APACHE PULSAR

1. Introduction

- Apache Pulsar is an open-source distributed pub-sub messaging platform developed by Yahoo and now being maintained under Apache Software Foundation.
- It's designed to handle high volumes of data with low latency, offering powerful capabilities for real-time data stream processing and analytics.
- Pulsar's key features include multi-tenancy, high throughput, low latency, and geo-replication.

2. Architecture

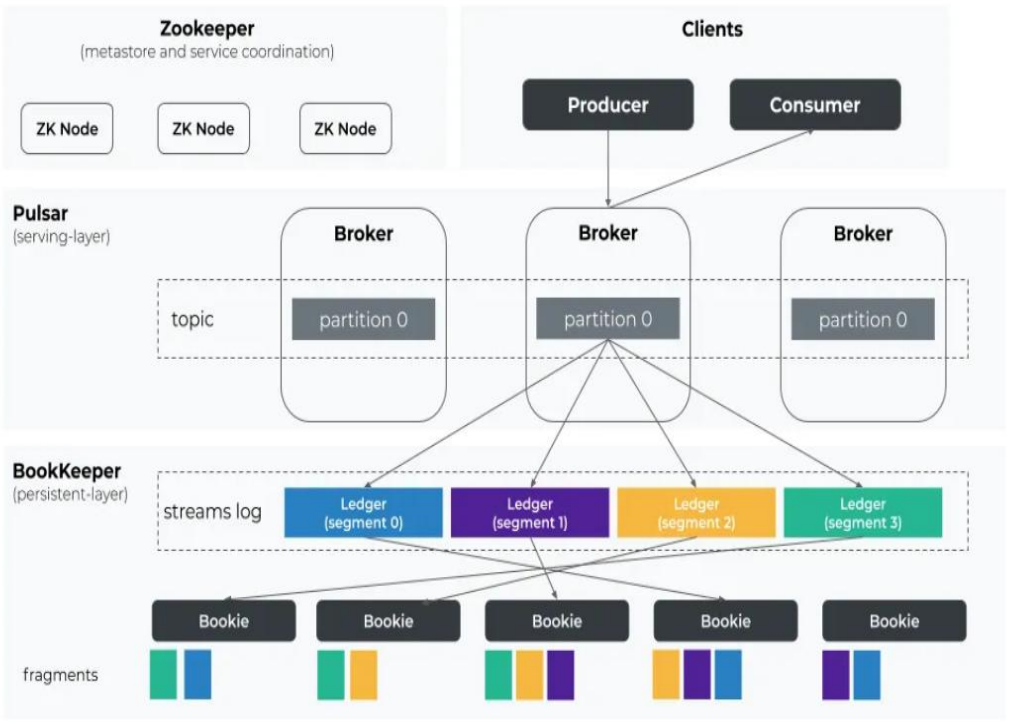
- An Apache Pulsar cluster can be viewed as a two-layer architecture, one is called the serving layer, the other is called the persistence layer.
- Brokers
- The serving layer is composed of one or more nodes, each hosting a broker, which are responsible for handling and load-balancing incoming requests from producers and consumers.
- The serving layer is stateless which means that brokers do not directly stored any data locally. Instead of that, brokers rely on the persistence layer, as its name suggests, to persist messages on disk through the used of Apache BookKeeper.

Log Streams, Ledgers and Bookies

- Apache BookKeeper is another open-source service that provides reliable and resilient persistent storage of streams of log entries.
- A log streams can be defined as an unbounded sequence of ordered and immutable records and is usually implemented using an append-only structure (also call a Write-aHead-Log).
- In BookKeeper streams of log entries are composed of ledgers which are managed by individual server nodes called bookies.
- Pulsar uses ledgers for storing messages written to topic partitions.
- Each topic partition is assigned to multiple ledgers and messages are written to those ledgers.
- Then, ledgers are stripped across an ensemble of bookies so that each bookie stores fragments of ledgers.

Apache Zookeeper

- Finally, both Pulsar and BookKeeper use Apache Zookeeper as a metastore.
- Pulsar uses it to persist specific information about the cluster, like the topics configuration, and as a coordination service to manage distributed elections, among other things.



Apache Pulsar — Two-Layer Architecture Overview

3. Key Features

- ***Rapid Horizontal Scalability***
  - Scales horizontally to handle the increased load. Its unique design and separate storage layer enable handling the sudden surge in traffic by scaling out in seconds.
- ***Low-latency messaging and streaming***
  - Acknowledge messages individually (RabbitMQ style) or cumulative per partition (i.e., offset-like). Enables use cases such as distributed work queues or order-preserving data streams at very large scales (hundreds of nodes) and low latency (<10ms).
  - ***Seamless Geo-Replication***
    - Protect against complete zone outages using replication across different geographic regions. Flexible and configurable replication strategies across distant Pulsar Clusters. Uniquely supports automatic client failover to healthy clusters.
- ***Multi-tenancy as a first-class citizen***

- Maintain one cluster for your entire organization using tenants. Access control across data and actions using tenant policies. Isolate specific brokers to a tenant when maximum noisy neighbor protection is needed.

