

DataStax

DataStax Monday Learning

- From engineers to engineers
- Open-source only
- Hands-on experience
- Live communication with experts

DataStax

Led By Aleks and Zeke

Event Streaming Series

Episode I: Introduction to Event Streaming



MENTI!

Agenda

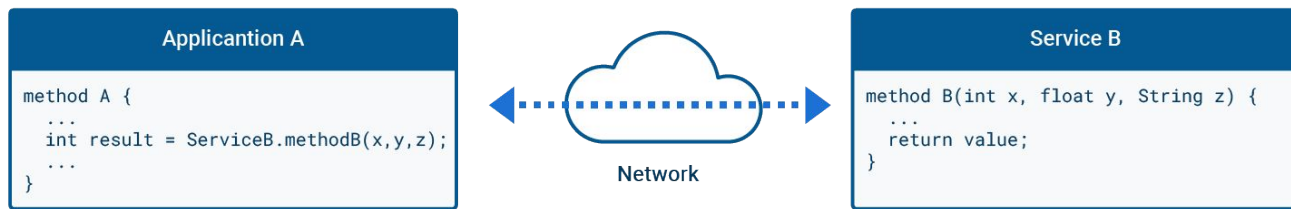
- Fundamentals of Messaging
- Introduction to Pulsar
- Hands on

DS

Fundamentals of Messaging

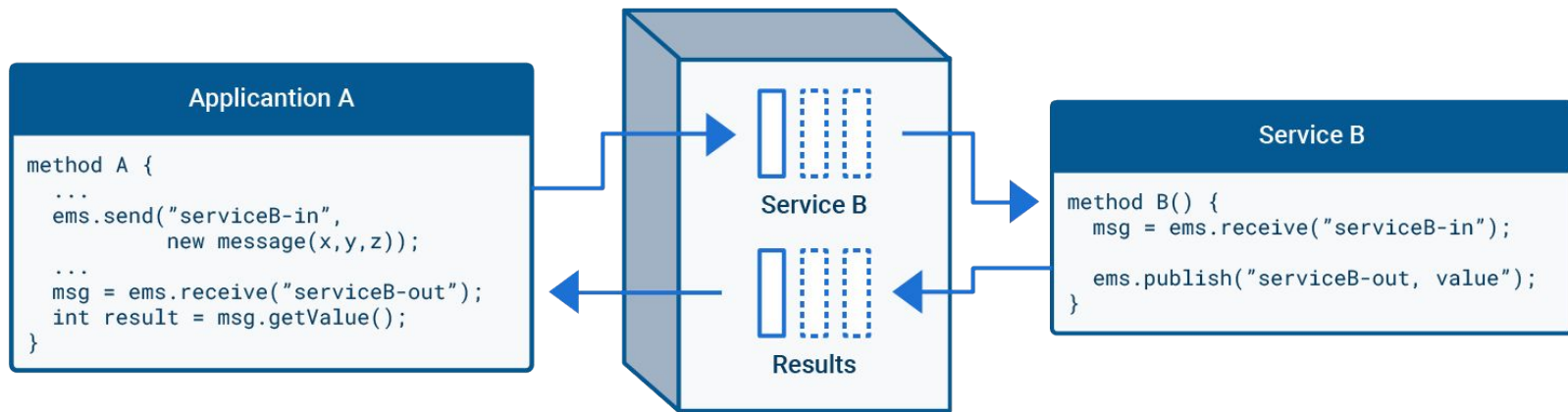
Remote Procedure Calls – Pain before Messaging Systems

- Remote Procedure Calls
 - Application A makes a request from service B
 - Service B responds

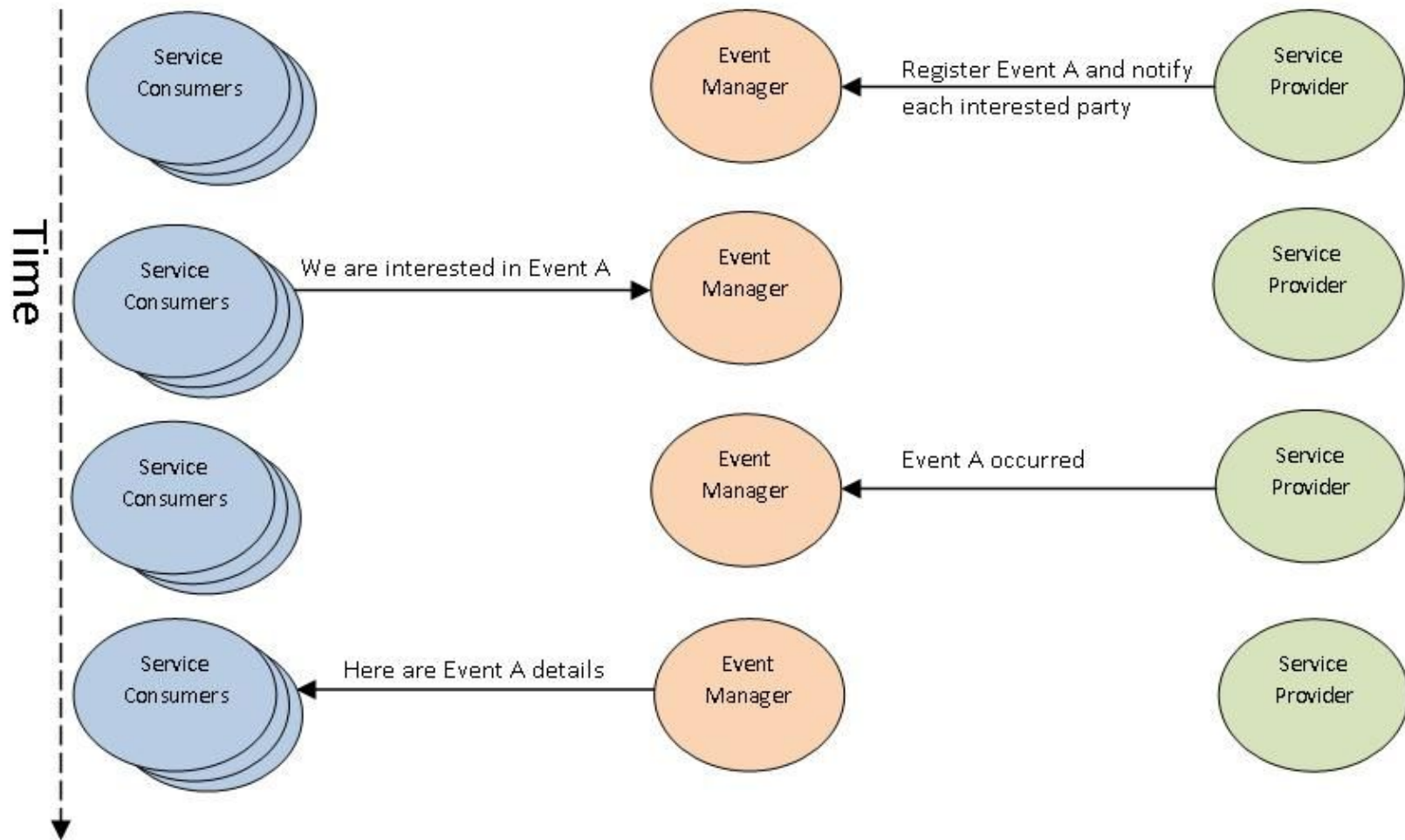


- Problems
 - Huge risk with networking
 - Needs service discovery
 - Point to Point communications
 - Slow (blocking) as you are waiting for one server to respond to another
 - If a receiver is not available, process is failed
 - One and Only One receiver

