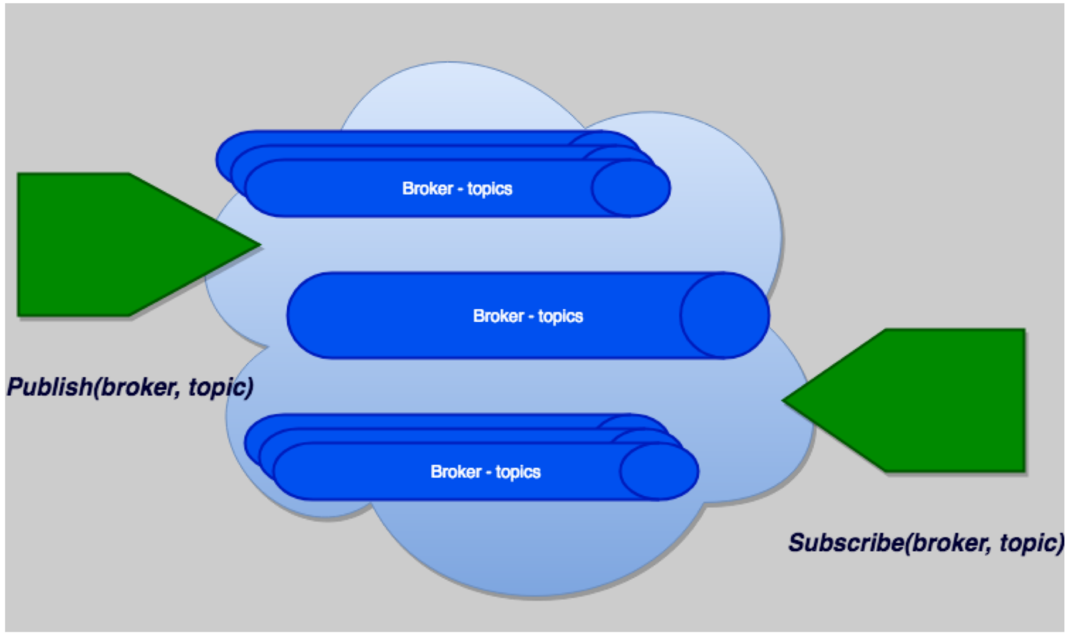
Publish/Subscribe - Kafka

Publish/Subscribe Abstractions

- Publish a topic to a broker – publish(broker, topic)
- Subscribe to a topic on a broker – subscribe(broker, topic)



Advantages

- Decouples producer and consumer. Consider a slow consumer.
- Many processes can subscribe to the data.
- It scales. Too much traffic? Start another broker.
- If consumer fails, replay the log with another machine.

Disadvantage

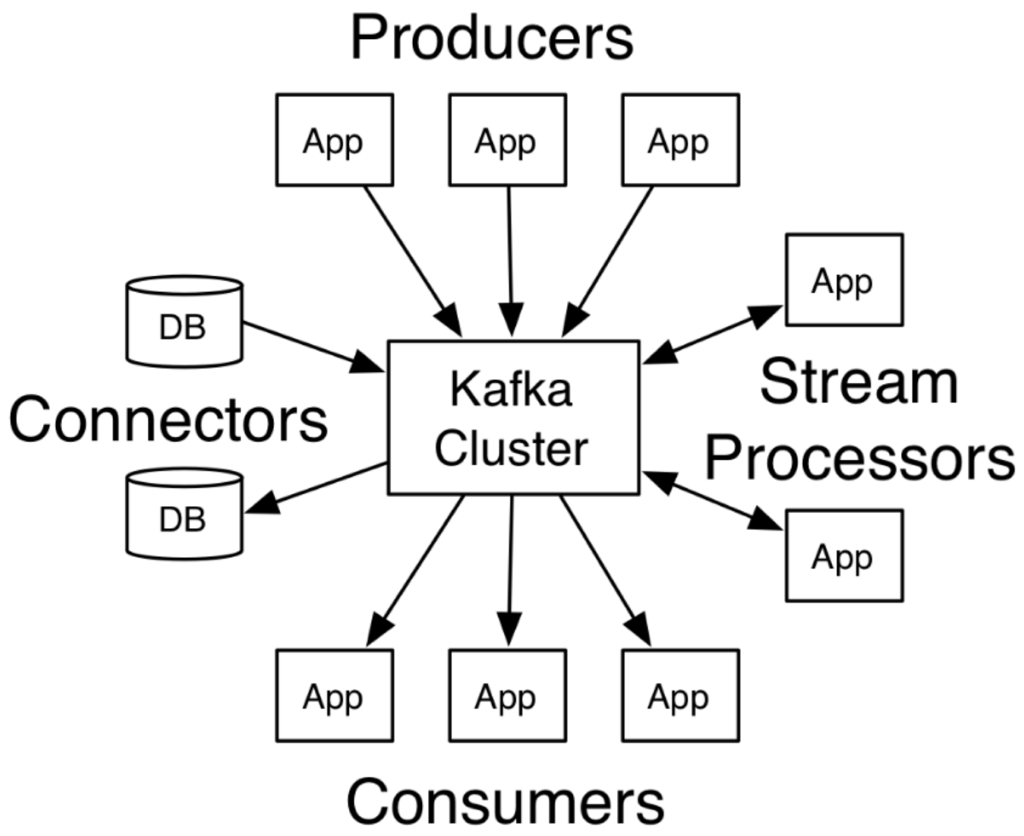
- Additional overhead
- Backpressure – messages pile up and no one consumes them

