
ONEWAY
SOLUTION

One Way Solution

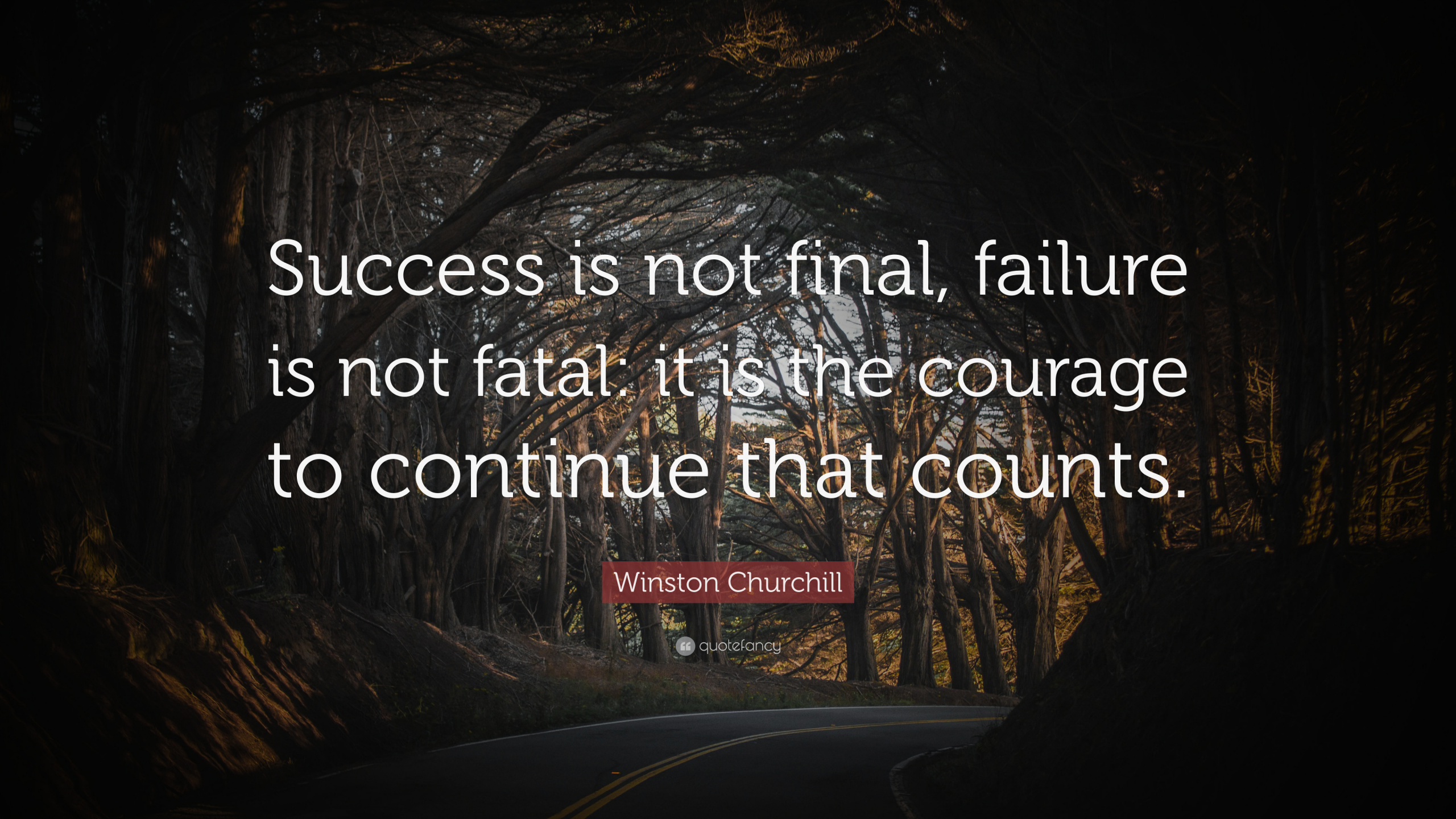
Near Real-Time ETL

Data Engineering – [Day 3]