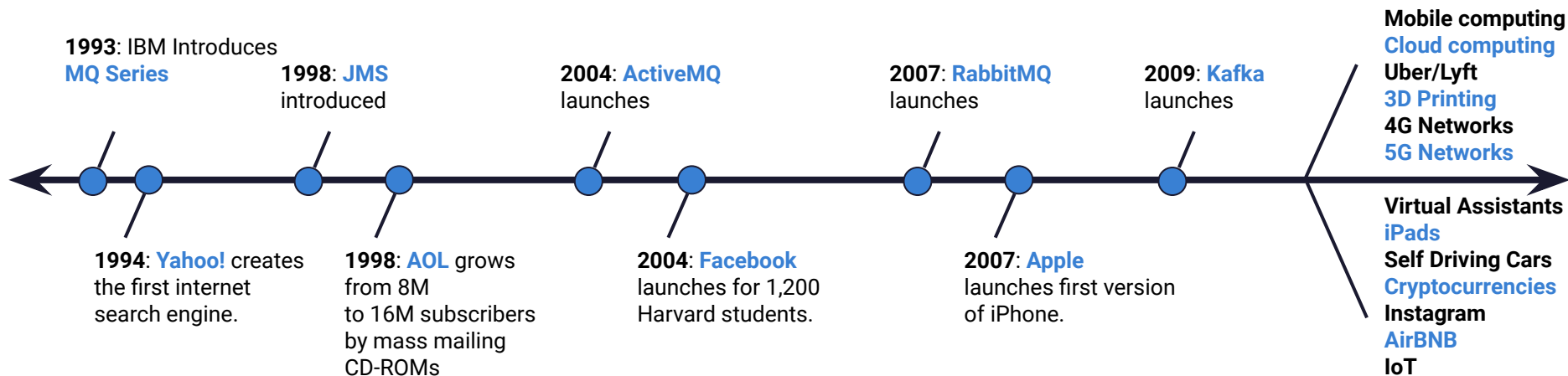
Message Systems solved RPC limits



- Decouple message senders from the receivers by using a **messaging service**
- Persistent buffer to handle requests and data
- Async = no need to wait!
- Multiple Receivers!
- Perfect fit for distributed systems / microservices
- Message will be consumed by the next available service instance



Most enterprise messaging systems were made to solve yesterday's problems



Messaging System Features

01

Asynchronous
Communication

02

Message Retention

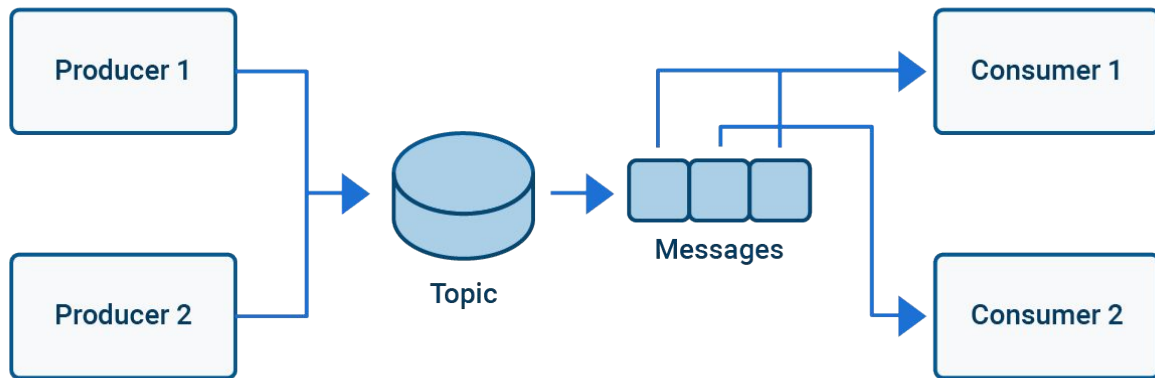
03

Acknowledgement

04

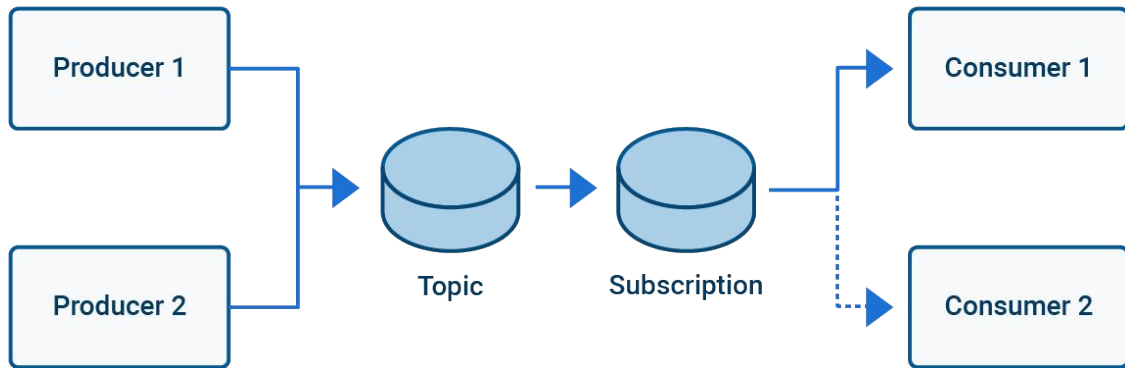
Message
Consumption

Message Consumption - Queuing



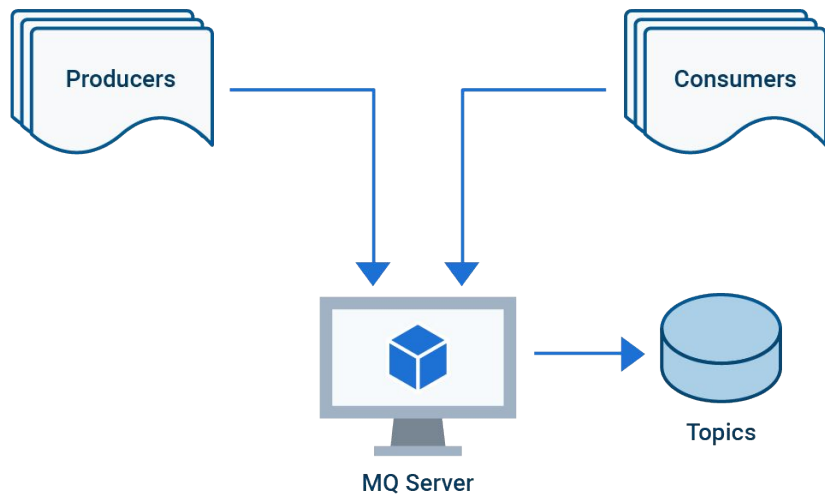
- Exactly One Processing of messages
- Backlog of messages
- Distributed Processing
- Message Failover
- Easy to Scale

Message Consumption – Publish / Subscribe Messaging



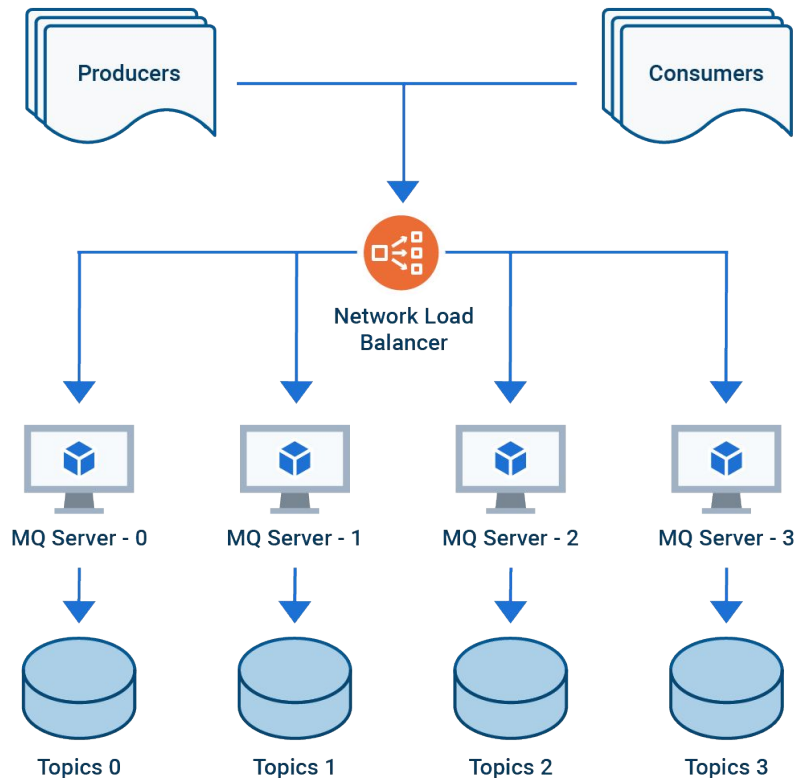
- Producers send to Topic
- Consumers subscribe to Topic
- Multiple consumers
- Easy to extend