Distributed Streaming Platform

Apache Kafka® is *a distributed streaming platform that can:*

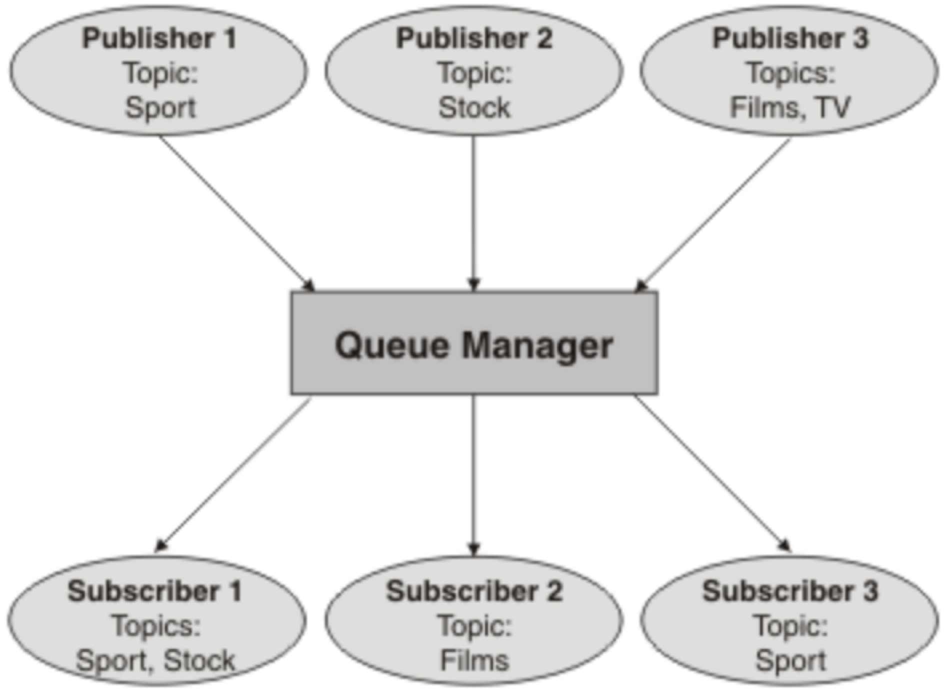
- ***Publish and subscribe*** to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in ***a fault-tolerant*** durable way.
- Process streams of records as they occur. Order of ***events*** is preserved.

Publish/Subscribe



Pub/Sub – [source – IBM]

- Publishers put messages on the queue
- Subscribers pull messages off the queue


Publisher 1 Topic: Sport
Publisher 2 Topic: Stock
Publisher 3 Topics: Films, TV
Queue Manager
Subscriber 1 Topics: Sport, Stock
Subscriber 2 Topic: Films
Subscriber 3 Topic: Sport

