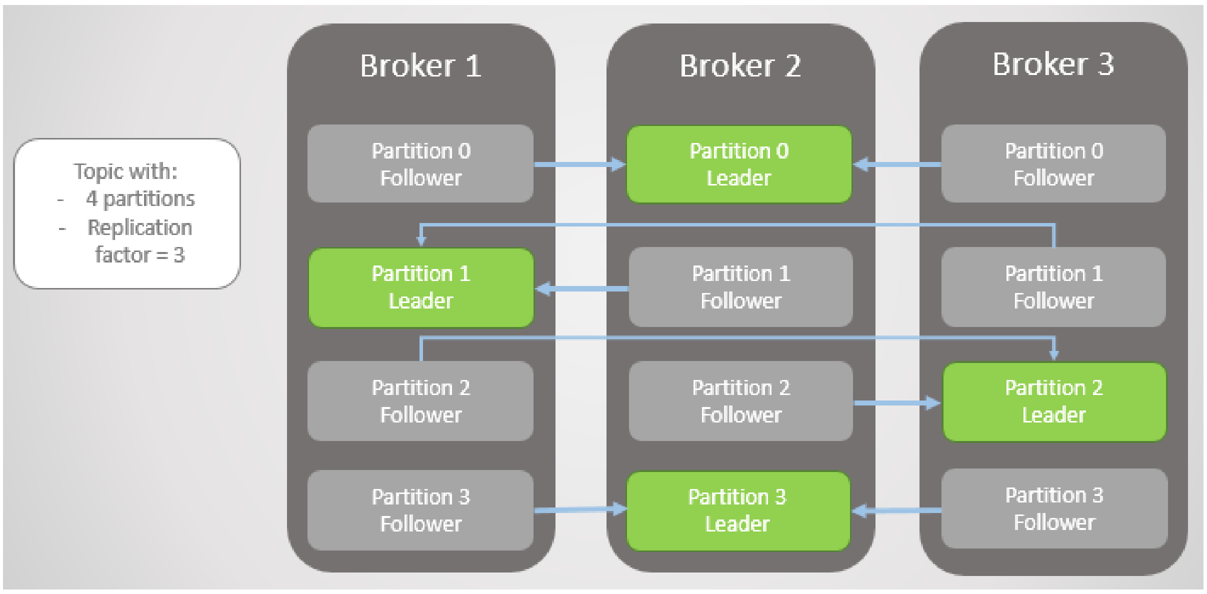
Topics, partitions, and offsets

- **Topic**
  - A particular stream of data - identified by its **name** - similar to a table in database (without all the constraints)
  - As many topics as you want. Ex - truck_gps
- **Partition**
  - Topics are split in **partitions**
  - **Data can be partitioned into diff partitions within diff topics**
  - You must choose no. of partitions at creation time of your topic
  - Each partition is ordered - partition 0, 1, 2...
  - Each **message** within a partition is strictly ordered (gets an incremental id), called **offset (the position of a message within a partition)**
- **Offset**
  - Only have a meaning for a specific partition
  - Order is guaranteed only within a partition (not across)
  - Immutability - Once the data is written to a partition, **it can't be changed**
  - Data is kept only for a limited time (default - one week)
  - Offsets keep on incrementing. They never go back to zero
  - Data is assigned in round-robin across partitions unless a key is provided



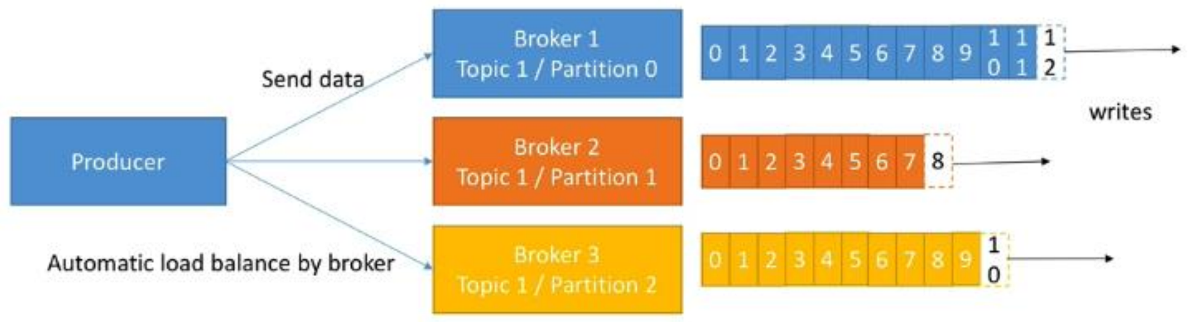
Brokers

- Kafka runs on a cluster of one or more servers (called brokers), and the partitions of all topics are distributed across the cluster nodes. Additionally, partitions are replicated to multiple brokers
- Each broker is identified with its ID (integer)
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- In a contemporary deployment, these may not be separate physical servers but containers running on pods running on virtualized servers running on actual processors in a physical datacenter somewhere
- **Leader of a partition**
    - At any time - only 1 broker can be a leader for a given partition
    - **Only that leader can receive and serve data for a partition**
    - The other brokers will synchronize the data
    - Thus - 1 leader and multiple ISR (in sync replica) also called **follower replica**
- **Topic replication factor** should be >1 (usually between 2 & 3) This way, if a broker is down, another broker can serve the data