Message Oriented Middleware – MQ – Original Design



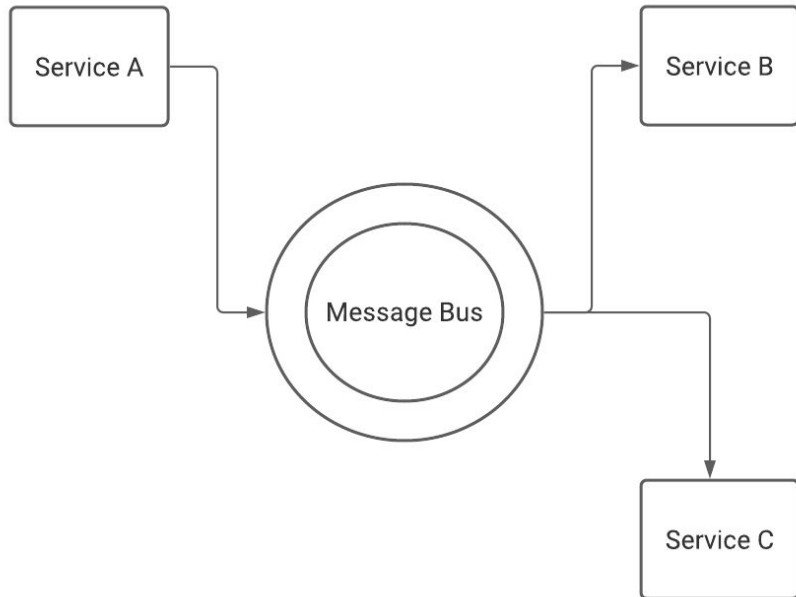
- IBM WebSphere MQ - 1993
- Single Machine Design
- Vertical Scaling
- Single Point of Failure

Message Oriented Middleware – MQ – Current Design



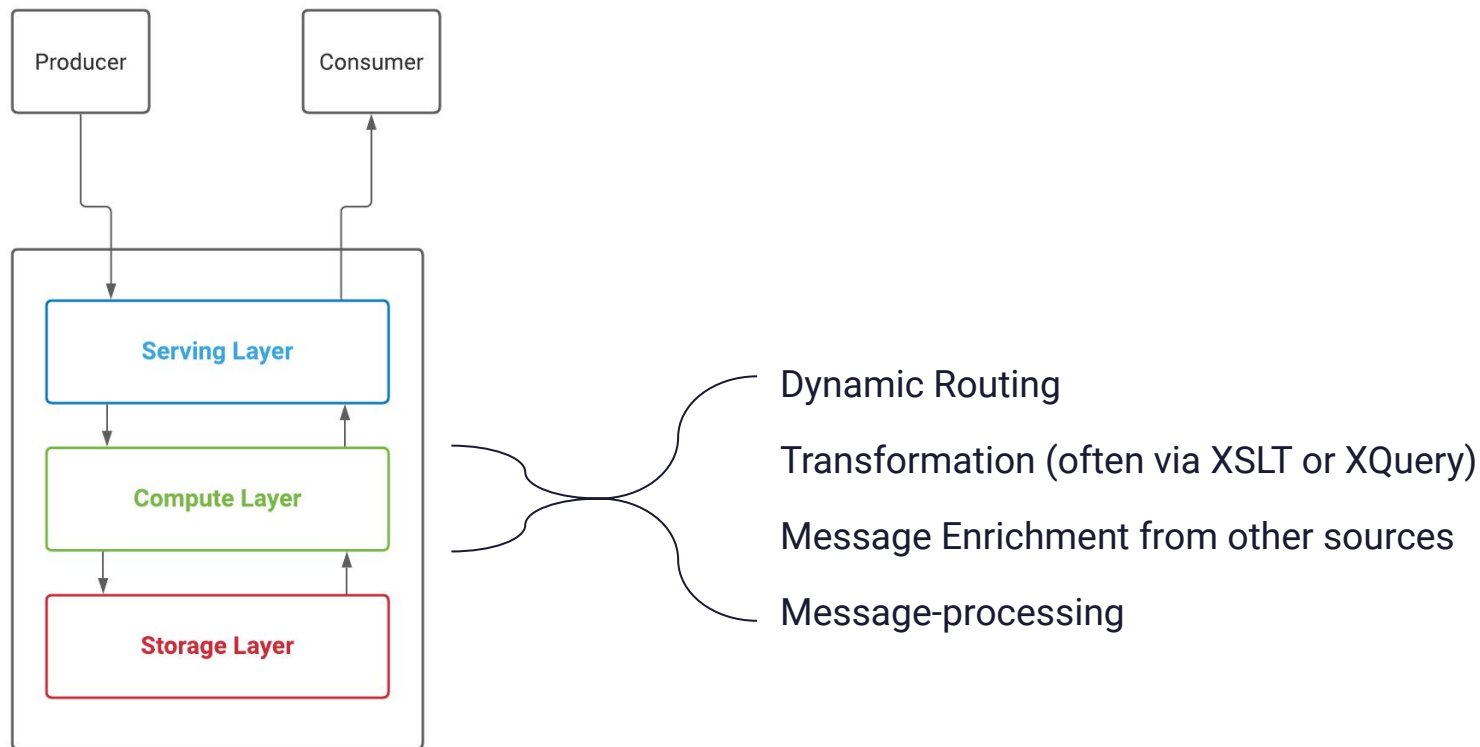
- Clustering allowed the load to be spread across multiple servers instead of just one.
- Each server in the cluster was responsible for handling only a portion of the topics.

Enterprise Message Bus – Message Routing

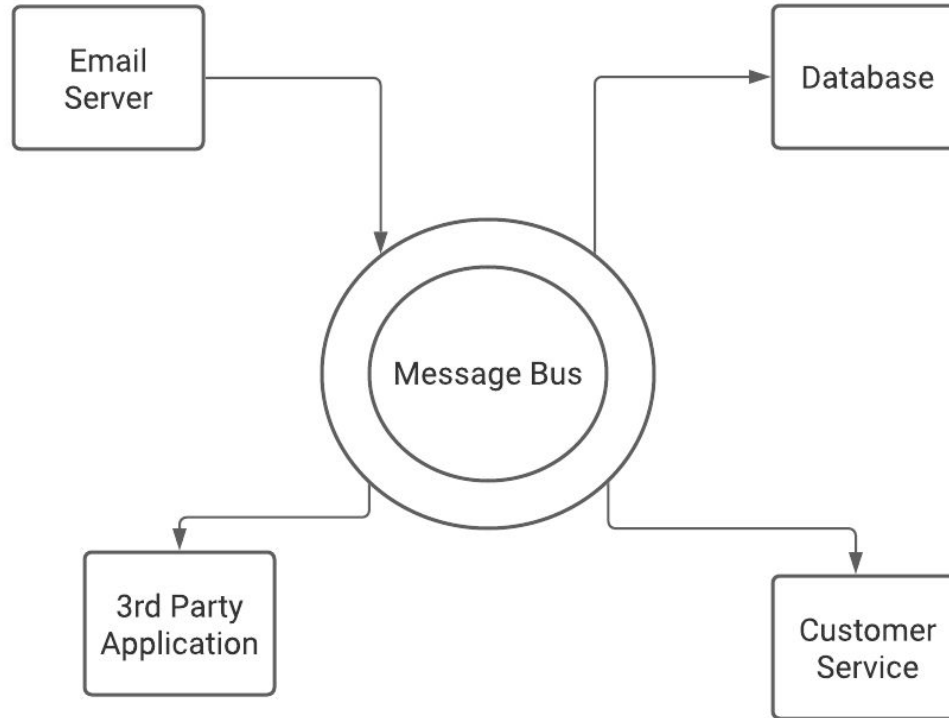


- Designed to eliminate the need for point-to-point communication.
- Service A merely publishes its message to the bus
- Message routed to applications B and C
- Data enrichment and transformations