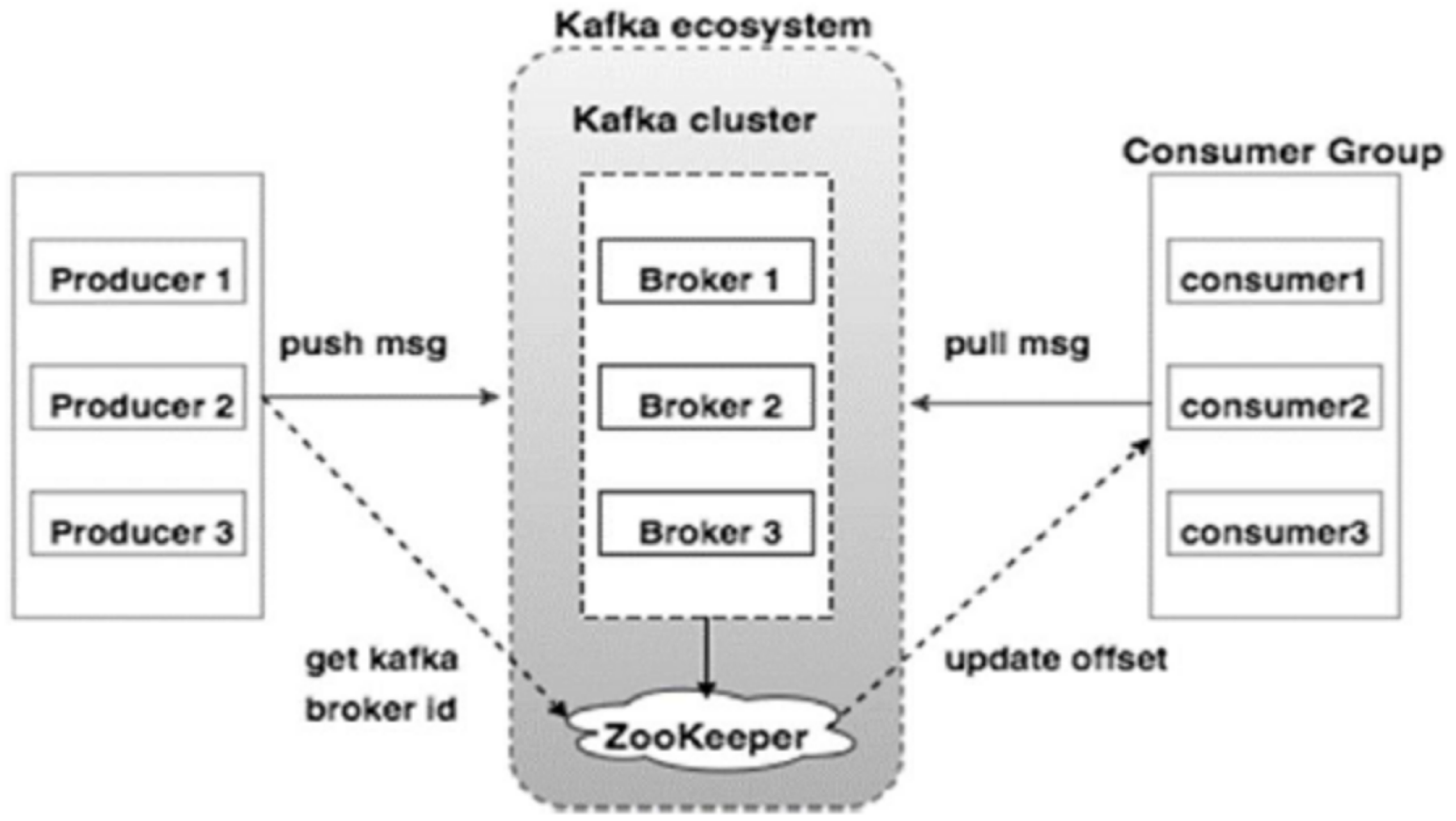
Use Cases

- Building ***real-time*** streaming data pipelines that ***reliably*** get data between systems or applications.
- Building real-time streaming applications that transform or react to the ***streams*** of data

Architecture Overview

- Kafka is run as a cluster on one or more servers (***brokers***) that can span multiple datacenters.
- The Kafka cluster stores streams of *records (**events**)* in categories called **topics**.
- Each record consists of a ***key***, a ***value***, and a ***timestamp***.

Architecture – Brokers [source]