LUAN MORENO

CEO & CDO

Data Engineer & Data Platform MVP

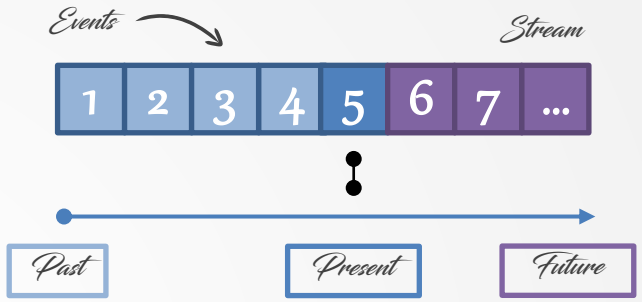

Success is not final, failure is not fatal: it is the courage to continue that counts.
Winston Churchill
quotefancy

Event Stream [ES]

Event Stream [ES] = Representation of an Unbounded DataSet
Unbounded = Infinite & Ever Growing

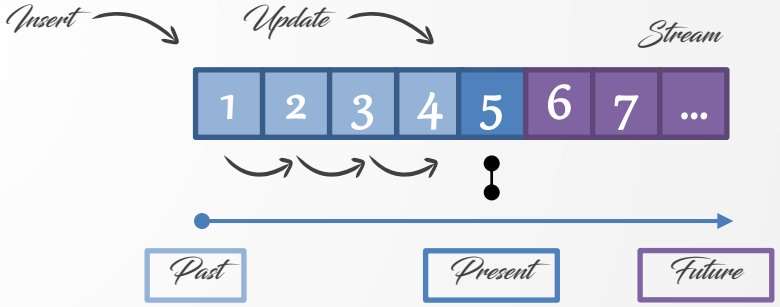
Event Streams [Ordered]

There is an inherent notion of which events occur before or after other events. This is clearest when looking at financial events. A sequence in which I first put money in my account and later spend the money is very different from a sequence at which I first spend the money and later cover my debt by depositing money back. The latter will incur overdraft charges while the former will not. Note that this is one of the differences between an event stream and a database table records in a table are always considered unordered and the order by clause of SQL is not part of the relational model.



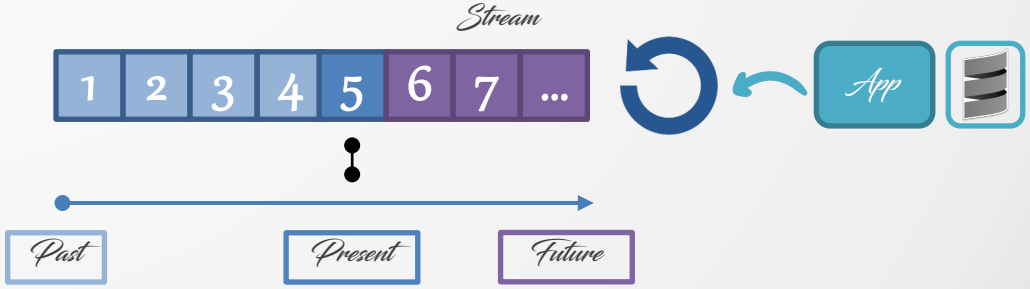
Immutable Data Records

Events, once occurred, can never be modified. A financial transaction that is cancelled does not disappear. Instead, an additional event is written to the stream, recording a cancellation of previous transaction. When a customer returns merchandise to a shop, we don't delete the fact that the merchandise was sold to him earlier, rather we record the return as an additional event. This is another difference between a data stream and a database table. We can delete or update records in a table, but those are all additional transactions that occur in the database, and as such can be recorded in a stream of events that records all transactions. If you are familiar with binlogs, WALs, or redo logs in databases you can see that if we insert a record into a table and later delete it, the table will no longer contain the record, but the redo log will contain two transactions - the insert and the delete.