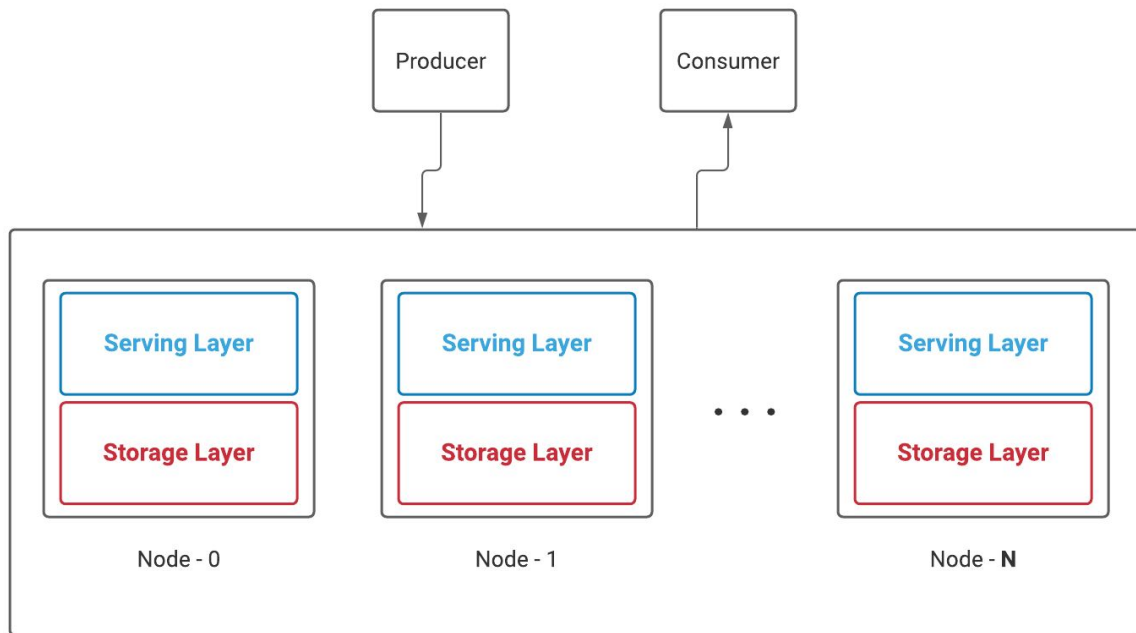
Enterprise Message Bus – Message Enrichment



Enterprise Message Bus – 3rd Party Integration



Distributed Messaging Systems – Kafka

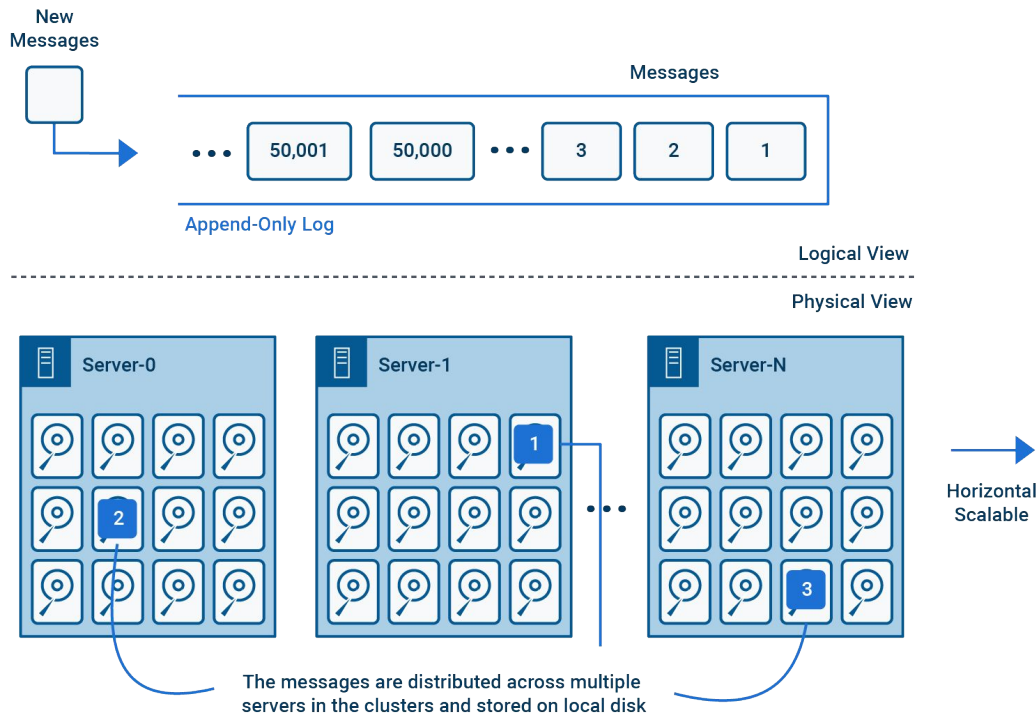


- Each machine should share the messaging load and scale
- Horizontally scalable on commodity hardware
- A single topic is distributed across multiple machines

What is Kafka great for?

- Streaming
- Distributed commit log
- Huge Scale
- Fast

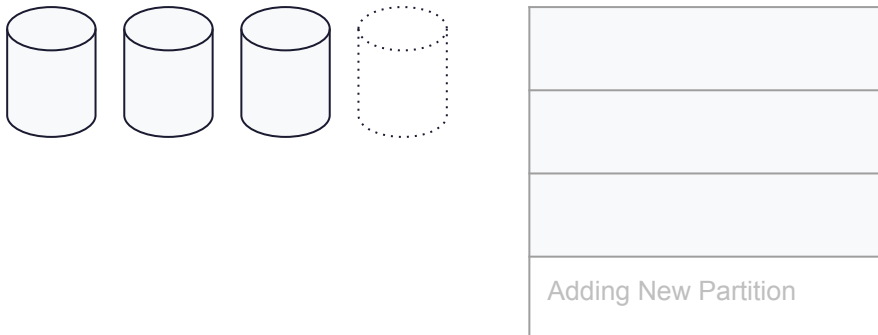
Distributed Messaging Systems – Kafka



- Append only log
- Messages in a topic stored sequentially
- Distributed Storage

Problems I ran into with Kafka

Scale



Application Development

- Steep Learning Curve
- Hard to Debug

DataStax

Introduction to Apache Pulsar

Agenda

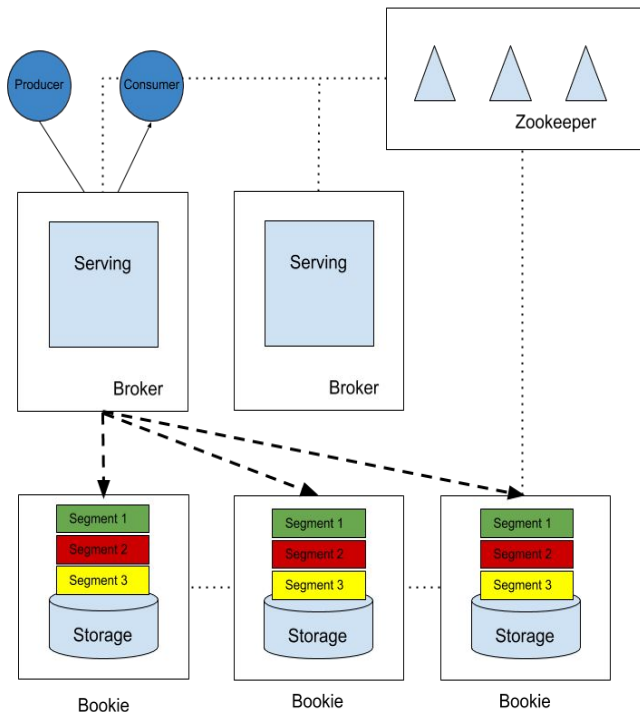
- Intro and Architecture
- Deployment and Development
- Pulsar Components and Ecosystem
- Message Processing - Core Concepts and Processing Model
- Multi-Tenancy in Pulsar
- Geo-replication In-depth
- Summary and Q&A

What is Apache Pulsar

- Distributed pub-sub messaging system
 - High throughput, low latency
 - Separates compute from storage
 - Horizontally scalable
 - Streaming and queuing
- Open source
 - Originally developed at Yahoo!
 - Contributed to the Apache Software Foundation (ASF) in 2016
 - Top-level project (2018)
 - 7.2K GitHub stars, over 300 contributors

Architecture

- Distributed, tiered architecture
- Separates compute from storage
- ZooKeeper holds metadata for the cluster
- Stateless Broker handles producers and consumers
- Storage is handled by Apache BookKeeper
 - BookKeeper distributed, append-only log
 - Data is broken into segments written to multiple bookies



Why Apache Pulsar

Four Reasons Why Apache Pulsar is Essential to the Modern Data Stack

- Geo-replication
- Scaling
- Multitenancy
- Queuing (as well as Streaming)

- Cloud Native
 - K8s
 - Multi-cloud, hybrid-cloud

- Performance
 - High throughput
 - Consistent low latency

Deployment and Development

Deployment

- On-prem
- Ansible script
- Docker image
 - OSS image: JDK8
 - DataStax image: JDK11 (more performing)
 - Bug fix + improvements (e.g. enhanced C* sink connector)
- K8s Helm chart
 - OSS and DataStax
- Replicated K8s (DataStax)
- Pulsar-as-a-service
 - Kesque
 - Astra Streaming (future)

