
ONEWAY
SOLUTION

One Way Solution

The Fundamentals & Internals

Data Engineering – [Day1]

LUAN MORENO

CEO & Data Architect

Data Engineer & MVP

MATEUS OLIVEIRA

Big Data Architect

Data In-Motion Specialist


Either you run the day
or the day runs you.
Jim Rohn
quotefancy

Big Data & Streaming [Era]

Netflix, Inc.

137 million users worldwide with consumption of 25% of the world's internet bandwidth

Spotify

191 million users worldwide with more than 30 million of songs available

Airbnb, Inc.

income of $107 millions with average of 2m+ of people staying in places

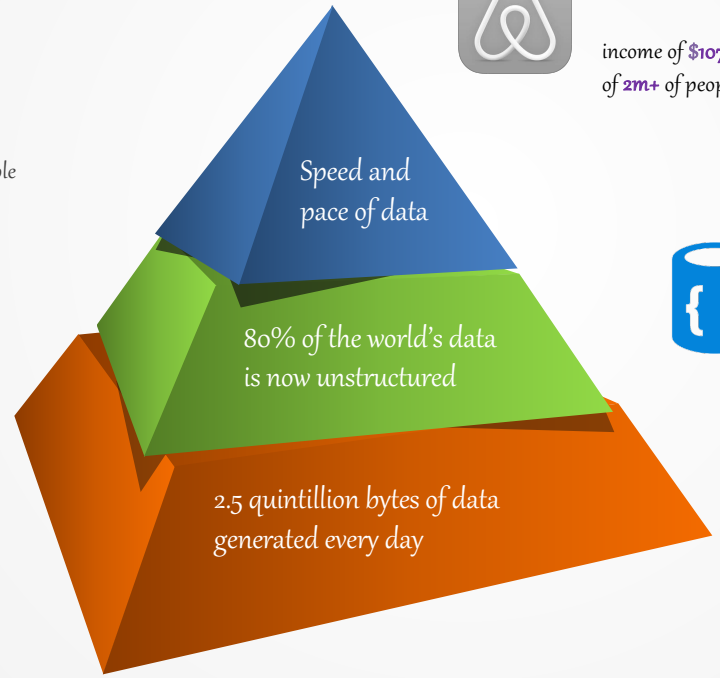
Lyft, Inc.

1m+ million of rides per day and 30M+ of users worldwide

Apache Kafka [2014]

speed of data generation & processing large datasets, ability to ingest data as fast as possible

Batch, Near, **Real-Time**



NoSQL [2009]

type & nature of data, different data sources & mappings, easier for developers

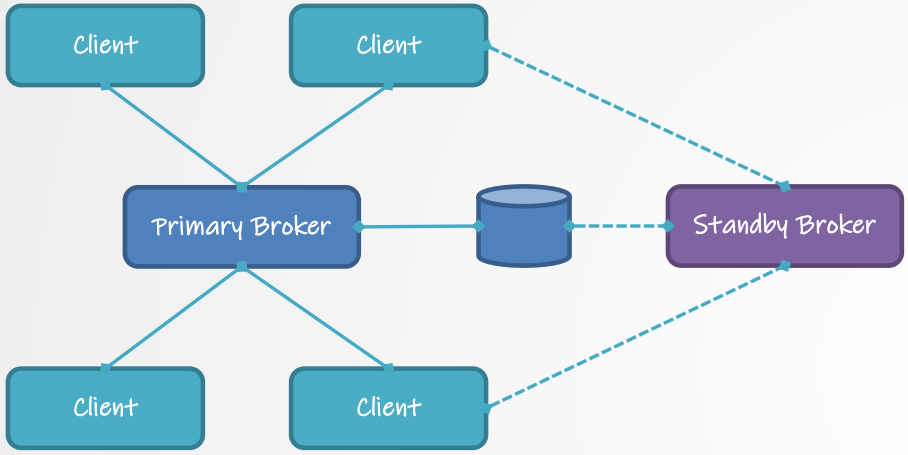
Key-Value Pairs, Column-Family, **JSON**, Graph

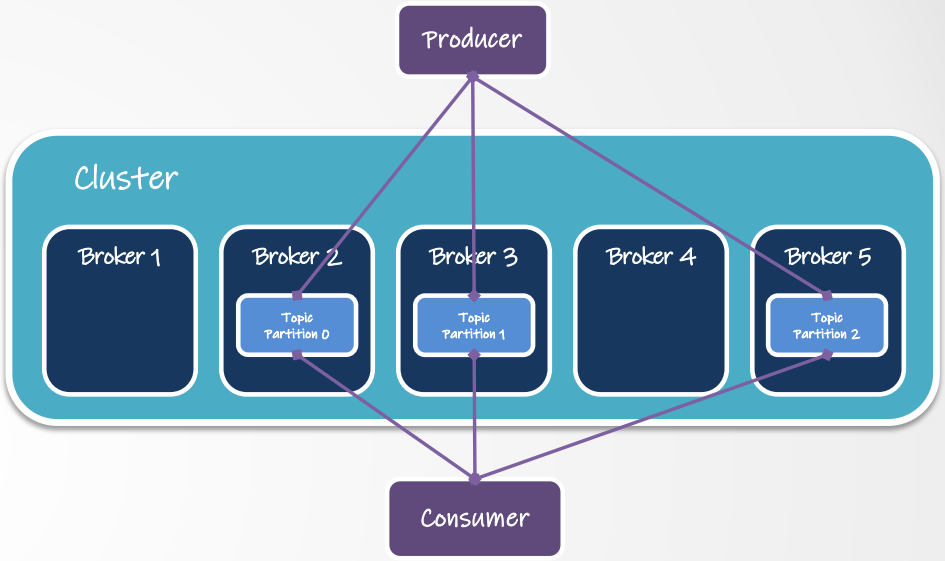
Apache Hadoop [2006]

quantity of generated & stored data, size of data & value of potential insights

MB | GB | TB | **PB**

Traditional Messaging Middleware [TMM] vs. Streaming Platforms [Message Broker]





Traditional Messaging Middleware

- Hub & Spoke Design
- Each Application Connects Only One Server at a Time
- Topics and Queues Reside in a Central Hub
- Bridged in a Larger Multi-Node Network

Considerations

- **Scalability** ~ Overwhelming Number of Requests (Microservices)
- **Durability** ~ Without Guarantee of Message Delivery (Network Issues)
- **Sync Communication** ~ Service Awaits Acknowledgement from System
- **Coupled Architecture** ~ Tightly Coupled

Message Broker

- Intermediary Program
- Store, Route, Monitor & Manage Messages
- Producer & Consumer ~ Write & Read Messages

Considerations

- **Scalability** ~ Middleman, Scale Vertically & Horizontally
- **Durability** ~ Persisted On-Disk & Acknowledged
- **Async Communication** ~ Without Waiting Consumer
- **Decoupled Architecture** ~ Producers <> Consumers

RabbitMQ

- One of Most Popular Open-Source **Message Broker System**
- Accepts, Stores & Forward Binary Blobs of Data ~ Messages
- Distributed & Federated Configurations
- Lightweight & Deployed Anywhere

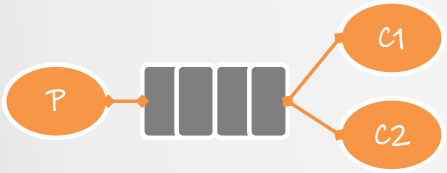
- Send Messages ~ Queue
- Host Memory & Disk Limits
- Large Message Buffer
- Waits for Messages

- Time-to-Live & Expiration
- Persistence Configuration
- Queue Length Limit
- Dead Letter Exchanges

Streams

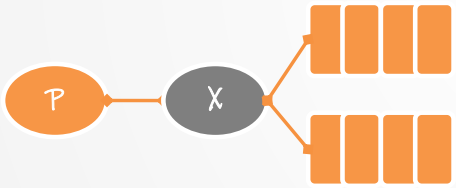
- **July 13, 2021**
- Append-Only Log
- Persisted & Replicated
- Leader & Replica
- Java Client
- At-Least-Once

Work Queues



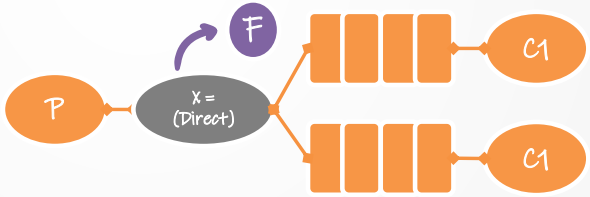
- Distribute Time-Consuming Tasks using Multiple Workers
- Avoid Resource-Intensive Task Immediately
- Schedule Activity for Later Processing
- Worker Process Running in Background Mode
- **Used for Web Applications ~ HTTP Request Window**

Pub & Sub



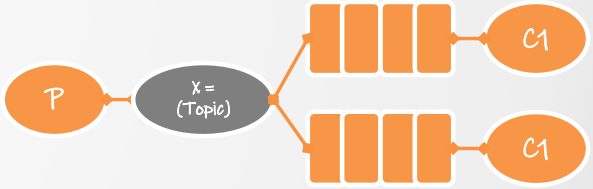
- Producer Never Sends Any Messages Directly ~ Queue
- Message is Sent to an **Exchange**
- Action of Message Defined by Exchange Types
- Direct, Topic, Headers & Fanout
- **Broadcast Messages to Many Receivers**