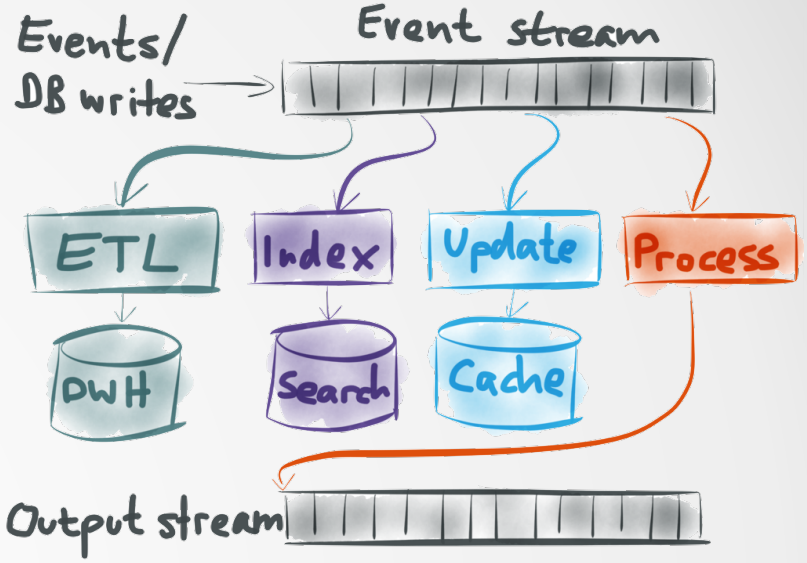


Event Streams [Replayable]

This is a desirable property. While it is easy to imagine nonreplayable streams (TCP packets streaming through a socket are generally nonreplayable), for most business applications, it is critical to be able to replay a raw stream of events that occurred months (and sometimes years) earlier. This is required in order to correct errors, try new methods of analysis, or perform audits. This is the reason we believe Kafka made stream processing so successful in modern businesses—it allows capturing and replaying a stream of events. Without this capability, stream processing would not be more than a lab toy for data scientists.

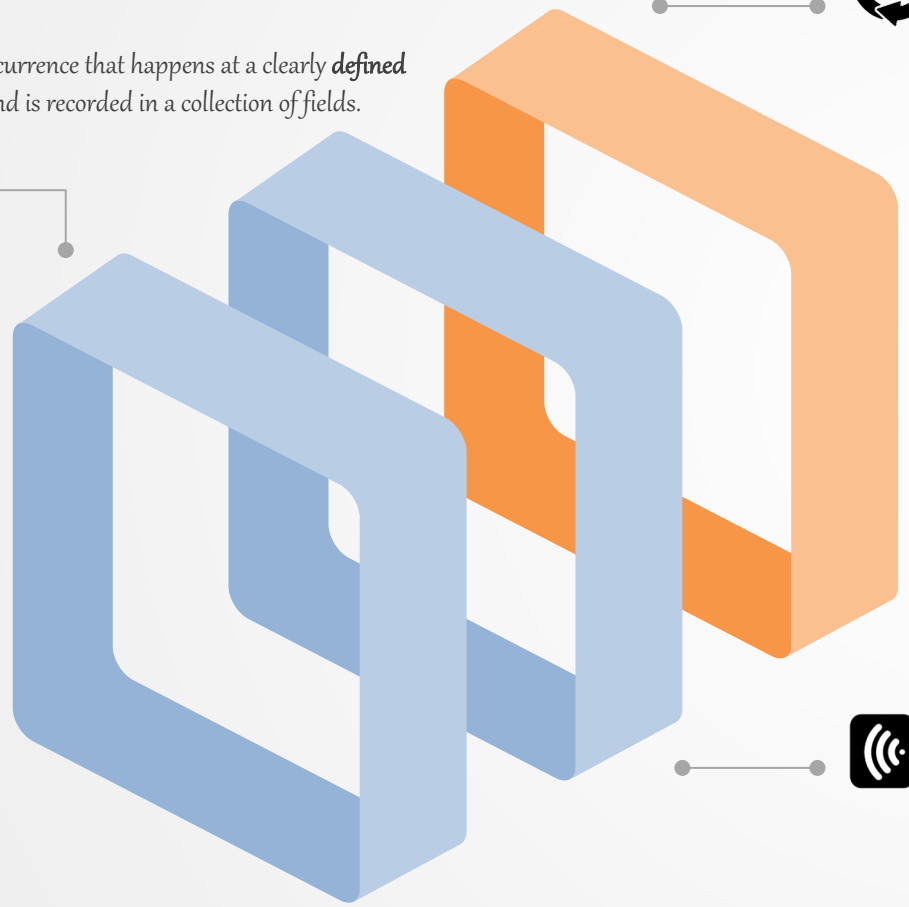


Event Stream Processing [ESP]



Event

any occurrence that happens at a clearly **defined time** and is recorded in a collection of fields.