Development

- Client APIs
 - Java
 - Python
 - C++
 - Go (C++ wrapper)
 - Native Go
 - Node.js
 - C#
 - WebSocket
- Pulsar Rest API
- Pulsar Admin Rest API
- Community clients
 - .NET
 - Scala
 - Rust
 - HTTP
 - Haskell

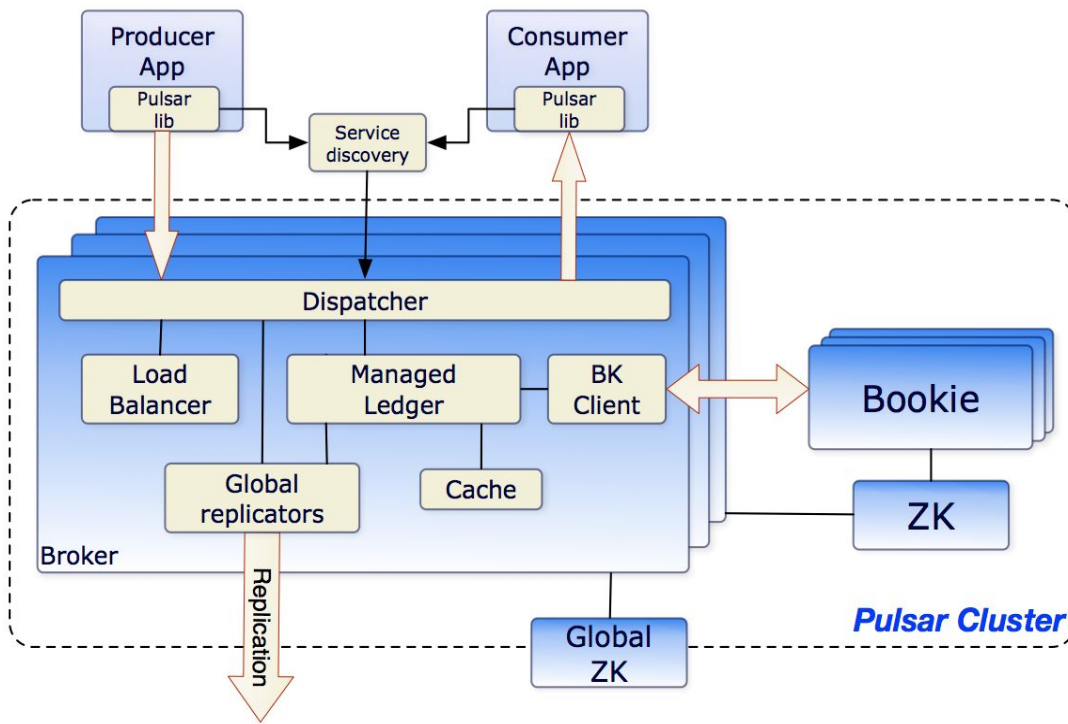
Pulsar Components and Ecosystem

Zookeeper

- For metadata storage, cluster configuration, and coordination
- Instance/Global level Zookeeper
 - Geo-replication
 - Configuration for tenants, namespaces, and other entities that need to be globally consistent
 - Optional
- Cluster level Zookeeper
 - Ownership metadata
 - Broker load reports
 - BookKeeper ledger metadata
 -

Broker

- Stateless
- Topic ownership
- Load Balancing
- Pulsar's "Brain"
 - HTTP Rest APIs for Admin tasks and topic lookup
 - TCP binary protocol for data transfers



Broker (Topic Ownership), cont'd

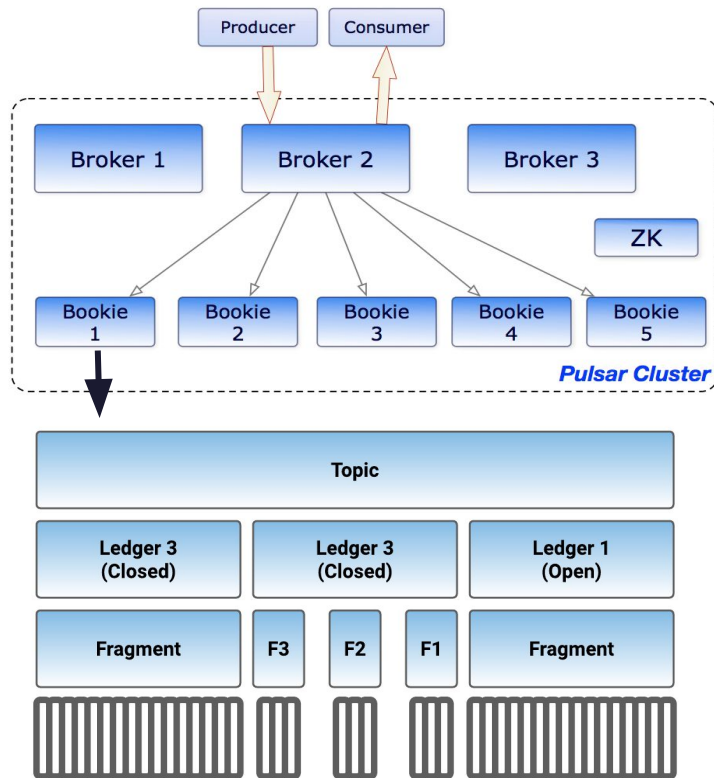
- Stateless broker makes possible of **dynamic assignment**
 - Topic ownership can change on the fly
 - Broker crash
 - Overloaded broker
- Ownership assignment granularity
 - Namespace **bundle** (subset of a namespace)
 - `defaultNumberOfNamespaceBundles=4`
 - Consistent hashing
 - C* ring

Broker (Load Balancing), cont'd

- Unload topics and bundles
 - Triggers topic re-assignment based on the current workload
 - Automatic or manual
- Bundle Split
 - Tunable thresholds
 - Automatically triggers topics unloading
- Automatic Load Shedding
 - Forces topics unloading from overloaded brokers
 - Enabled by default and can be disabled

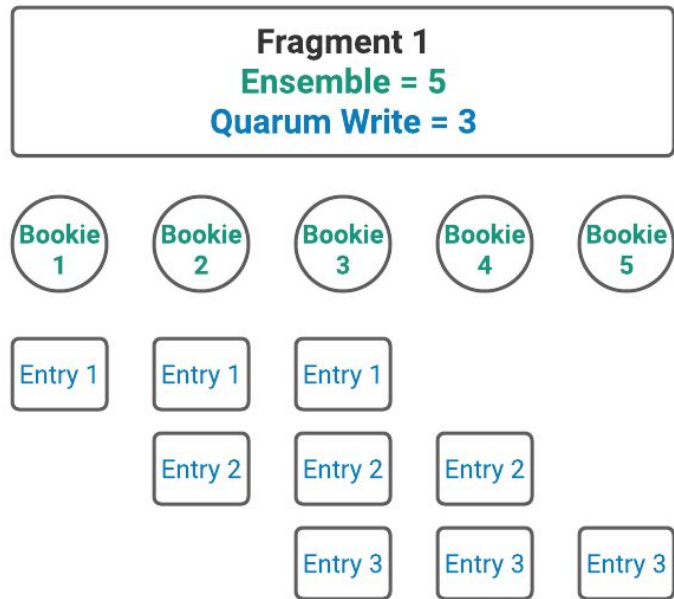
Bookkeeper (Bookie)

- Managed ledger (Topic)
 - Logical abstraction
 - A stream of ledgers
- A Ledger is created when:
 - A new topic is created, or
 - Roll-over occurs (size/time limit, ownership change)
- Append only
 - Read only after closed
- Fragments and Entries