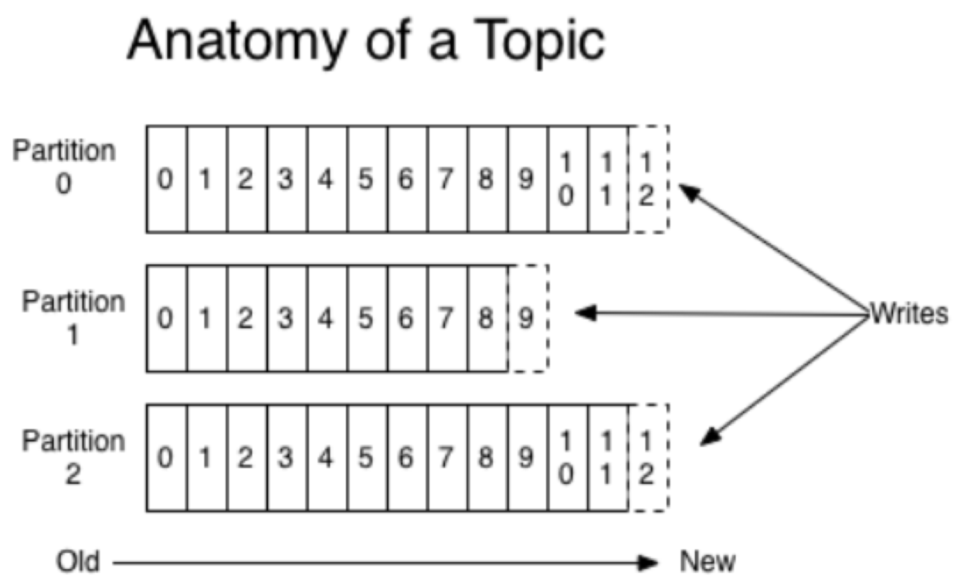


Kafka APIs

- The Producer API allows an application to ***publish*** a stream of records to one or more Kafka topics.
- The Consumer API allows an application to ***subscribe*** to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a ***stream processor***, consuming an input stream from one or more ***topics*** and producing an output stream to one or more output topics, effectively ***transforming*** the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that ***connect*** Kafka ***topics*** to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Topics and Logs

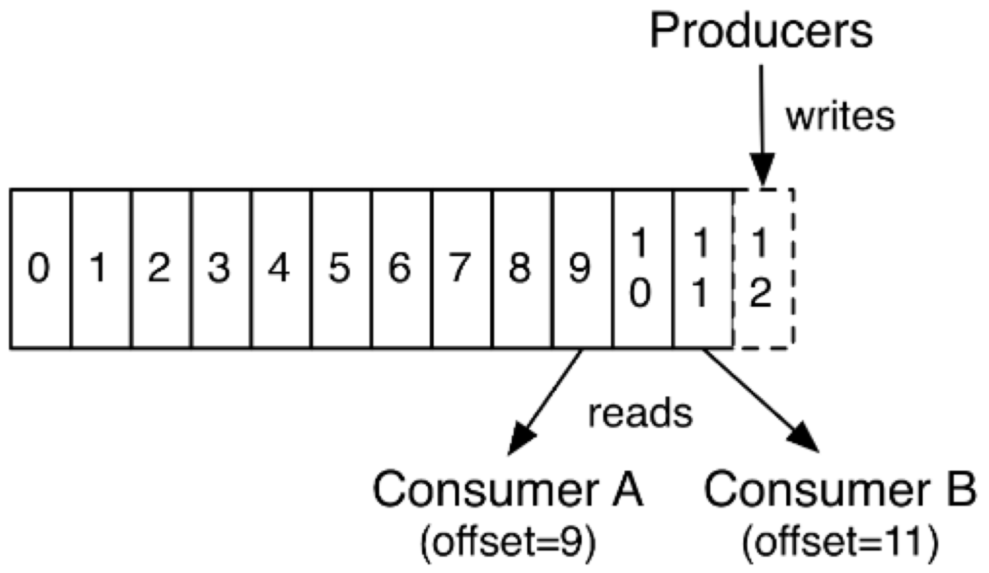
- A ***topic*** is a category or feed name to which records are published.
- For each ***topic***, the Kafka cluster maintains a ***partitioned*** log