Routing



- Subscribe Only ~ Subset of Messages
- Direct Only Critical Error Messages ~ Log File
- Queue Interested in Messages ~ Exchange
- **Filtering Messages Based in an Argument**

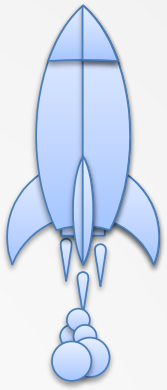
Topics



- Valid Routing Key
- Matching Binding
- **Bindings ~ # & ***

Apache Pulsar

- Originally Created at **Yahoo**
- Publish & Subscribe Pattern using Topics for Event Retention
- Cloud-Native & Distributed Messaging & Streaming Platform
- Top-Level Apache Software Foundation Project
- Local, Docker & Kubernetes Installation



- Native Support for Multiple Clusters in a Pulsar Instance
- Seamless **Geo-Replication** of Messages ~ Clusters
- Topic **Compaction** Period
- Low Publish & End-to-End Latency
- **Bindings** for Java, Go & Python
- Multiple Subscription Modes ~ **Exclusive, Shared & Failover**
- Persistent Message Storage Provided **By Apache BookKeeper**
- Light-Weight Computing Framework ~ **Pulsar Functions**
- Serverless Connector Framework ~ **Pulsar IO**
- **Tiered Storage** for Hot, Warm, Cold & Long-Term Retention
- Deduplication & Effectively-Once Semantics ~ **EOS**
- **Pulsar Schema Registry** for Type Safety ~ Client & Server Side
- Transactions (TXN) ~ Consume, Process and Produce Atomically

Producer

- Send Events and Publish ~ Topic in a Pulsar Broker
- Producer Idempotency
- Send Modes = Sync and Async
- **Access Mode** = Shared & Exclusive
- Compression, Batching, Chunking

Consumer

- Subscribe in a Topic via Subscription
- Receive Modes = Sync and Async
- **Positive** Acknowledgment = Individually & Cumulatively
- **Negative** Acknowledgment = Out of Order Consume
- Timeout, Dead Letter Topic, Retry
- Subscription = Exclusive, Shared, Failover & Key Shared
- Delayed Message Delivery

Topics

- Topics = Channels for Transmitting Messages
- Persistent and Non-Persistent Topics
- **Routing Modes** = Round Robin, Single, Custom
- Immediately Delete Consumed Messages
- Backlog Unacknowledged Messages
- Message Deduplication Feature = EOS
- **Namespace** = Logical Nomenclature within a Tenant

IO

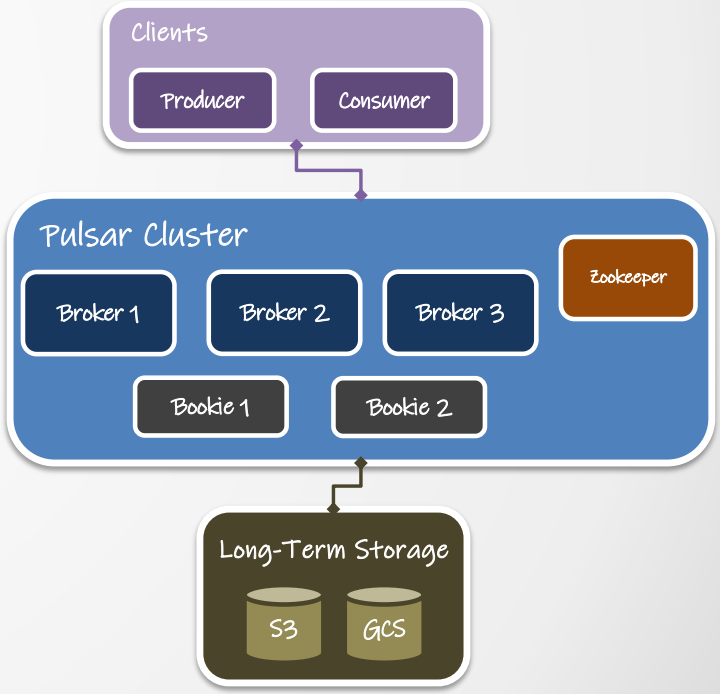
- Connect with External Systems
- Source & Sink Types of Connector
- MySQL, Postgres, MongoDB, RabbitMQ
- Cassandra, ElasticSearch, Apache Kafka
- **Processing Guarantees** ~ At-Most, At-Least & Effectively Once

Functions

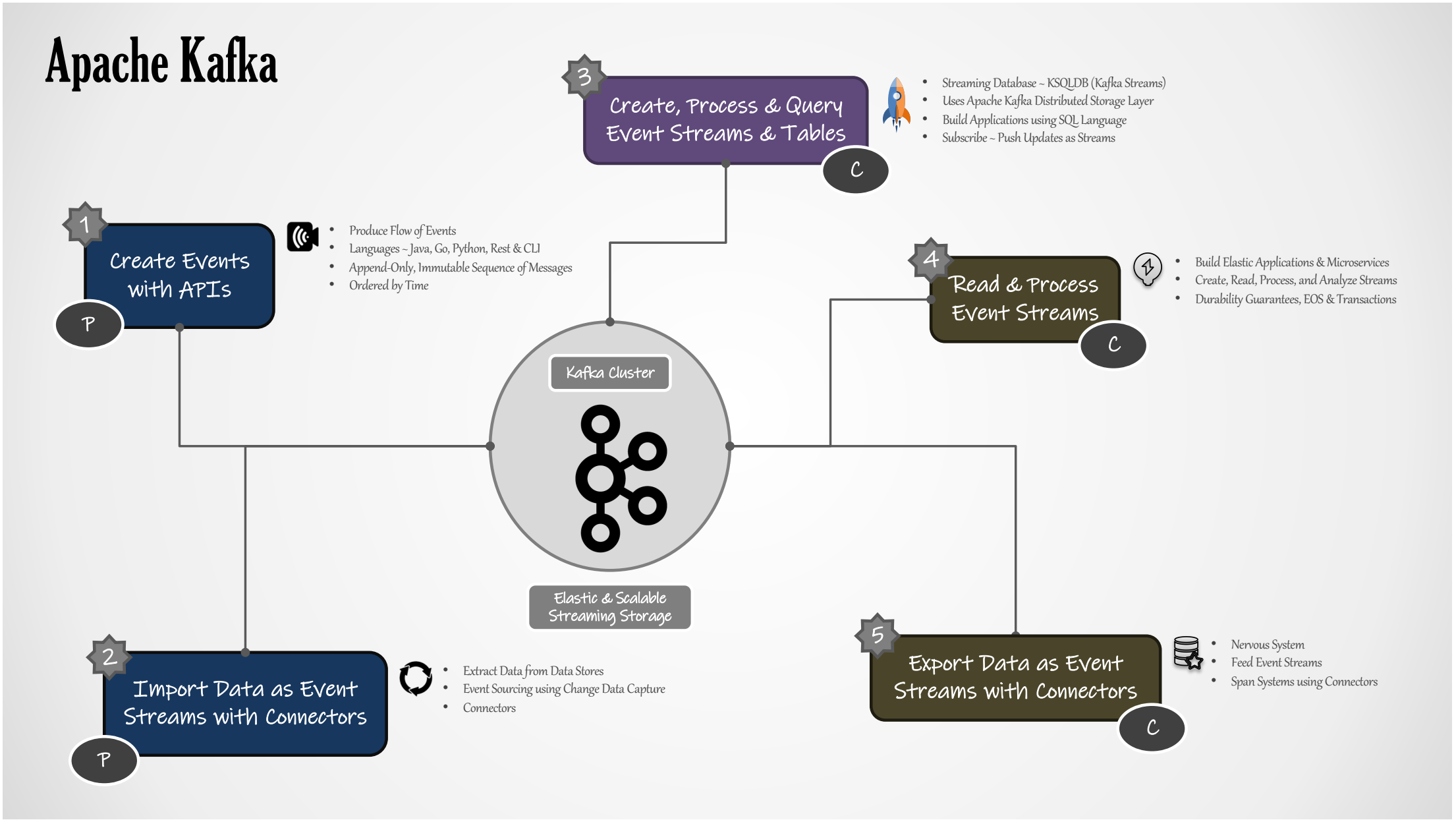
- Lightweight Compute Process
- **Consume, Apply Logic & Publish Results ~ Topic**
- Without Neighboring System Problem
- Operational Simplicity ~ Without External Processing
- Inspired By Storm, Flink, Lambda, Cloud & Azure Functions
- **Processing Guarantees** ~ At-Most, At-Least & Effectively Once

SQL

- **Query using Trino**
- Store Structured Data in Pulsar
- Connector Enables Multiple Workers
- **Data Stored** in Apache BookKeeper