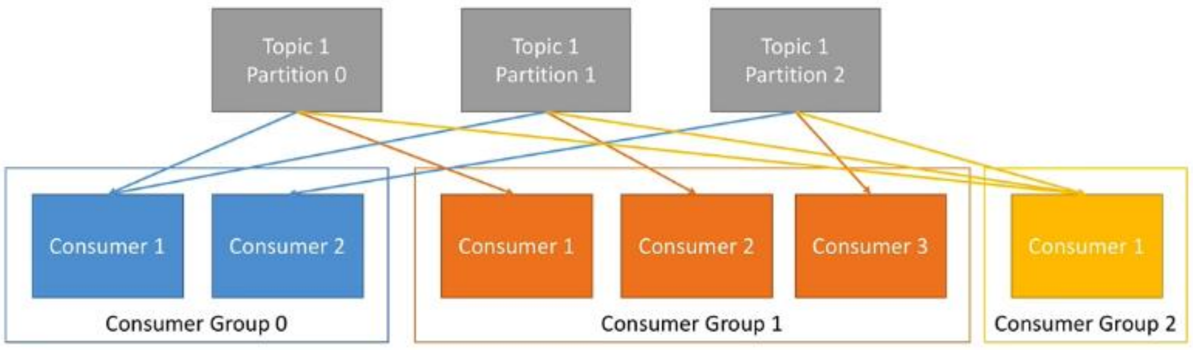
Producers

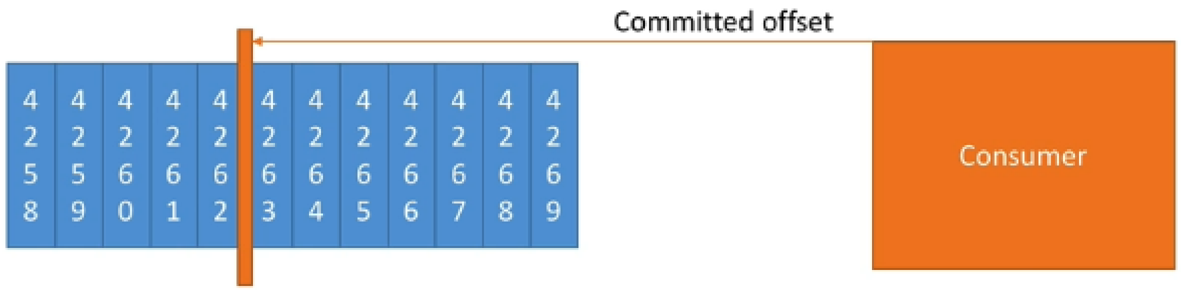
- Write data to topics
- Kafka will automatically take care of **routing the data to the right brokers**
- Producers can choose to receive ack of data writes
    - Acks=0 - Producer won't wait for ack (possible data loss)
    - Acks=1 - Producer will wait for leader ack (limited data loss)
    - Acks=all - Leader + replicas ack (no data loss)
- **Message Keys**
    - Producer can choose to send a key with the message
    - If a key is sent, then the producer has the guarantee that all messages for that key will always go to the same partition - Based on hashing the key value
    - Enables to guarantee ordering for a specific key



Consumers

- Read data from topics
- Kafka will automatically take care of **pulling the data from the right brokers**
- Data is read in order for each partitions
- Messages are always read in order within a partition but in parallel across partitions
- **Consumer Groups**
  - To enhance parallelism
  - Consumers read data in consumer groups
  - Each consumer within a group reads from exclusive partitions
  - You cannot have more consumers than partitions
- **Consumer offsets**
  - How consumers know where to read from - Consumer offsets
  - Kafka stores the offsets at which a consumer group has been reading
  - Offsets commit live in a Kafka topic named "__consumer_offsets"
  - When a consumer has processed data received - will be committing the offsets
  - If a consumer process dies, it will be able to read back from where it left off





Use cases

- Messaging system
- Activity tracking
- Gather metrics from many diff locations (such as IoT devices)
- Application logs gathering
- Stream processing (with Kafka Streams API or Spark)
- Decoupling of system dependencies

Applications

- Netflix - Apply recommendations in real time while you're watching shows
- Uber - Gather user, taxi, trip data in real time to compute and forecast demand, compute surge pricing in real time
- LinkedIn - Prevent spam, collect user interactions to make better connection recommendations