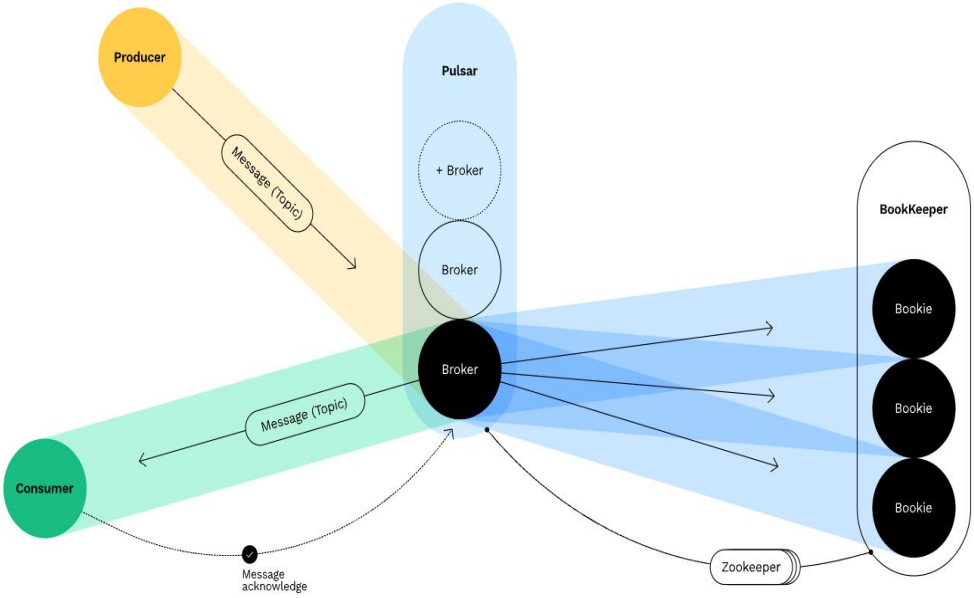
- ***Automatic Load Balancing***

- Add or remove nodes and let Pulsar load balance topic bundles automatically. Hot spotted topic bundles are automatically split and evenly distributed across the brokers.

Official multi-language support

- Officially maintained Pulsar Clients for Java, Go, Python, C++, Node.js, and C#.
- Official 3rd party integrations
- Pulsar has officially maintained connectors with popular 3rd parties: MySQL, Elasticsearch, Cassandra, and more. Allows streaming data in (source) or out (sink).
- Serverless Functions
- Write and deploy functions natively using Pulsar Functions. Process messages using Java, Go, or Python without deploying fully-fledged applications. Kubernetes runtime is bundled.
- Supports up to 1M topics
- Pulsar's unique architecture supports up to 1 million topics in a single cluster. Simplify your own architecture by avoiding multiplexing multiple streams into a single topic.

How does Pulsar work



- Pulsar is a multi-tenant, high-performance solution for server-to-server messaging.
- It's composed of a set of brokers and bookies along with an inbuilt Apache ZooKeeper for configuration and management.
- The bookies are from Apache BookKeeper which provide storage for the messages until they are consumed.
- In a cluster we'll have:
- Multiple cluster brokers to handle the incoming message from producers and dispatch the message to consumers
- Apache BookKeeper to support message persistence
- Apache ZooKeeper to store the cluster configuration

Producer & Consumer

- A Pulsar client contains a consumer and a producer. A producer writes messages on a topic. A consumer reads messages from a topic and acknowledges specific messages or all up to a specific message.

Metadata Store

- Pulsar and BookKeeper use a metadata store to save metadata coordinated between nodes, such as a list of ledgers per topic, segments per ledger, and mapping of topic bundles to a broker.
- New clusters use Oxia (recommended); Apache ZooKeeper is also supported. It's a cluster of highly available and replicated servers (usually 3).

Pulsar Brokers

- Topics (i.e., partitions) are divided among Pulsar brokers. A broker receives messages for a topic and appends them to the topic's active virtual file (a.k.a ledger), hosted on the Bookkeeper cluster.
- Brokers read messages from the cache (mostly) or BookKeeper and dispatch them to the consumers.
- Brokers also receive message acknowledgments and persist them to the BookKeeper cluster as well. Brokers are stateless (don't use/need a disk).

Apache Bookkeeper

- Apache BookKeeper is a cluster of nodes called bookies.
- Each virtual file (a.k.a ledger) is divided into consecutive segments, and each segment is kept on 3 bookies by default (replicated by the client - i.e., the broker).
- Operators can add bookies rapidly since no data reshuffling (moving) between them is required. They immediately share the incoming write load.

Flexible subscriptions

- Apache Pulsar supports four different subscription types1 to help users configure messaging patterns:

- **Exclusive:** Only one consumer is allowed to receive messages through the subscription.
- **Failover:** When message transmission to a designated primary consumer fails, messages are sent to a standby consumer instead for processing.
- **Shared:** Messages are delivered in a "round robin" manner, with multiple consumers receiving messages, but each message is delivered to one consumer
- **Key shared:** Messages are assigned to a message key, which helps correlate related messages. Messages with the same key are sent to consumers in a specific order.

Apache Pulsar use cases

- ***Consolidation of messaging platforms***
  - Apache Pulsar is often the platform of choice when enterprises decide to pursue messaging technology consolidation because it supports multiple messaging use cases (including message queuing and streaming) and multi-tenancy, enabling multiple teams to use it in the ways that serve them best.
- ***Failure resiliency for mission critical applications***
  - Enterprises seeking to minimize the chance of data loss for critical applications—such as financial transactions—can leverage Apache Pulsar for its failure resiliency features. Namely, messages fed through Pulsar are replicated to several BookKeeper nodes. In addition, messages don't get lost even when hardware fails.
- ***Microservices communication***
  - Apache Pulsar supports the constant communication required between microservices through indirect API calls. Services can send messages to topics to which other services are subscribed. While other messaging systems can provide this capability, Pulsar's horizontal scalability sets it apart—the platform can scale up within minutes to accommodate large influxes of requests.
- ***Task queue support***
  - Messaging systems commonly support task queues—systems that organize asynchronous execution of background jobs without hindering an application's performance. Apache Pulsar supports task queue systems through its shared subscription—which distributes messages to multiple consumers—and message acknowledgment capability, which confirms a task's completion.