Apache Kafka



RabbitMQ vs. Apache Pulsar vs. Apache Kafka

RabbitMQ

- Traditional Messaging System for Pub & Sub
- Components ~ RabbitMQ
- Consumption Model ~ Push
- Storage Architecture ~ Index
- Programming Languages Supported ~ 22
- Stack Overflow Questions ~ 11.430
- Job Mentions in USA ~ 536

General Purpose Message Broker System with Rich Developer Experience

Apache Pulsar

- High-Performance Solution
- Components ~ Pulsar, Zookeeper, BookKeeper & RocksDB
- Consumption Model ~ Push
- Storage Architecture ~ Index
- Programming Languages Supported ~ 6
- Stack Overflow Questions ~ 2.332
- Job Mentions in USA ~ 23

Cloud-Native, Richest Feature Set & Gaining Traction on Market

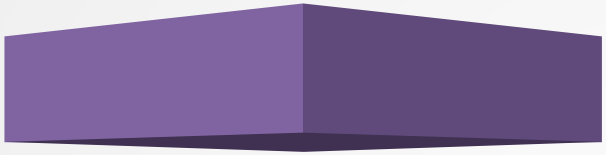
Apache Kafka

- Pure Distributed Log Designed for Efficient Event Streaming at High-Scale
- Components ~ Kafka, Zookeeper (KIP-500)
- Consumption Model ~ Pull (Higher Throughput)
- Storage Architecture ~ Log
- Programming Languages Supported ~ 17
- Stack Overflow Questions ~ 21.333
- Job Mentions in USA ~ 4.293

Distributed Event Streaming Platform for Microservices, Big Data & Analytics


Always turn a negative situation
into a positive situation.
Michael Jordan
quotefancy

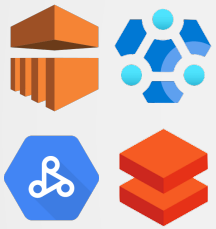
Big Data Eras

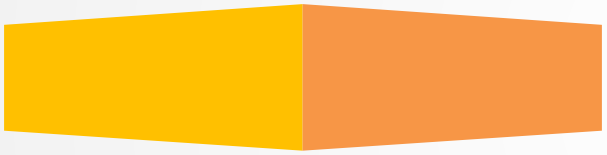


Kubernetes as Cornerstone Solution for Apps & Data ~ [2018]

- Multi-Cloud Strategy First
- Use of Managed Kubernetes Solutions with 99.99%
- Simplify Operations By using Kubernetes as Infrastructure Backbone
- Microservice Driven Approach ~ Containerize Environments
- Cheapest Solution for Application, DataStore & Big Data Solution

Cons ~ Steep Learning Curve, Mindset Change

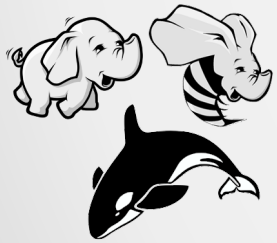


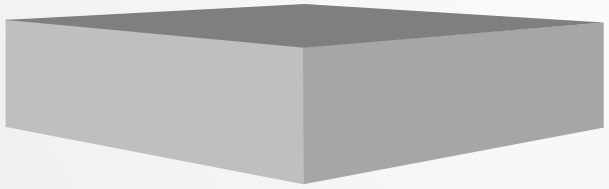


Use of PaaS & SaaS Software's ~ [2014]

- Solution Managed by Cloud Vendor
- Big Data as a Service ~ BDaaS
- Decoupling Computation from Storage
- Pay for Use Only
- Strategy to Reduce Overall Big Data Costs

Cons ~ Promise of Cost Reduction, Myriad of Options





Big Data on Premises using Vendors ~ [2009]

- Cloudera, MapR & HortonWorks
- Based on Open-Source Technologies
- On-Premises Data Centers using Physical Hardware
- Expensive & Dependency Wise
- Large Operations Team ~ Manage Clusters

Cons ~ Expensive, High-Level of Complexity, Burdensome

Real-Time [Data Ingestion]

Open-Source Platform [OSS]

- Apache Kafka

Microsoft Azure

- Azure Event Hubs

Google Cloud Platform [GCP]

- Cloud Pub/Sub

Amazon Web Services [AWS]

- Amazon Kinesis Data Streams
- Amazon Kinesis Firehose

Microsoft Azure vs. GCP vs. Amazon Web Services for Streaming Offerings

Azure Event Hubs

Event Hubs is a Fully Managed, Real-Time Data Ingestion Service

- Ingest Millions of Events per Second
- Enable Real-Time and Micro-Batch Processing Concurrently
- Managed Service with Elastic Scale
- Serverless Streaming Solution
- Auto Inflate & Capture Retention

Google PubSub

Messaging & Ingestion for Event-Driven Systems and Streaming Analytics

- Scalable, In-Order Message Delivery with Pull and Push Modes
- Auto-Scaling and Auto-Provisioning with Support from Zero ~ Hundreds of GB/Sec
- Independent Quota and Billing for Publishers and Subscribers
- Global Message Routing to Simplify Multi-Region Systems

Amazon Kinesis

Amazon Kinesis Offers Key Capabilities ~ Cost-Effectively Process Streaming Data at Any Scale

- Amazon Kinesis Video Streams ~Securely Stream Video
- Amazon Kinesis Data Streams is a Scalable and Durable Real-Time Data Streaming Service
- Amazon Kinesis Data Firehose ~ Easiest Way to Capture, Transform, and Load Data Streams
- Amazon Kinesis Data Analytics ~ Easiest Way to Process Data Streams with SQL or Apache Flink

Real-Time [Data Processing]

Open-Source Platform [OSS]

- Apache Kafka
- Apache Spark
- Apache Flink
- Apache Samza

Microsoft Azure

- Azure Stream Analytics
- Azure HDInsight
- Azure Databricks

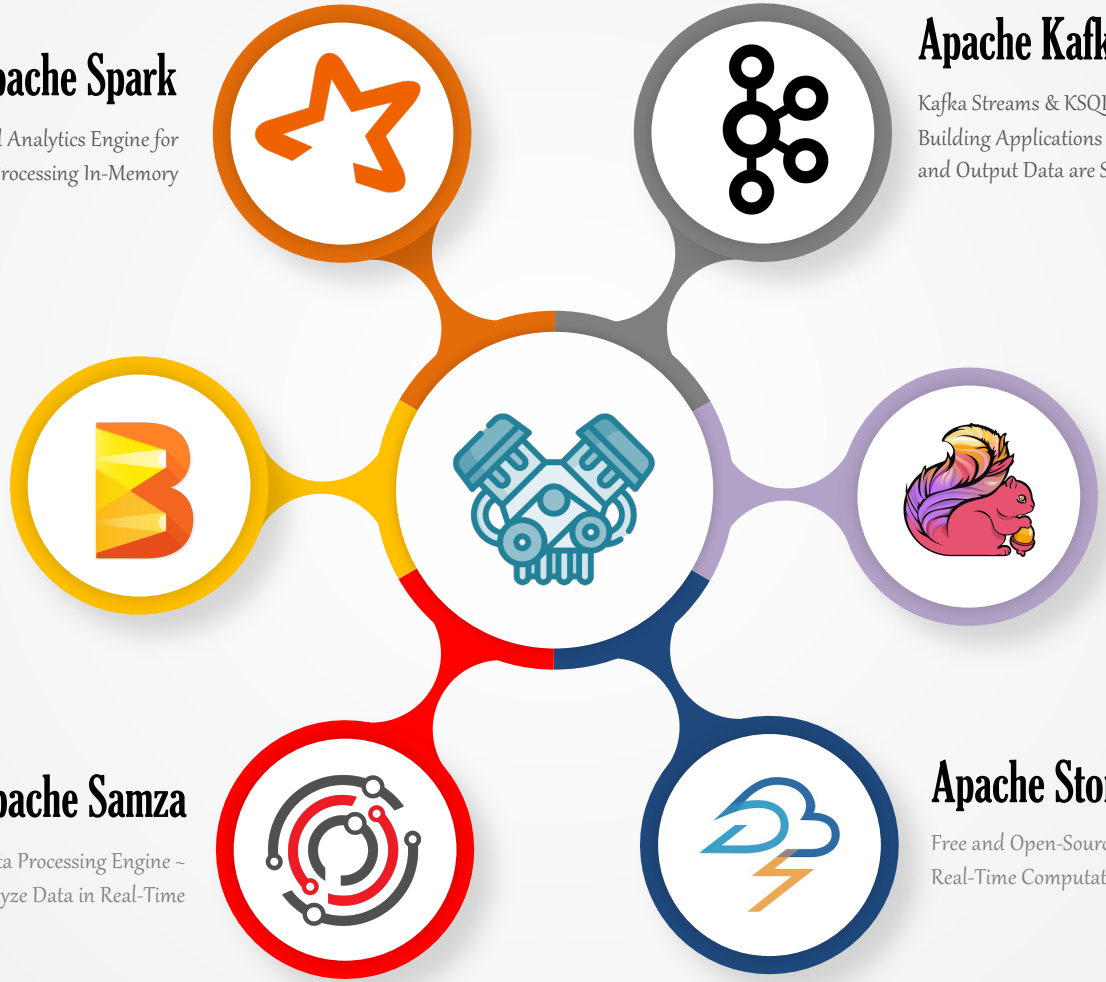
Google Cloud Platform [GCP]