Bookkeeper (Bookie), cont'd

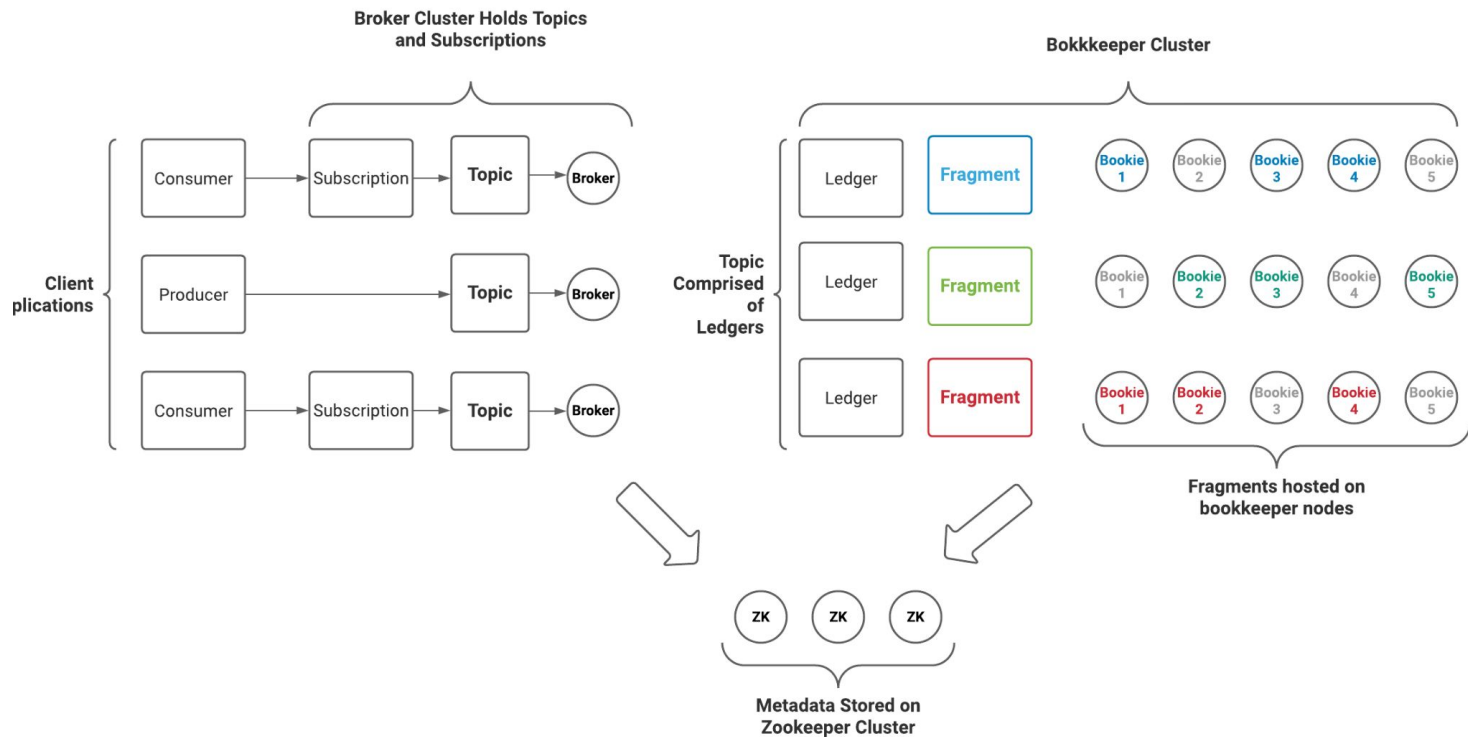
- Key Configuration
 - Ensemble Size (E)
 - The size of the pool of Bookies available for writes
 - Write Quorum Size (Qw)
 - The number of actual Bookies that Pulsar will write an entry to
 - Ack Quorum Size (Qa)
 - The number of Bookies that must acknowledge the write
- Rack-awareness
- WAL Journal
 - Separate Journal and Ledger disks



Bookkeeper (Bookie), cont'd

- Bookie Recovery
 - The process of "replicating" fragments to ensure the replication factor (Qw) is maintained for each ledger.
 - Manual recovery
 - `bin/bookkeeper shell recover <failed_bookie_ip_port>`
 - Automatic recovery
 - Detects failed bookie automatically
 - Replicates all the ledgers that were stored on that bookie.
 - Dedicated recovery hosts or sharing the same bookie hosts

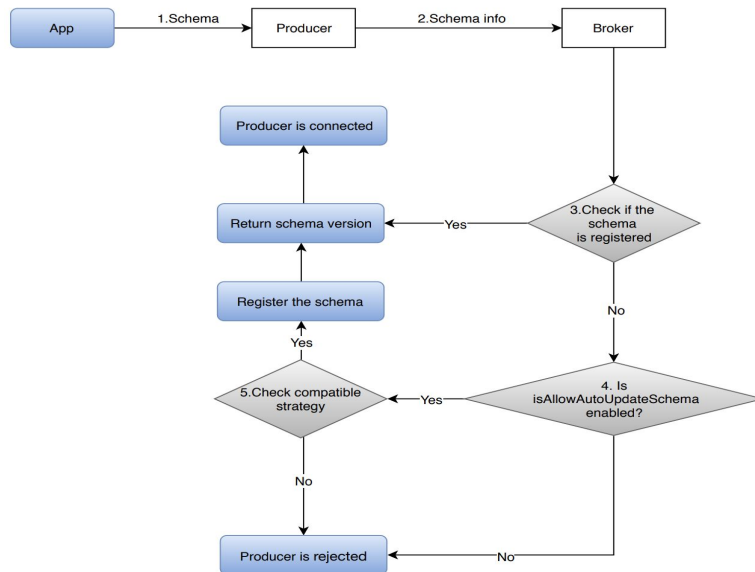
Put all Together



Other Components and Ecosystem

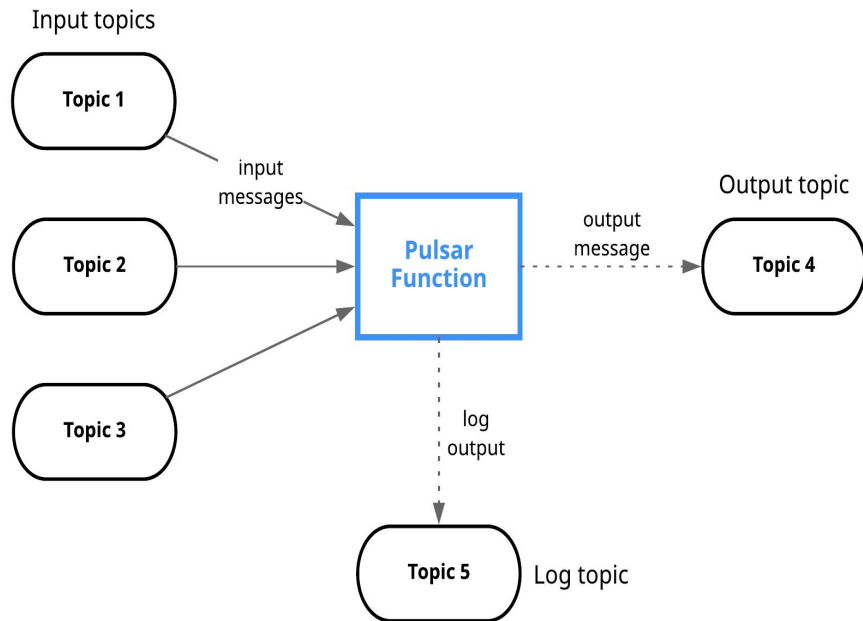
- Pulsar Proxy
 - Internal “gateway” to brokers
 - No direct connection to broker
 - K8s deployment

- Pulsar Schema
 - Built-in schema registry
 - Schema type
 - Primitive
 - Key/value pair
 - Avro, JSON, Protobuf
 - Schema evolution
 - Version
 - Compatibility
 - Schema Management
 - Automatic
 - Manual



Other Components and Ecosystem, cont'd

- Pulsar Function
 - Allows complex streaming processing
 - Light-weight
 - Function-as-a-service (AWS Lambda, Google Function, ...)
 - Main languages:
 - Java
 - Python
 - Go



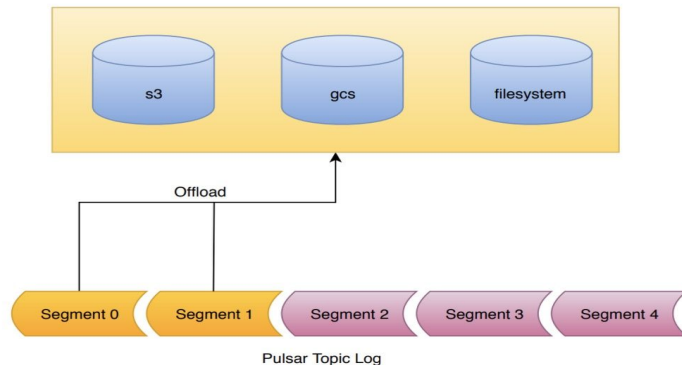
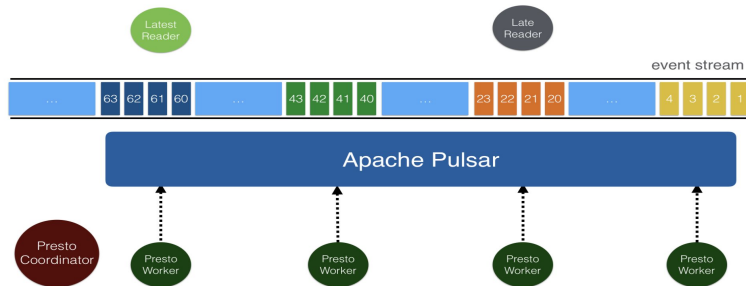
Other Components and Ecosystem, cont'd

