Apache Kafka

Next Generation Distributed Messaging System

INTRODUCTION

- Apache Kafka is a distributed publish-subscribe messaging system. It was originally developed at LinkedIn Corporation and later on became a part of Apache project. Kafka is a fast, scalable, distributed in nature by its design, partitioned and replicated commit log service.
- If huge data set has to be processed in real time than traditional Queue system like RabbitMQ or ActiveMQ does not help.
- Apache Kafka differs from traditional messaging system in:

  It is designed as a distributed system which is very easy to scale out.

  It offers high throughput for both publishing and subscribing.

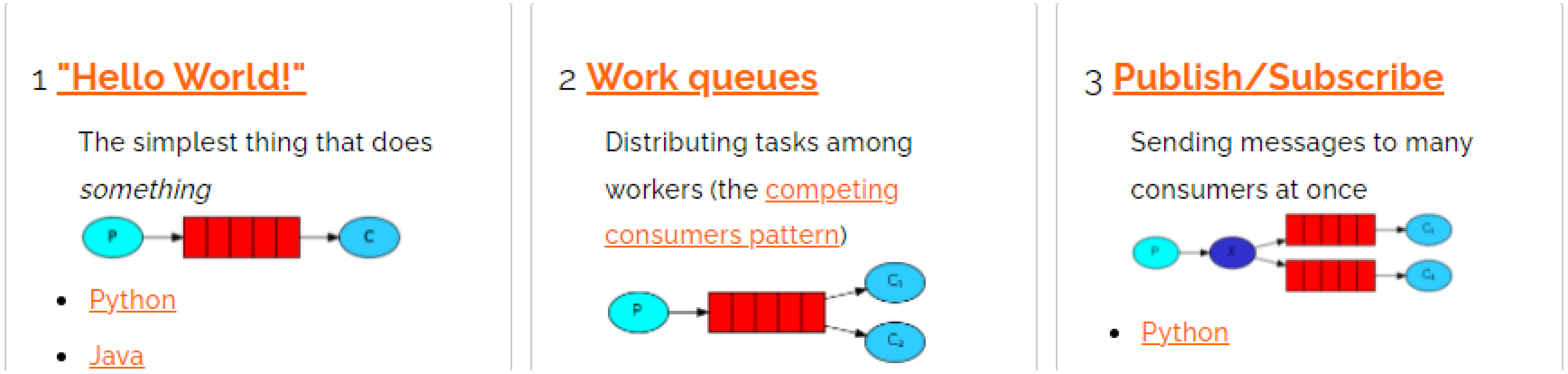
  It supports multi-subscribers and automatically balances the consumers during failure.

  It persist messages on disk and thus can be used for batched consumption such as ETL, in addition to real time applications.

  Kafka is a much better option when compared to traditional message server like RabbitMQ or ActiveMQ

Kafka VS RabbitMQ

- Kafka’s append-only log allows developers to access stream history and direct stream processing
- RabbitMQ’s message broker design excels in use cases that have specific routing needs and per-message guarantees.



1 "Hello World!"

The simplest thing that does *something*

- Python
- Java

2 Work queues

Distributing tasks among workers (the competing consumers pattern)

3 Publish/Subscribe

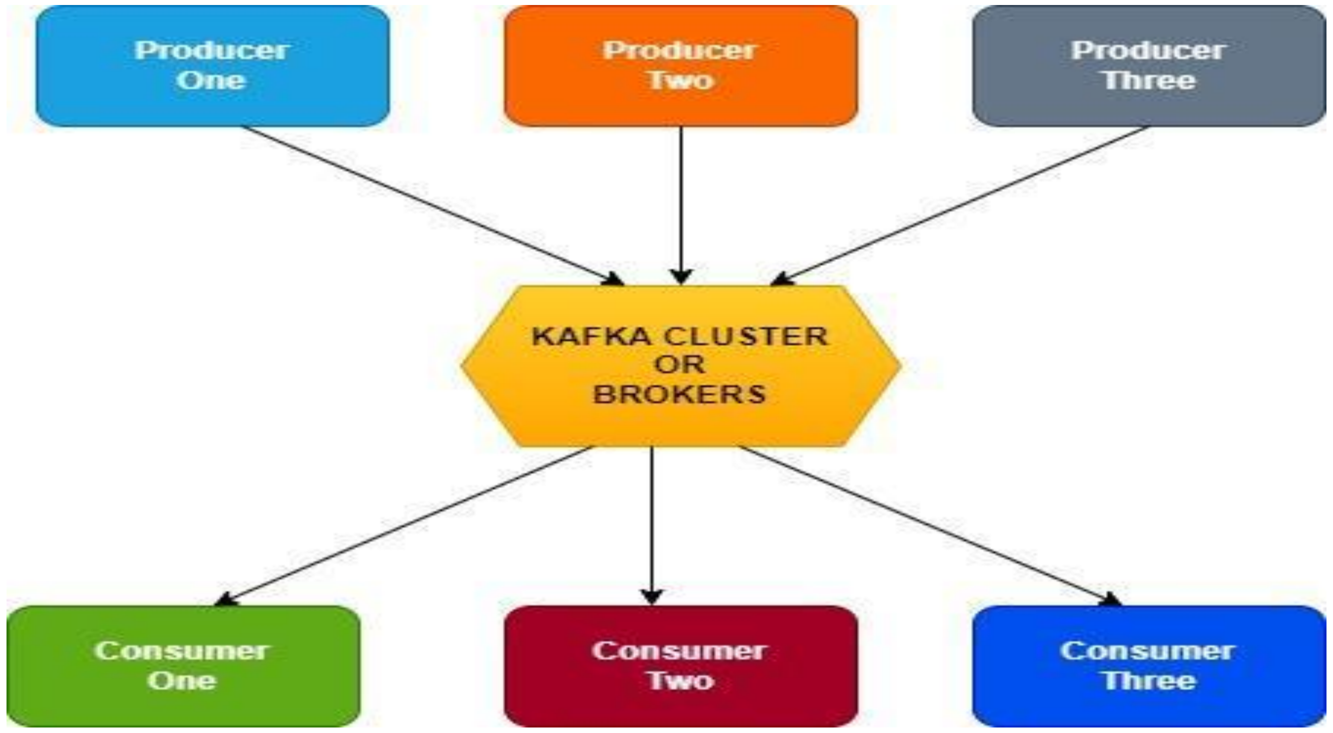
Sending messages to many consumers at once