- Cloud DataFlow
- Google DataProc
- Databricks

Amazon Web Services [AWS]

- Amazon Kinesis Data Analytics
- Amazon EMR
- Databricks

Streaming Platform Engines for Data Processing



Apache Spark

Open-Source Unified Analytics Engine for Large-Scale Data Processing In-Memory

Apache Kafka

Kafka Streams & KSQLDB ~ Client Library for Building Applications and Microservices, for Input and Output Data are Stored in Kafka Clusters

Apache Beam

Open-Source, Unified Model for Defining Batch & Streaming Data-Parallel Processing Pipelines

Apache Flink

Framework and Distributed Processing Engine for Stateful Computations over Unbounded and Bounded Data Streams

Apache Samza

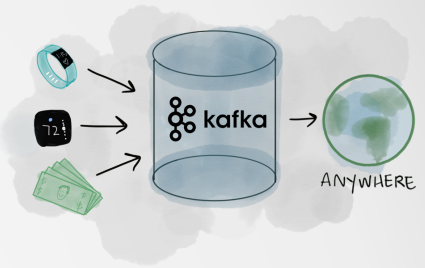
Scalable Data Processing Engine ~ Process and Analyze Data in Real-Time

Apache Storm

Free and Open-Source Distributed Real-Time Computation System


All things are difficult before they are easy.
Thomas Fuller
quotefancy

Apache Kafka [Fundamentals]



Apache Kafka

- Open-Source Software – [Scala & Java]
- 7,500 Companies using Apache Kafka
- + 200 Connectors on Confluent Hub
- De-Facto Streaming Platform of Fortune 100 [80%]
- High-Throughput & Low-Latency Real-Time Data Feeds
- Storage & Processing System using a Distributed Transaction Log

Performance

- Latency < 10 milliseconds [Apache Kafka]
- > 4.5 Trillion of Messages per Day [LinkedIN]
- > 1 Trillion of Messages per Day & 6 PB of Data [Netflix]

History

- Franz Kafka ~ Apache Kafka
- Developed by LinkedIN
- Open-Sourced in 2011
- Jun Rao | Jay Kreps | Neha Narkhede ~ Confluent

Core APIs

- Producer API – Publish Streams of Records
- Consumer API – Subscribe Topic
- Connector API – Reusable Producers & Consumers
- Streams API – Java & Scala Processor
- KSQL API – SQL Processor

Stream Processing Systems

Key Capabilities

- Publish & Subscribe Streams of Records [Producer & Consumer]
- Store Streams of Records in a Fault-Tolerant Durable Way [Cluster & Storage]
- Process Streams of Records [Kafka Streams & KSQL]

Confluent Platform

Commercial

Free Forever on a Single Kafka Broker

- Tiered Storage
- Operator
- Role-Based Access Control
- Structured Audit Logs
- Secret Protection
- Schema Validation
- Support 24x7x365
- Commercial Connectors
- Control Center
- Multi-Region Clusters
- Replicator
- Auto Data Balancer

Community

Free Forever

Kafka Streams

Client Library for Building Applications & Micro-Services
Input & Output Data are Stored in Kafka Clusters
Writing and Deploying Standard Java & Scala Apps

KSQLDB

Streaming SQL Engine for Apache Kafka
Powerful & Easy-to-Use Interactive Query
Operations – Filtering, Aggregation, Joins, Windowing

Schema Registry & REST Proxy

Serving Layer for Metadata using Apache Avro
Stores & Versioned History of All Schemas
RESTful Interface to Produce & Consume Data

Apache Kafka

Streaming Platform
Publish & Subscribe Streams of Records
Store Streams of Records in a Fault-Tolerant Durable Way

Kafka Producer & Consumer

Producer API = Allows an App to Publish a Stream of Records
Consumer API = Allows an App to Subscribe to Topics
Clients = Java, Scala, Python, C++, Go & .Net

Kafka Connect & Connectors

Tool for Scalably & Reliably Streaming Data into Kafka
Connectors Developed by Community & Supported by Confluent
Standalone & Distributed Modes

Apache Kafka History

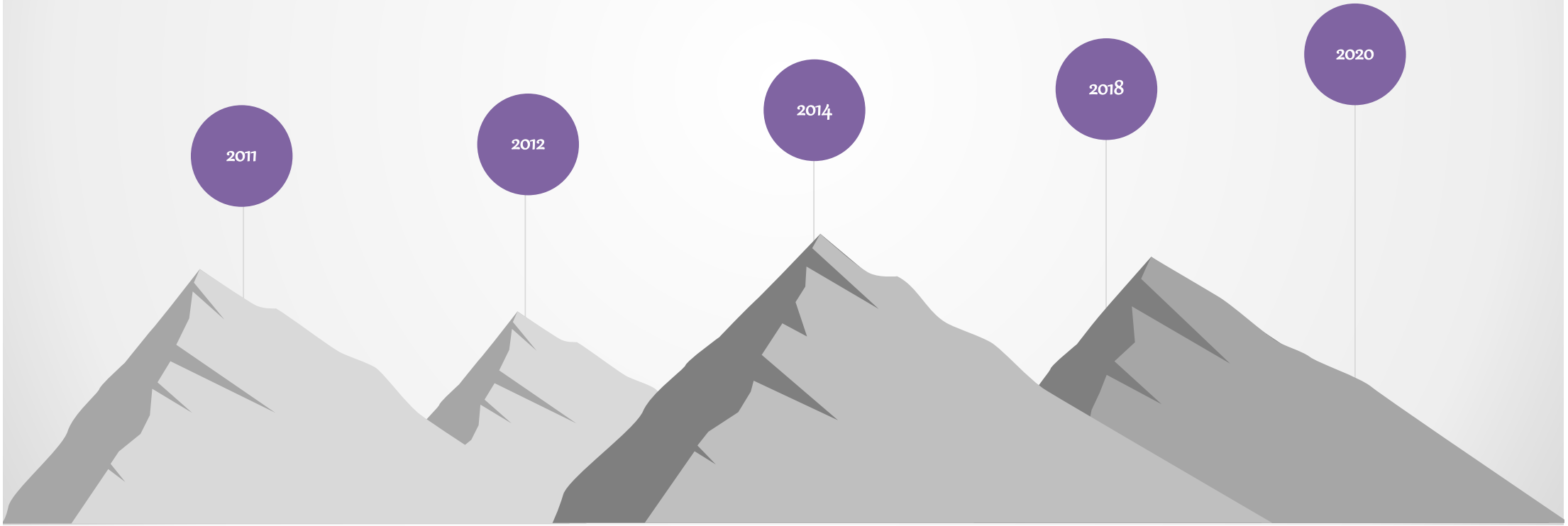
The Born
Originally Developed by LinkedIN and Open-Sourced.

The Graduation
Graduation from Apache Incubator

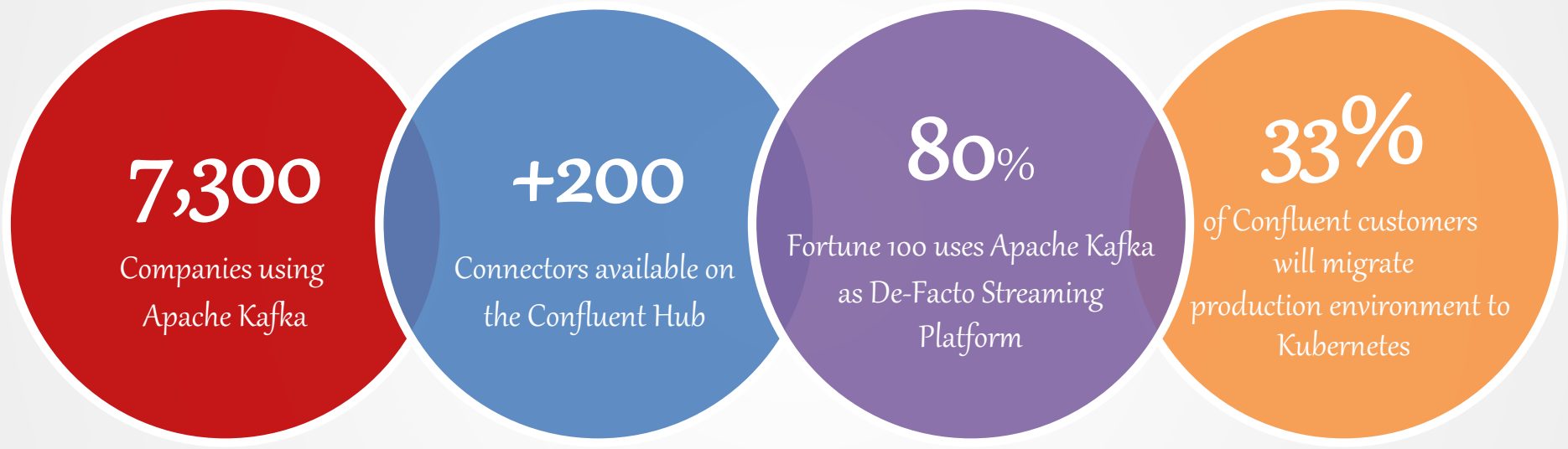
The Raise
Jun Rao, Jay Kreps, & Neha Narkhede, Announces Confluent

The Funding
Confluent Raises a $125M Series D Funding Round.