- Pulsar I/O
 - Source Connectors
 - Sink Connectors
- Built-in Source Connector
 - RDBMS
 - Kafka (**DataStax Enhanced version**)
 - Kinesis
 -
- Built-in Sink Connector
 - ElasticSearch
 - Cassandra (**DataStax Enhanced Version**)
 - MongoDB
 - RDBMS
 -
- CDC Connector
 - Canal
 - Debezium (MySQL, PostgreSQL, MongoDB)
- Custom I/O Connector through API



Other Components and Ecosystem, cont'd

- Pulsar SQL
 - Through Presto (now called Trino)
 - Presto Pulsar Connector
 - `${project.root}/conf/presto/catalog/pulsar.properties`
- Tiered Storage
 - Pulsar is tiered (compute and storage)
 - Further tier in storage
 - AWS S3 offloader
 - GCS offloader
 - File system offloader
 - Azure blob offloader
 - Transparent to client
 - Can be configured to run automatically or manually

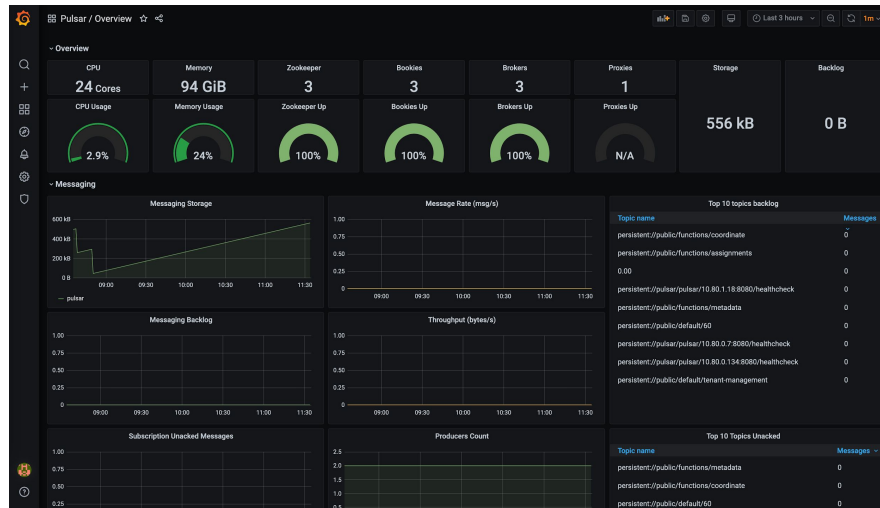


Other Components and Ecosystem, cont'd

- Security
 - Pluggable Authentication
 - Certificate based authentication
 - JWT token based authentication
 - Authenz
 - Kerberos
 - OAuth 2.0
 - Pluggable Authorization
 - Role based ("role token")
 - Authenz
 - Internal
 - TLS encryption
 - Traffic encryption
 - Between clients (producers/consumers) and Pulsar
 - Among Pulsar components
 - Message encryption (outside Pulsar)
 - End-to-end encryption
 - Volume based encryption

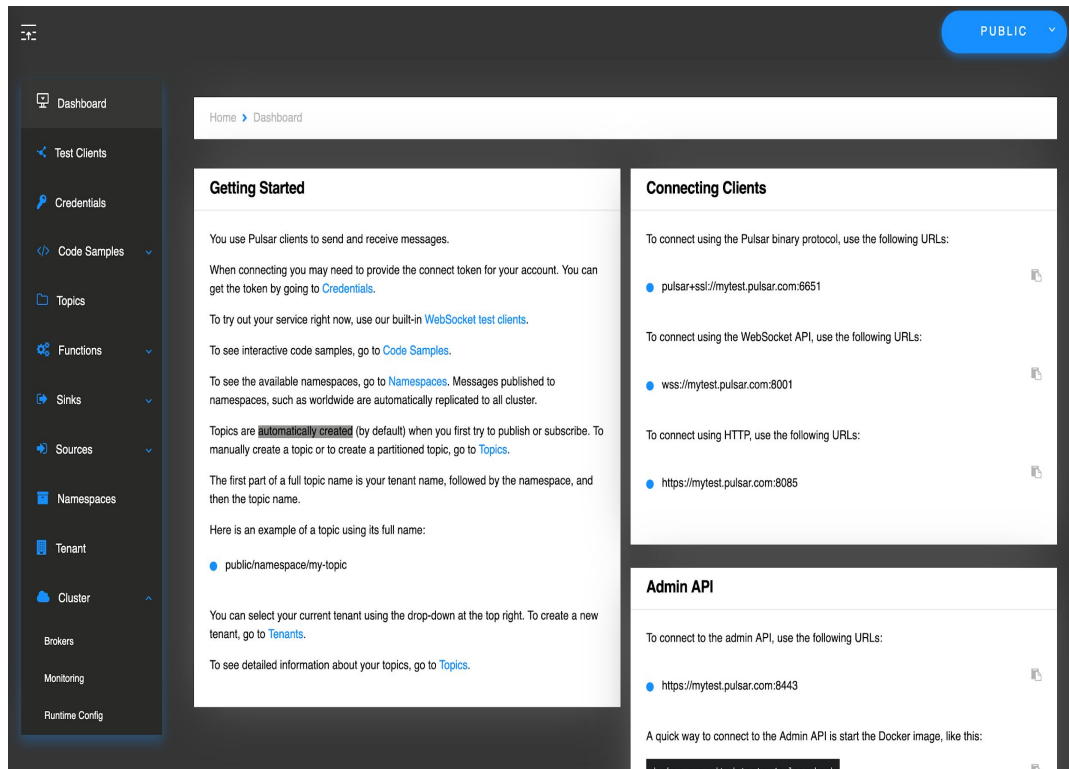
Other Components and Ecosystem, cont'd

- Metrics Monitoring
 - Expose metrics in **prometheus** format
 - `"/metrics"` at different ports (e.g. 8080 for broker, 8000 for zookeeper, etc.)
 - Pulsar Helm chart (for K8s) has embedded Prometheus and Grafana components
 - Built-in dashboards
 - Prometheus Alert-manager