- Python

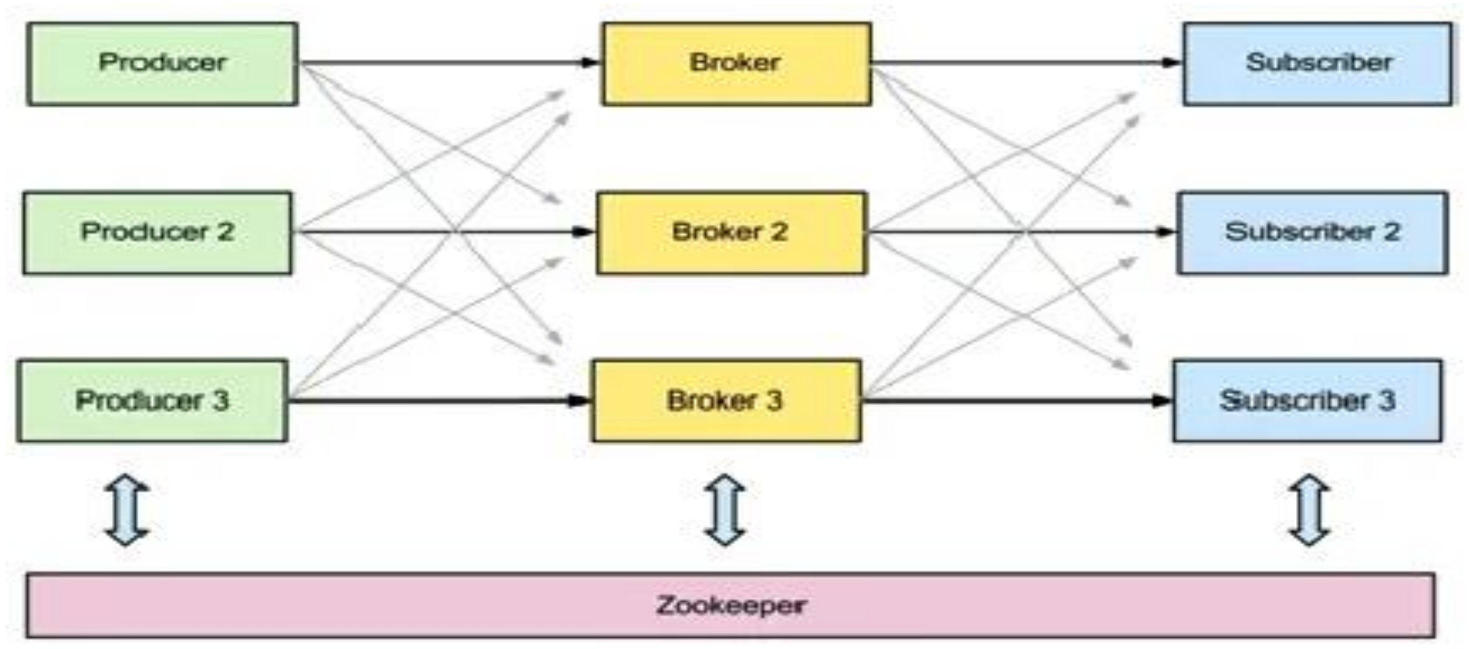
Terminology in Kafka

- A stream of messages of a particular type is defined as a topic.
- A Message is defined as a payload of bytes and a Topic is a category or feed name to which messages are published.
- A Producer can be anyone who can publish messages to a Topic.
- The published messages are then stored at a set of servers called Brokers or Kafka Cluster.
- A Consumer can subscribe to one or more Topics and consume the published Messages by pulling data from the Brokers.