The Leader
Raises $250M Series E & Evaluation of $4.5B



Apache Kafka [Quick Stats]



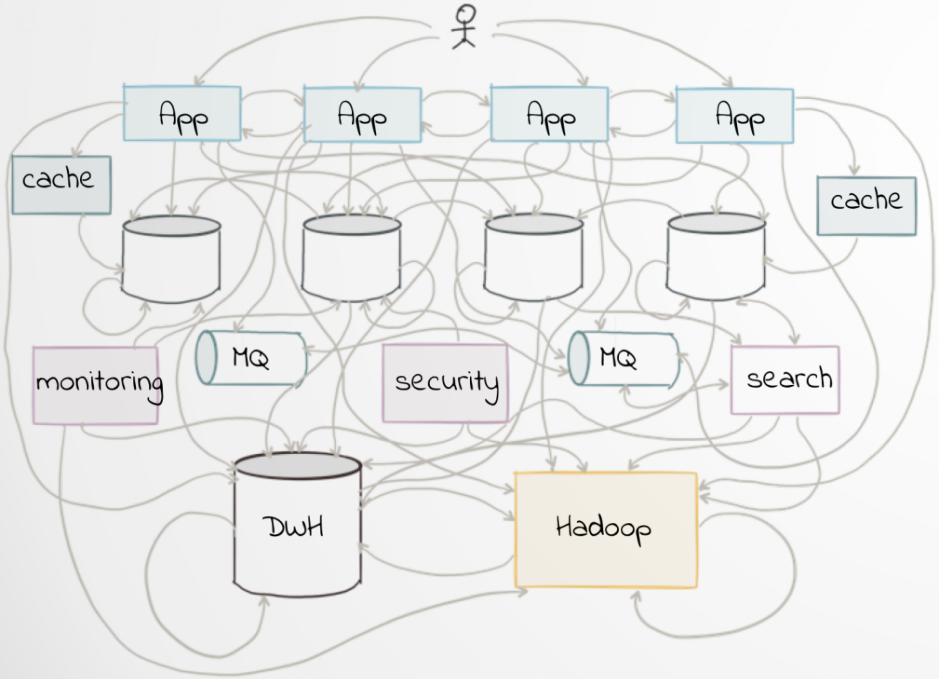
Apache Kafka [De-Facto Streaming Platform]

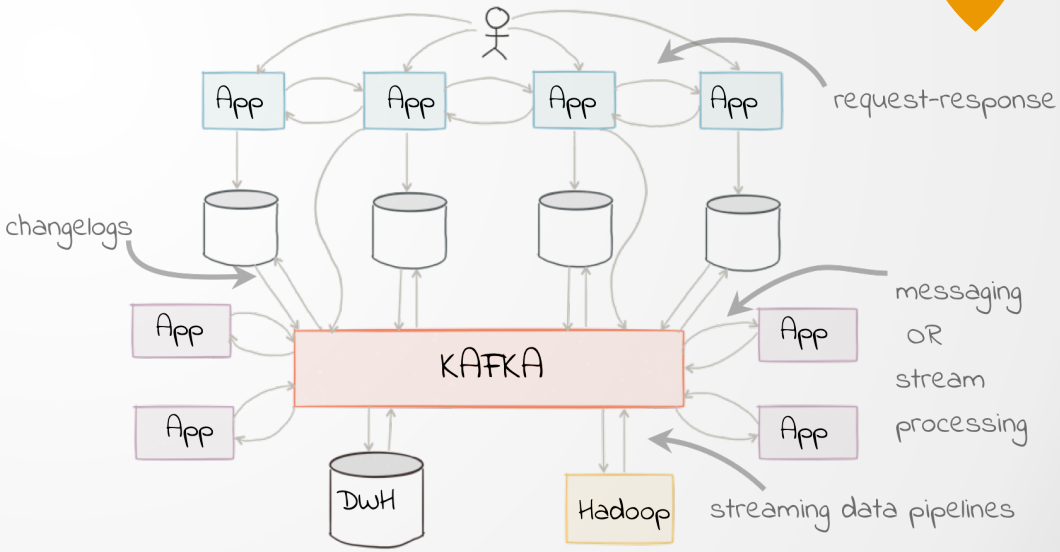
< Apache Kafka

Data Stored in a Variety of Places
Difficult for Data Integration
Dramatic Business Impact
Siloed Data

> Apache Kafka

Centralized Data Exchange Hub
Data Sent in Binary
Fast & Reliable Architecture
Commit-Log Structured
Easy for Data Integration





Apache Kafka [Deployment Options]

Confluent Cloud

- Re-Engineered for Cloud Computing
- Throughput Limit = 100 MBps & Storage Limit = 5 TB
- Schema Registry, KSQL, Connectors
- **Current Version = Latest**

Strimzi

- Apache Kafka on Kubernetes [Operator]
- NodePort, Load Balancer & Ingress
- Broker, Zookeeper, Connect, MirrorMaker, Exporter, Bridge
- TLS & SCRAM-SHA
- **Current Kafka Version = 3.1**

Apache Kafka

- Open-Source Stream-Processing Software
- Publish & Subscribe
- Process & Store
- **Current Version = 3.1**

Confluent Platform & Operator

- Event-Streaming Platform [Apache Kafka]
- Kafka Streams & Kafka Connect
- Clients [C++, Python, Go & .Net]
- Connectors, KSQL, Schema Registry, REST Proxy
- Terraform & Ansible Playbooks
- **Current Confluent Version = 7.0.1**
- **Current Kafka Version = 3.0**

Amazon MSK

- Managed Streaming for Apache Kafka
- Fully Compatible & Managed
- Elastic Stream Processing [Apache Flink]
- High Available & Secure
- Process & Store
- **Current Version = 2.8.1**

Azure HDInsight

- Cost-Effective and Enterprise Grade
- SLA of 99.9%
- Azure Managed Disks [16 TB]
- **Current Version = 2.1.1 [HDI 4.0]**

IaC [Infrastructure as Code] & CM [Configuration Management]

IaC

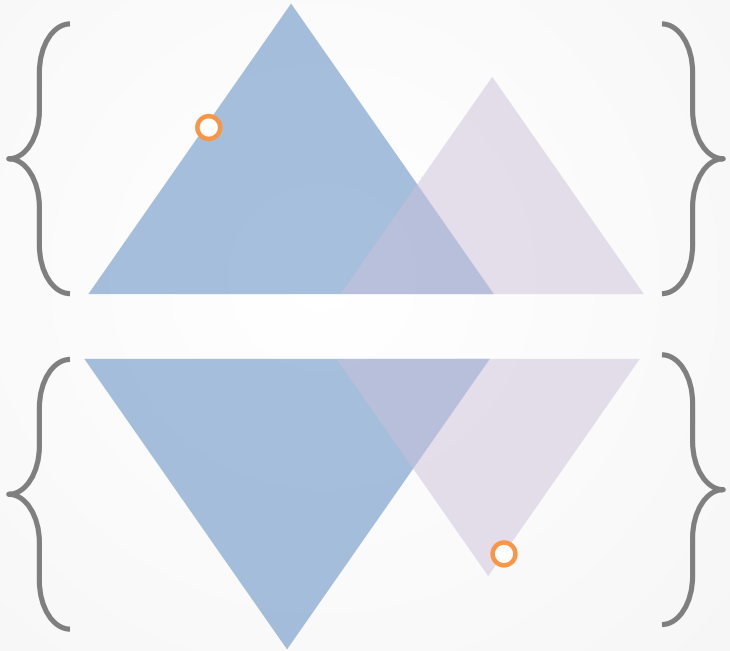
- Write Code to Provision, Manage & Deploy IT Infrastructure
- Manage Cloud, Infrastructure & Services
- Modular Development & Agile

Terraform

- Immutable Infrastructure Approach
- Tool for Building, Changing & Versioning Infrastructure
- Configuration Files Describe Terraform Components
- Generates a Plan & Reach a Desired State
- Terraform Init, Plan, Apply, Show

Pulumi

- Modern Infrastructure as Code for Developers & Infrastructure Teams
- Build, Deploy, and Manage Modern Cloud Applications
- Infrastructure using Familiar Languages
- TypeScript, JavaScript, Python, GO, C#



CM

- Stablishing & Maintaining Consistency of a Product
- Always Up-to-Date Repository of Assets
- Easy Recoverable & Resumable Workspace

Ansible

- Simplest Solution for Configuration Management
- Minimal in Nature, Secure & High Reliable
- Low Learning Curve for Administrators & Developers
- Human-Readable
- Agentless [OpenSSH]

Apache Kafka vs. Cloud Offerings Comparison

Apache Kafka

- ✓ Manual Installation & Management
- ✓ Broker, Zookeeper & Connect Components
- ✓ Install in a VM or Bare-Metal Machine