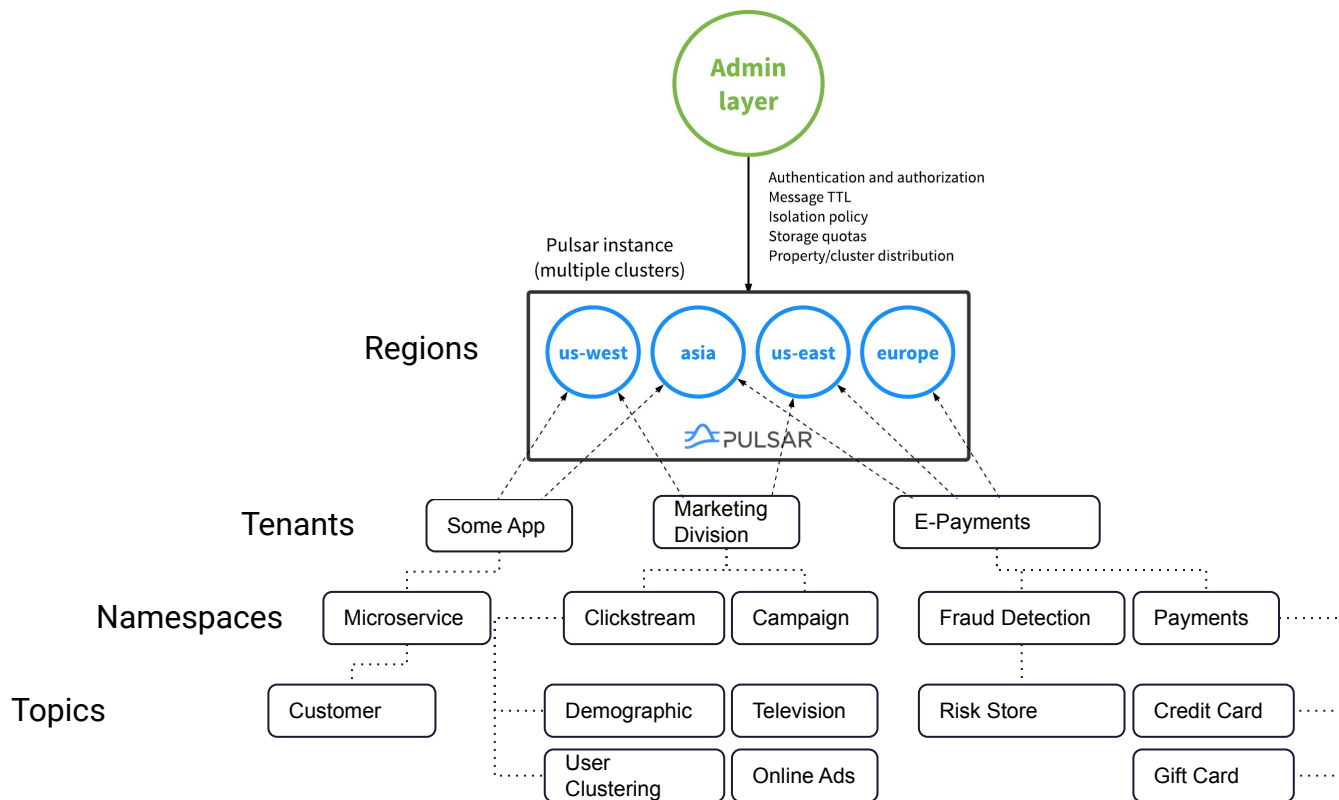
Other Components and Ecosystem, cont'd

- Pulsar Admin Console (DataStax)
- Pulsar Manager
 - Web based Pulsar cluster management tool
 - Able to manage multiple environments
- Admin CLI tools
 - Pulsar-admin
 - Pulsar-client
 - Pulsar-perf
 - ...



Multi-Tenancy in Pulsar

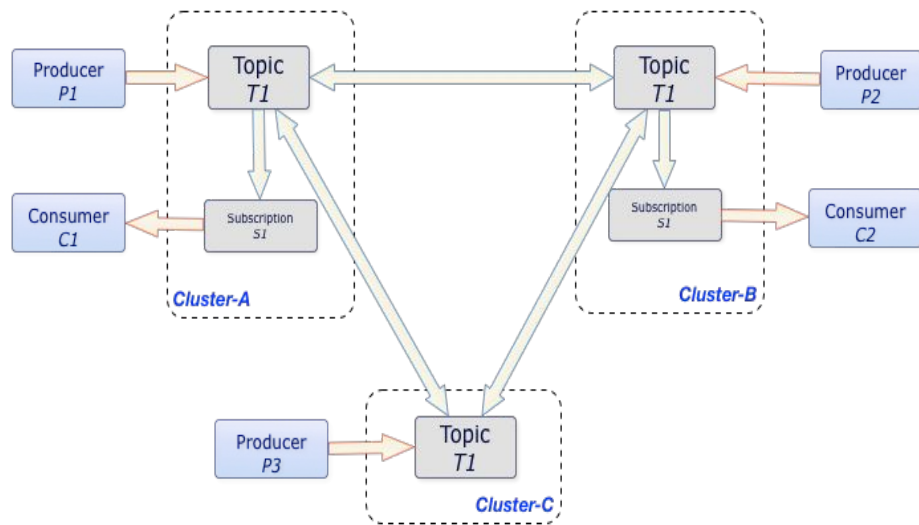
Multi-Tenancy



Geo-replication in Depth

Geo-Replication

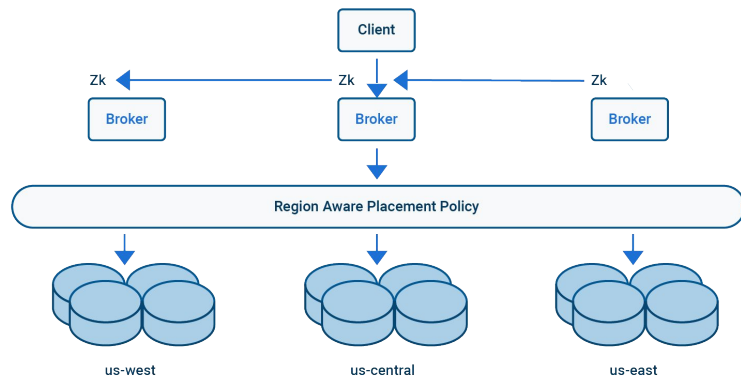
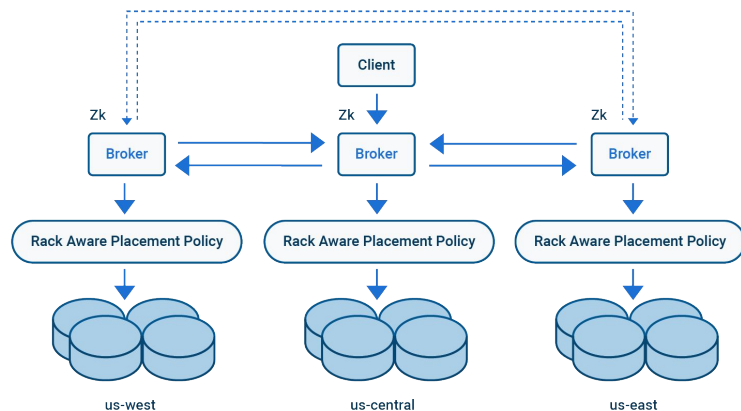
- Per-tenant basis
 - Only when a tenant has access to both clusters
- Enabled at namespace-level
 - Assign clusters to namespace
- Replicated Subscription
 - Keep subscription state in sync
 - Consumer failover to another cluster and resume from the failure point
 - Sub-second delay and possible duplicates



Geo-Replication, cont'd

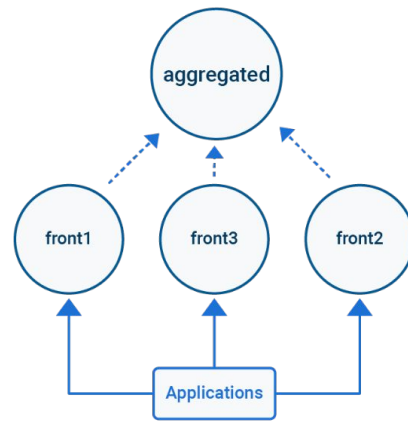
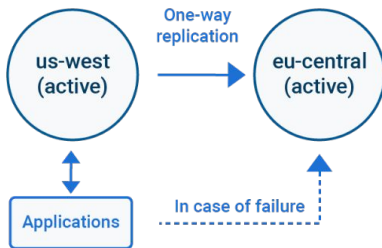
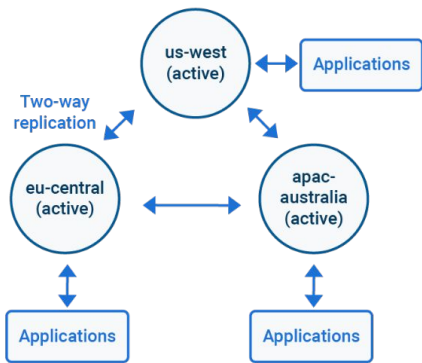
- Geo-replication
 - Local acknowledgement
- Synchronous Geo-replication
 - Global acknowledgement
 - Rack-awareness Policy
- Selective Geo-replication per message

```
List<String> restrictDatacenters =  
Lists.newArrayList("apac-australia");  
  
Message message = MessageBuilder.create()  
...  
    .setReplicationClusters(restrictDatacenters)  
    .build();  
  
producer.send(message);
```



Geo-Replication, cont'd

- Asynchronous Geo-replication pattern
 - Active-active
 - Active-standby
 - Aggregated replication
 - Edge computing
- Monitor replication statistics
 - Replication backlog
- Throttle replication



DS

LAB TIME!



dtsx.io/ess-lab1

DS

QUIZ TIME!

DataStax

SUBSCRIBE NOW: dtsx.io/ess-ep2

NEXT WEEK:

June 14th, same time!

Episode II: Pulsar in Action

Led By Aleks and Zeke

DataStax

THIS WEEK:

9th / 10th of June

Build a Netflix clone with GraphQL, React and NoSQL

Led By Cedrick, Ryan and Aleks

DS

THANK YOU!