Architecture



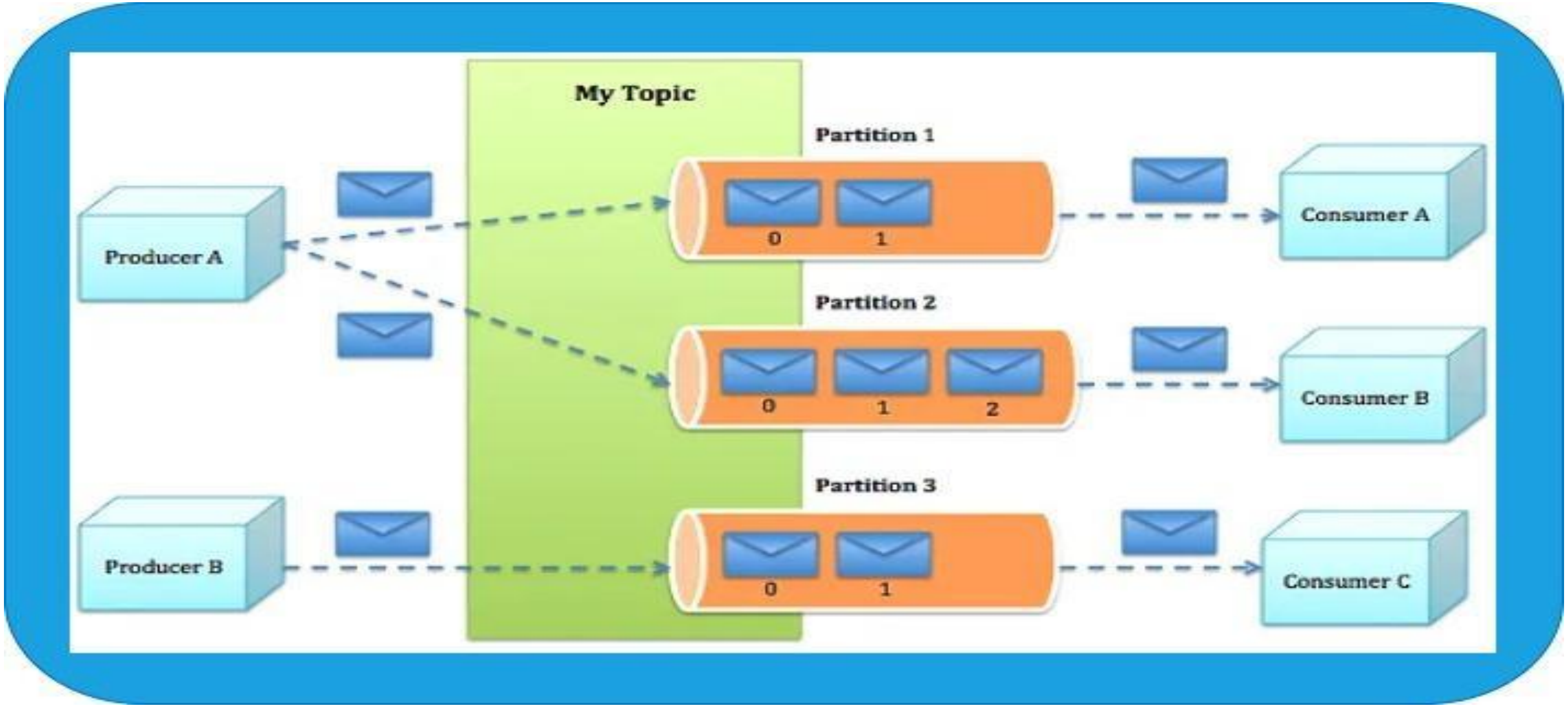
Architecture continued ...



Broker and Zookeeper(as loadbalancer)

- Unlike other message system, Kafka brokers are stateless. This means that the consumer has to maintain how much it has consumed. Consumer maintains it by itself and broker would not do anything.
- It is very tricky to delete message from the broker as broker doesn't know whether consumer consumed the message or not. Kafka innovatively solves this problem by using a simple time-based SLA for the retention policy. A message is automatically deleted if it has been retained in the broker longer than a certain period.
- This innovative design has a big benefit, as consumer can deliberately rewind back to an old offset and re-consume data. This violates the common contract of a queue, but proves to be an essential feature for many consumers.
- ZooKeeper is used for managing, coordinating Kafka broker. Each Kafka broker is coordinating with other Kafka brokers using ZooKeeper. Producer and consumer are notified by ZooKeeper service about the presence of new broker in Kafka system or failure of the broker in Kafka system.
- Since Kafka is distributed in nature, a Kafka cluster typically consists of multiple brokers. To balance load, a topic is divided into multiple partitions and each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

Detailed Architecture



Kafka Storage

- Kafka has a very simple storage layout.
- Each partition of a topic corresponds to a logical log.
- Physically, a log is implemented as a set of segment files of equal sizes.
- Every time a producer publishes a message to a partition, the broker simply appends the message to the last segment file.
- Segment file is flushed to disk after configurable numbers of messages have been published or after a certain amount of time elapsed.
- Messages are exposed to consumer after it gets flushed.
- Unlike traditional message system, a message stored in Kafka system doesn’t have explicit message ids.