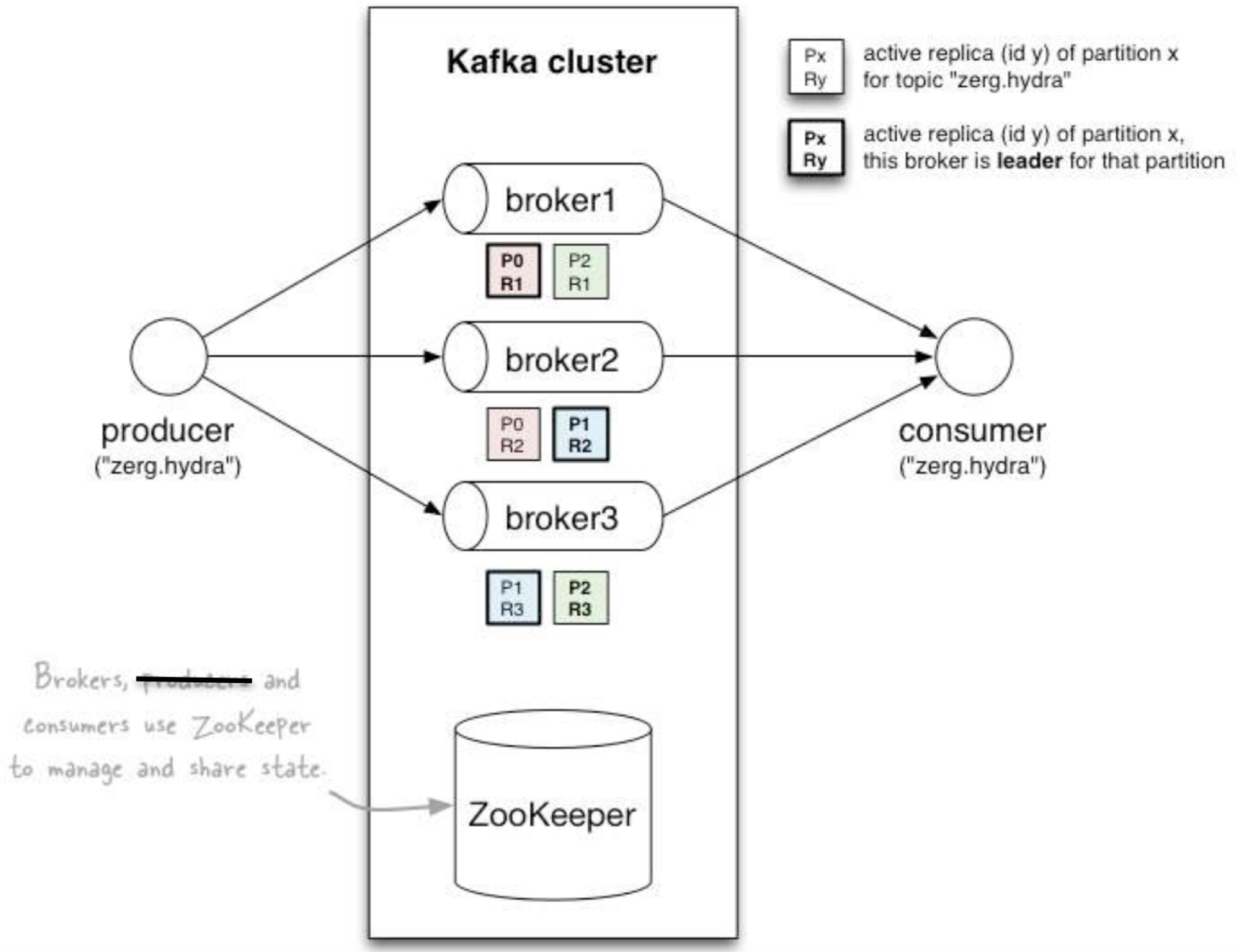
Partitions and Logs

- Each partition is an ordered, immutable sequence of records that is continually appended to—a structured commit log. The records in the partitions are each assigned a sequential id number called the *offset* that uniquely identifies each record within the partition.
- The Kafka cluster durably persists all published records—whether or not they have been consumed—using a configurable retention period. For example, if the retention policy is set to two days, then for the two days after a record is published, it is available for consumption, after which it will be discarded to free up space. Kafka's performance is effectively constant with respect to data size so storing data for a long time is not a problem.

Data is read in the order it arrives



Partitions are used for redundancy


Kafka cluster
Px Ry
active replica (id y) of partition x for topic "zerg.hydra"
Px Ry
active replica (id y) of partition x, this broker is **leader** for that partition
broker1
P0 R1
P2 R1
broker2
P0 R2
P1 R2
broker3
P1 R3
P2 R3
producer ("zerg.hydra")
consumer ("zerg.hydra")
Brokers, ~~producers~~ and consumers use ZooKeeper to manage and share state.
ZooKeeper

Summarize

- Publishers ***publish*** messages(***events***) to topics on ***brokers*** within the Kafka ***cluster***.
- Subscribers ***subscribe*** to topics within ***brokers***
- Messages (***events***) within ***topics*** arrive as an ***ordered sequence*** of events.
- This ordering is by ***the arrival time*** of the message(event) to the Kafka cluster.
- Messages(***events***) are consumed in the ***order they arrive*** to the Kafka cluster.

Implications for Time Series Analysis

You must consider the following with respect to time

- The time an event actually occurred.   For example, the time a stock price changed.
- The time this event arrived at the Kafka cluster. What happens if the data arrives late?
- The time that the event is processed or consumed by a Kafka subscriber. What if you have a slow consumer?