Processing

the act of **analyzing data** and perform data analytics on top of that.

Stream

constant **flow of data** events, or a steady rush of data that flows into and around your business.

ESP for Processing Changes

- Store Data Reliability
- Process Incoming Data
- Perform Queries & Analytics
- Push Results Immediately to Subscribers

Real-Time Stream Processing [Engines]

Open-Source Platform [OSS]

- Apache Kafka
- Apache Spark
- Apache Apex
- Apache Flink
- Apache Storm
- Apache Beam

Microsoft Azure

- HDInsight
- Azure Synapse Analytics
- Azure Stream Analytics
- Azure Functions

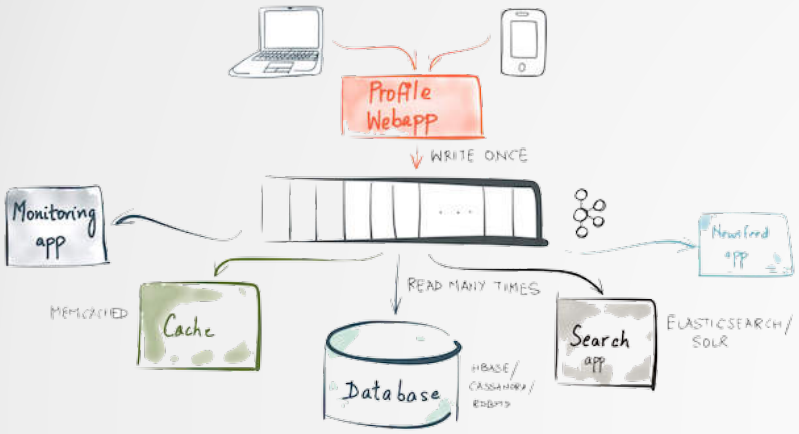
Google Cloud Platform [GCP]

- Cloud DataProc
- Cloud DataFlow
- Cloud Functions

Amazon Web Services [AWS]

- Amazon EMR
- Amazon Kinesis Data Streams
- Amazon Kinesis Data Analytics
- AWS Lambda