Confluent Cloud

- ✓ Fully Managed By Confluent ~ SaaS Platform
- ✓ Available in Cloud Providers ~ GCP, AWS & Azure
- ✓ Broker, Zookeeper, Connect, Schema Registry, KSQLDB and Control Center
- ✓ Available Version of Apache Kafka = 3.1

AWS Managed Streaming for Apache Kafka

- ✓ Fully Managed By AWS
- ✓ Broker and Zookeeper Components
- ✓ EM2 VM For Bastion Server
- ✓ Available Version of Apache Kafka = 2.8.1

Azure HDInsight

- ✓ Fully Managed By Microsoft Azure
- ✓ Broker and Zookeeper Components
- ✓ Available Version of Apache Kafka = 2.4.1


Life begins at the end of your comfort zone.
Neale Donald Walsch
quotefancy

Virtual Machines [VMs] vs. Containers

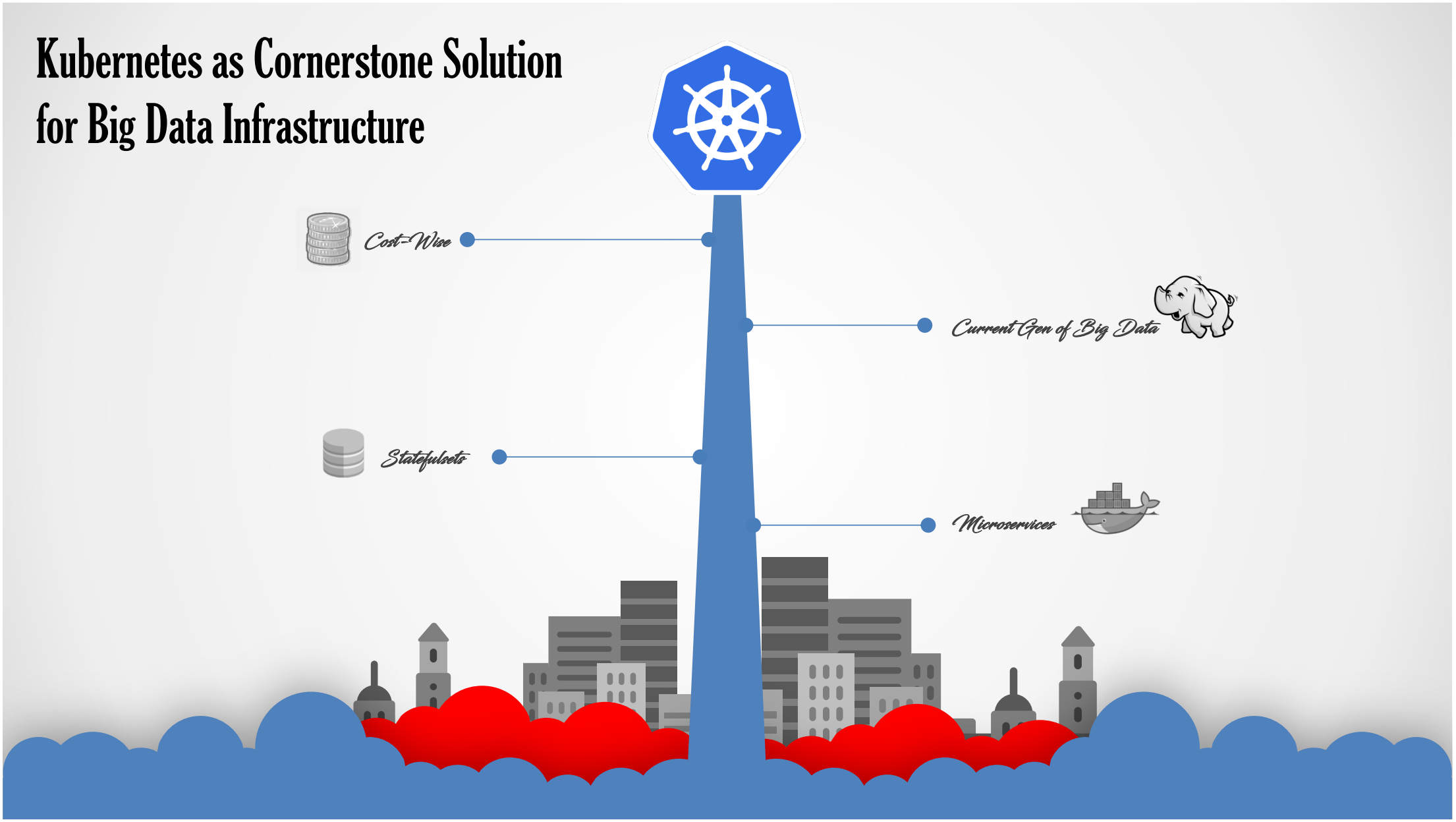
Virtual Machines [VM]

- Abstraction of Physical Hardware
- Hypervisor ~ Multiple VMS in a Single Machine
- Machines ~ Full Copy of an Operating System [Binaries & Libraries]
- GB in Size & Slow to Boot

Containers

- Abstraction of an Application [App]
- Packages Code & Dependencies
- Multiple Containers ~ Same Machine & Share OS Kernel
- MBs in Size & Fast Boot

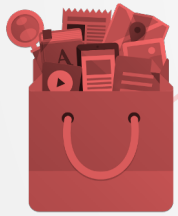
Kubernetes as Cornerstone Solution for Big Data Infrastructure



Strimzi Operator

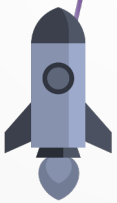
Apache Kafka on Kubernetes

Strimzi Simplifies ~ Process of Running Apache Kafka in a Kubernetes Cluster. Apache Kafka is an Open-Source Distributed Publish-Subscribe Messaging System for Fault-Tolerant Real-Time Data Feeds



All Kafka Goodies

Deployments Available: Kafka Connect, Kafka Connectors, Kafka MirrorMaker, Cruise Control, Apache Kafka & Kafka Bridge, Managed By Strimzi Operator as Kubernetes Resources (CRD)

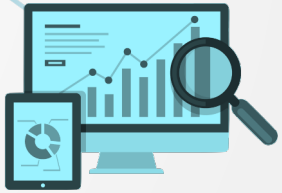


Scale on Demand

Scale Your Kafka Broker or Resources Independently, YAML File Based for Changing Deployment Spec

Heavy Power with Few Resources

Resources Based in Container Images, Working with Resources Such 6 GB of Memory and Achieving +100K/TPS. Reduction of CPU and Memory Footprint & Optimized



Observability

Fully Integrated with Prometheus and Grafana, 360° View of Heath of Your Deployment

Kubernetes & Cloud Computing

Open-Source Platform [OSS]

- Kubernetes [K8S]

Microsoft Azure

- AKS – Azure Kubernetes Service

Google Cloud Platform [GCP]

- GKE – Google Kubernetes Engine

Amazon Web Services [AWS]

- EKS – Elastic Kubernetes Service

Installing Apache Kafka on Kubernetes ~ [AKS] + Strimzi Operator

1. Digital Ocean Overview
2. Installing Strimzi using Helm
3. Verify Kubernetes Deployment
4. Scale-Up & Scale-Down Deployment

Apache Kafka - Enterprise Data Hub [EDH] - Product Cost Comparison

Amazon Managed Streaming for Apache Kafka [MSK]

Fully Managed Apache (Kafka Broker)
AWS Glue Schema Registry
Custom MSK Configuration

Cost Model

- **Region** – EastUS
- **Instance** – kafka.m5.large
- **Spec** – 2 vCPUs & 8 GB of RAM
- **VMs** – 3
- **Storage** – 1 TB
- **Cost** – USD 562

- R$ 3.194

Azure HDInsight - Apache Kafka

Fully Managed Apache (Kafka Broker)
Open-Source Distributed Streaming Platform
Up To 16 TB of Storage per Broker

Cost Model

- **Region** – EastUS
- **Instance** – A2M
- **Spec** – 2 vCPUs & 16 GB of RAM
- **VMs** – 3
- **Storage** – 1 TB
- **Cost** – USD 731

- R$ 4.149

Confluent Cloud

Cloud-Native Service for Apache Kafka
Apache Kafka Broker Pricing (Base)
Self-Service Provisioning
Elastic Scaling

Cost Model

- **Region** – EastUS
- **Kafka Base** – $1.50 per Hour
- **Total Hours** – 744 Hours
- **Storage** – 1 TB
- **Cost** – USD 1.156

- R$ 6.571

Strimzi Apache Kafka Operator

Kafka on Kubernetes
Kafka, Zookeeper, Kafka Connect,
Kafka MirrorMaker, Kafka Exporter,
Kafka Bridge & Schema Registry

Cost Model

- **Region** – EastUS
- **Instance** – D2V3
- **Spec** – 2 vCPUs & 8 GB of RAM
- **VMs** – 3
- **Storage** – 1 TB
- **Cost** – USD 251

- R$ 1.426

Total Annual Cost for Apache Kafka

- **Confluent Cloud** = R$ 78.852
- **Azure HDInsight** = R$ 49.788
- **Amazon MSK** = R$ 38.328
- **Strimzi Operator** = R$ 17.112

If opportunity doesn't knock, build a door.

Milton Berle

Apache Kafka ~ [The Log Structure]

Logs in a Database

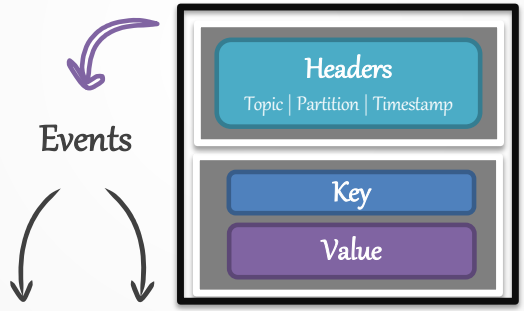
- Data Structured & Indexes
- Immediately Persisted on Disk
- ACID [Atomic, Consistency, Isolation & Durability]
- Master & Slave [Sync of Log File]

Logs in a Distributed System

- State Machine Replication Principle [== State, Input, Order, Output, End Result]
- Implement Multiple Machines with a Consistent Log
- Log Entry Act as a Clock for [State of Replicas] = Timestamp
- Active-Active Model with a Primary-Backup [Leader & Replica]

Writing into Log

- Data is Written in Sequential Order [Binary]
- Batch-Operation ~ Prefetch
- Sequential Writes
- Data is Persisted on Disk



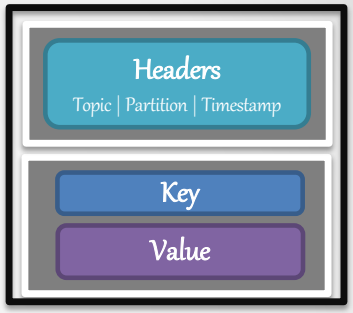
Reading from Log

- Passthrough JVM [Java Virtual Machine]
- Copy from Disk Buffer to Network Buffer [Zero Copy]
- Sequential Reads
- Batch-Operation for Reading Logs

Apache Kafka ~ [Topic, Partition & Offset]

Partitioning Strategy

- **With Key** = Default Partitioner
- **Without Key** = Round-Robin Fashion

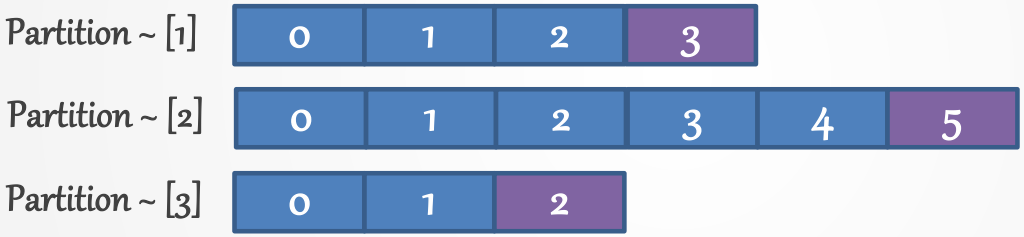


Topic

- Store Streams of Records
- Events are Written into Topics

Partition[s]

- Ordered & Immutable Sequence of Records [Not Global]
- Persisted in an Append-Only Fashion
- Horizontal Scale for Write Throughput



Producer [Writes]

- Producer Writes Data [In]

Consumer [Reads]

- Partition 1 ~ Offset 2
- Partition 2 ~ Offset 3
- Partition 3 ~ Offset 0

Consumer [Reads]

- Partition 1 ~ Offset 0
- Partition 2 ~ Offset 0
- Partition 3 ~ Offset 0

Offset

- Integer Number
- Current Position of a Consumer

Apache Kafka ~ [Log Compaction]

Type = Event Data vs. Keyed

Event Data

- Unrelated Occurrences
- Retains a Window of Data = [Days or GBs]
- Data Retention of 1 Week [7 Days]
- Infinite Retention

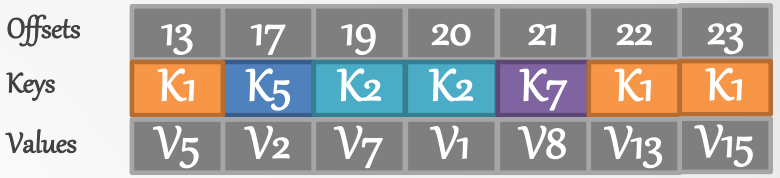
Keyed

- Record a State Change in an Entity [Key]
- Changelog of a Database
- Capability to Replay a State of a Source System

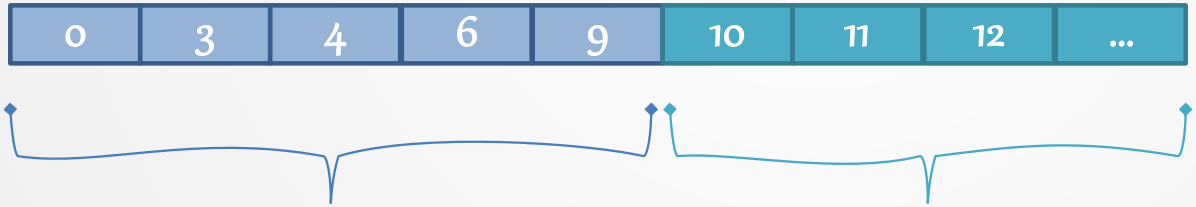
Log Compaction

- Different Type of Retention Strategy
- Keep Last Record by Storing Current State
- Retention Policy = Compact
- Requisite = KEY

Before Compaction
all keys & values



After Compaction
last value of a key



Clean

This section is now clean. Note how some offsets are missing. Those are messages that were removed by compaction.

Dirty

This section is the dirty head of the log. It will get compacted later

Apache Kafka ~ Topic, Partition, Offset, Segment & Log Compaction

1. Topic, Partition & Offset Info
2. Storage Internals = Segments
3. Log Compaction

Apache Kafka ~ [Broker]

Topics

- Stores Stream of Records
- Users & Transactions
- Partition = 3

users

transactions

Brokers

Receives & Stores Messages in a Topic
Manage Multiple Partitions
Messages Identified by Offset

Producers

- Send Messages to a Topic

Producer
users

Producer
transactions

Consumers

- Read Messages from a Topic

Consumer
users

Consumer
transactions

Broker 01
Partition 1
Partition 1

Broker 02
Partition 2
Partition 2

Broker 03
Partition 3
Partition 3

Advantages of Pull Architecture

- Producers & Consumers are Decoupled
- Producers Don't Affect Consumers
- Consumers Don't Affect Producers
- [New] Consumers without Affecting Producers
- Failure of Consumers Does Not Impact System

Apache Kafka ~ [Broker] = Partition Leadership & Replication

1 Topic

- Topic = Users
- Partition = [4]
- Replication = [3]

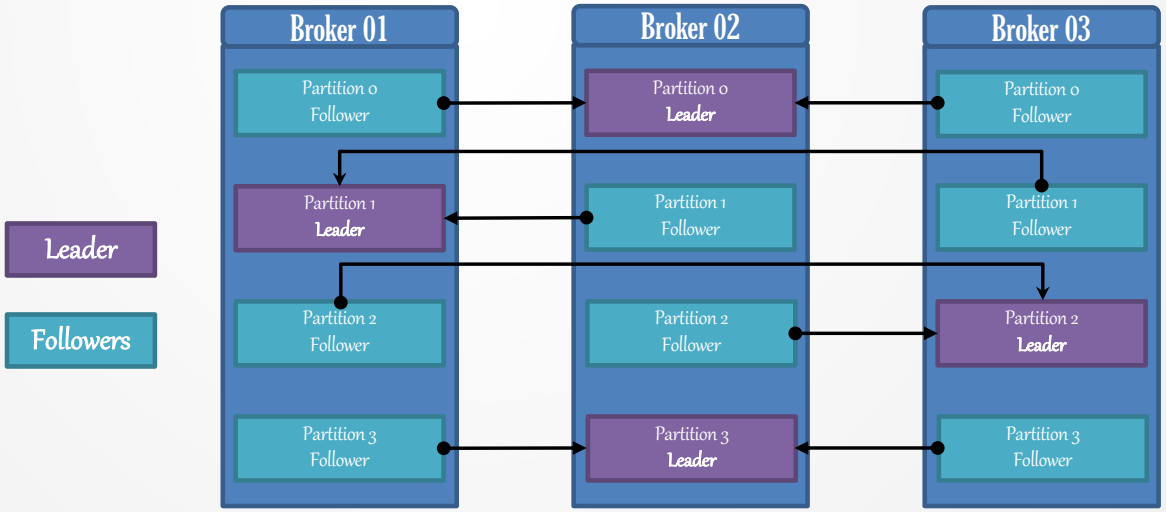
2 Partitioning

- Ordered & Immutable Sequence of Records [Not Global]
- Persisted in an Append-Only Fashion
- Horizontal Scale for Write Throughput

3 Replication

- Guarantees Availability & Durability
- Leader = Single Replica Partition [Producer Send Messages]
- Follower = Don't Serve Client Requests [> 10 Secs Out-of-Sync]
- Number of Copies of a Topic [Defined at Topic Level]