Python & Streaming Engines

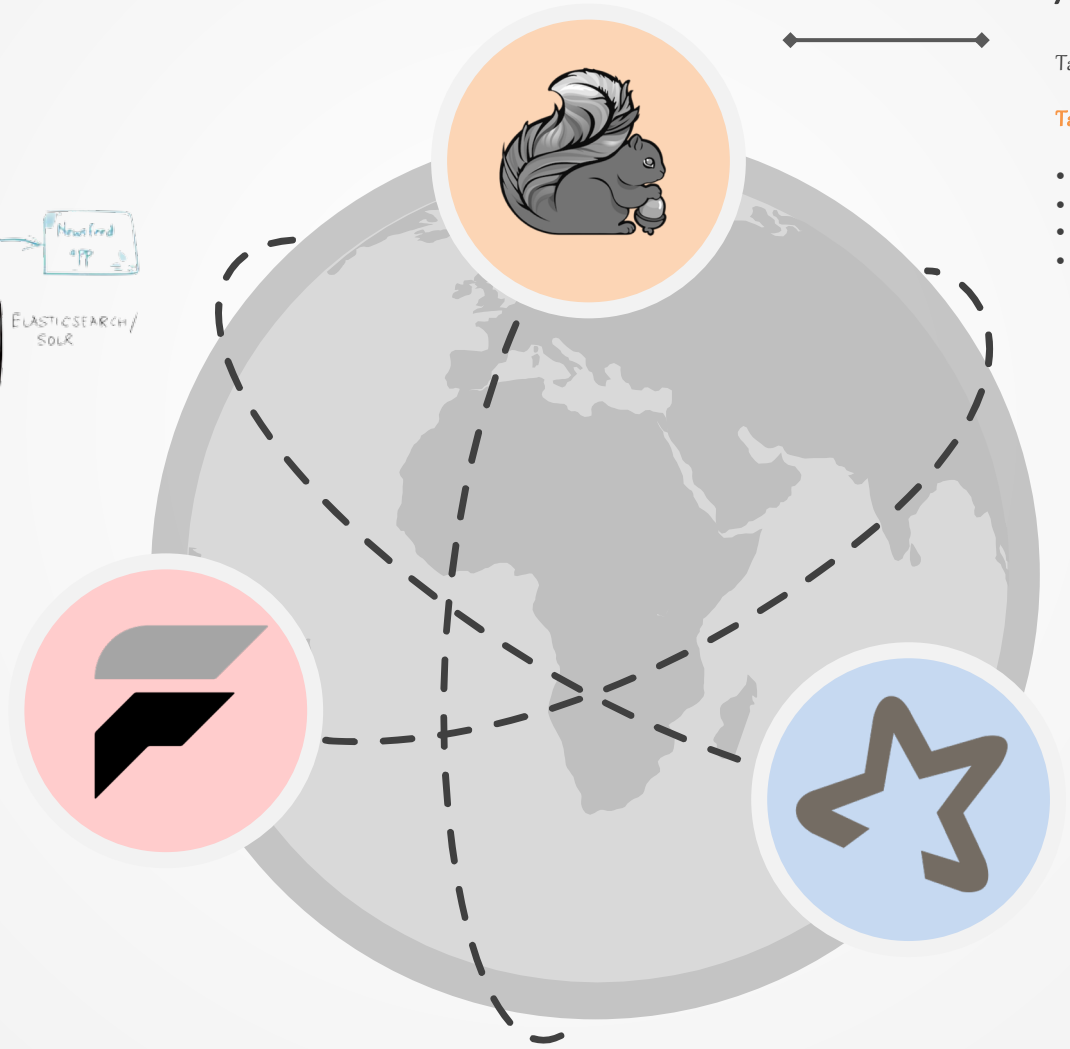


Faust

Python Stream Processing Library

Faust API

- Stream Processing ~ Kafka Streams, Spark, Storm, Samza, Flink
- Numpy, PyTorch, Pandas, NLTK, Django
- Tables ~ Distributed Key & Value
- Use Regular Python Dictionaries



Apache Flink

Table API for a Unified Stream & Batch Processing Experience

Table API

- Language-Integrated Query API for Scala & Java
- Batch & Streaming Input without Code Changes
- Fully Integrated with DataStream and DataSet APIs
- Integrated with Complex Event Processing API

Apache Spark

PySpark ~ Python API for Apache Spark Engine

PySpark API

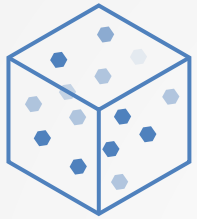
- PySpark ~ Spark DataFrame
- Distributed Table ~ Apache Spark Cluster
- SQL, Streaming, ML ~ Packages
- RDD, DStream & DataFrame

StreamingSQL [Engines]