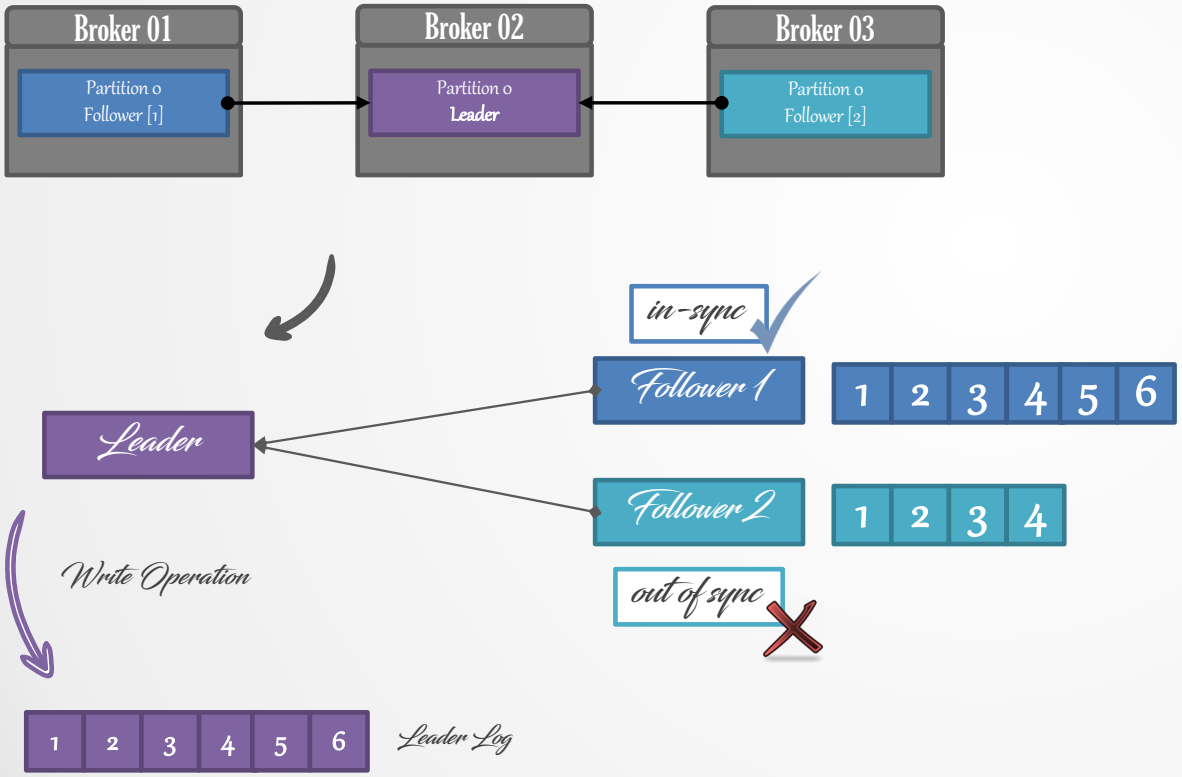
```
1,
{
  "id":1,
  "first_name":{"string":"luan"},
  "last_name":{"string":"moreno"},
}
2,
{
  "id":2,
  "first_name":{"string":"mateus"},
  "last_name":{"string":"oliveira"},
}
3,
{
  "id":3,
  "first_name":{"string":"leonardo"},
  "last_name":{"string":"coco"},
}
4,
{
  "id":4,
  "first_name":{"string":"priscila"},
  "last_name":{"string":"raisa"},
}
```



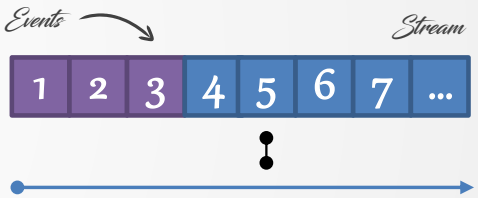
Apache Kafka ~ [Broker] = Log Replication & Retention

Broker	Value	Default
Replica.Lag.Time.Max.Ms	30000	30000
Min.InSync.Replicas	1	1

log replication process

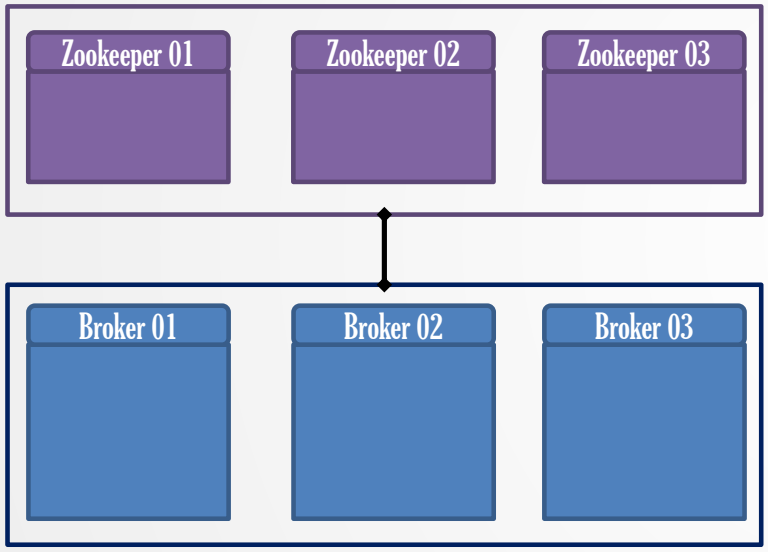


log retention



Broker	Value	Default
Log.Retention.Hours	168	168
Log.Retention.Minutes	NULL	NULL
Log.Retention.Ms	NULL	NULL
Log.Retention.Bytes	-1	-1

Apache Kafka ~ [Broker] = Zookeeper & Controller



Apache Zookeeper

- Centralized Service for Maintaining Config Information
- Distributed Synchronization
- Used By Distributed Applications

Apache Zookeeper & Apache Kafka

- Cluster Management
- Failure Detection & Recovery
- Storage of Security Settings = ACLs & Passwords

Controller

- Electing Partition Leaders
- [First] Broker Elected as Controller
- Creates an Ephemeral Node in Apache Zookeeper

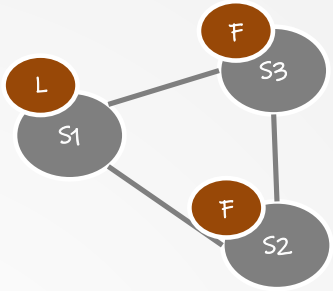
Apache Kafka ~ Partition Leadership, Log Replication, Zookeeper & Controller

1. Partition Leadership Info
2. Log Replication [ISR]
3. Retention

Zookeeper-Less Apache Kafka Option

- KIP-500 ~ Replace Zookeeper with a Self-Managed Metadata Quorum
- Apache Kafka Raft Metadata Mode ~ **KRaft**
- Consensus Protocol for Communication
- Removal of Apache Zookeeper Dependency
- New Quorum Controller Service

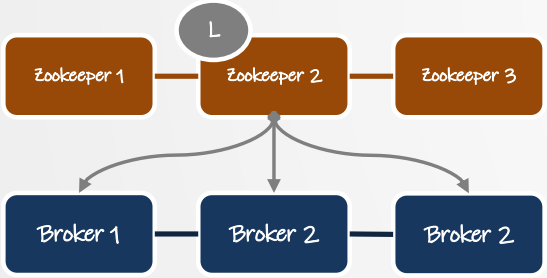
Raft Consensus Algorithm



Explanation

- Consensus is a Fundamentals Problem in Fault-Tolerant Systems
- Consensus Problems ~ Election, Replication & Safety
- Multiple Servers for Agreeing in a Common State even in Failures
- Supported By a Replicated Log & Majority Wins

With Zookeeper



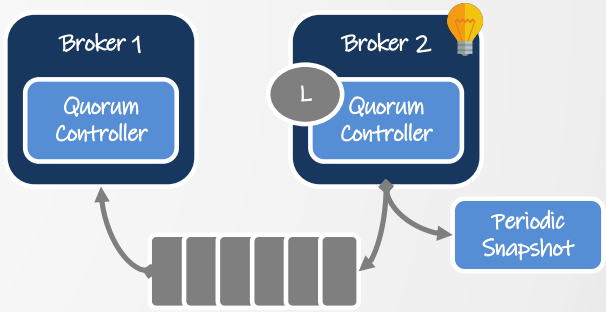
Drawbacks

- Used as Metadata Store with Different Storage Models
- Controller Election Increases Partition Unavailability
- Limits Overall Scalability of Apache Kafka Brokers
- Loading Metadata is an Inefficient and Cumbersome Activity

With Quorum Controller

Benefits

- Scale at Million of Partitions using Improved Control Plane Performance
- Improves Stability & Simplifies Deployment
- Single Security Model
- Lightweight and Single Process
- Controller Failover Near-Instantaneous



Metadata Topic (Event Store)

- Continuous Sync In-Memory using Commit Log
- Replicate Metadata Accurately Across Quorum
- Event-Sourced Storage Model ~ Log Communication
- Periodic Snapshots for Log Growth Control


It's better to be a lion for a day than a sheep all your life.
Elizabeth Kenny
quotefancy


ONEWAY
SOLUTION