Apache Flink

Flink SQL

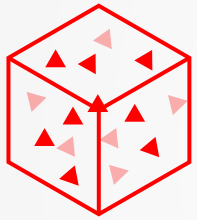
2016



Apache Spark

Structured Streaming

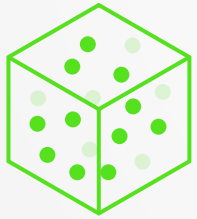
2016



Apache Kafka

KSQL

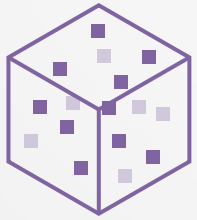
2017



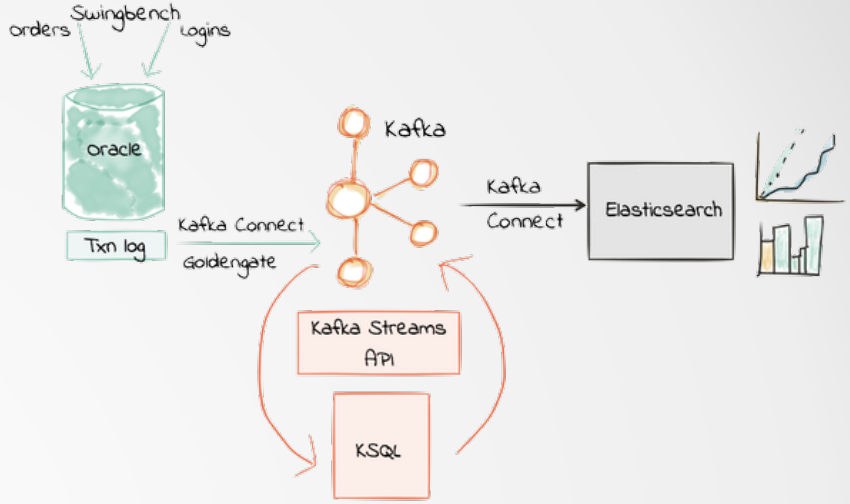
Apache Beam

Beam SQL

2017



Streaming SQL



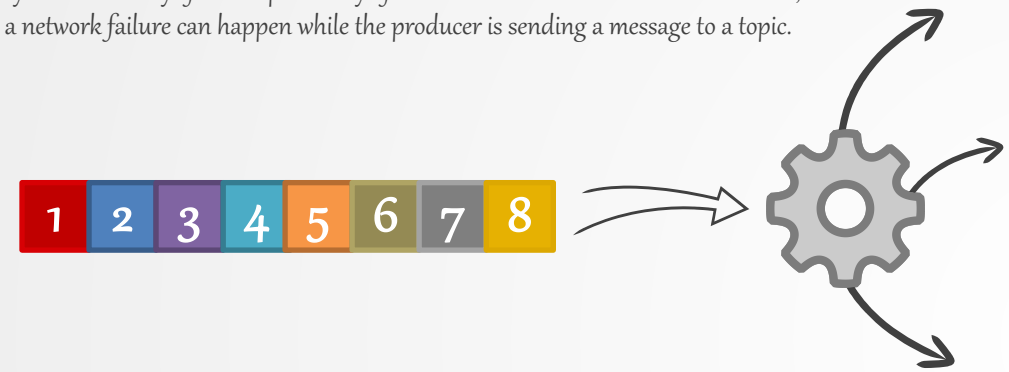
Data Processing

- Manipulate Streams
- Query over Continuous Flow of Data
- Select | Join | Union & Merge | Window & Aggregation
- Real-Time Analytics
- Predictions & ML

Message Delivery Guarantees [Idempotent Producers & EOS]

Messaging System Semantics

in a distributed publish-subscribe messaging system, the computers that make up the system can always fail independently of one another. Individual broker can crash, or a network failure can happen while the producer is sending a message to a topic.



At [Most] Once

if producer does not retry when an ack times out or returns an error, the message might end up not being written to a kafka topic, and hence not delivered to consumer.

ups, lost some...

At [Least] Once

producer receives an (ack) from broker with acks=ALL, meaning it was written once to the kafka. however, if an error arise, it might retry sending the same message, assuming that the message was not written.

hmm, got some duplicates...

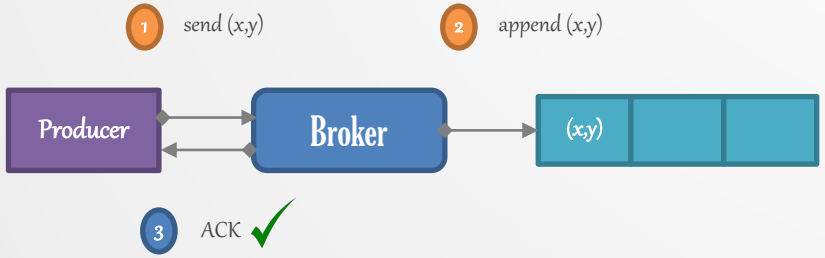
[Exactly] Once Semantics [EOS]

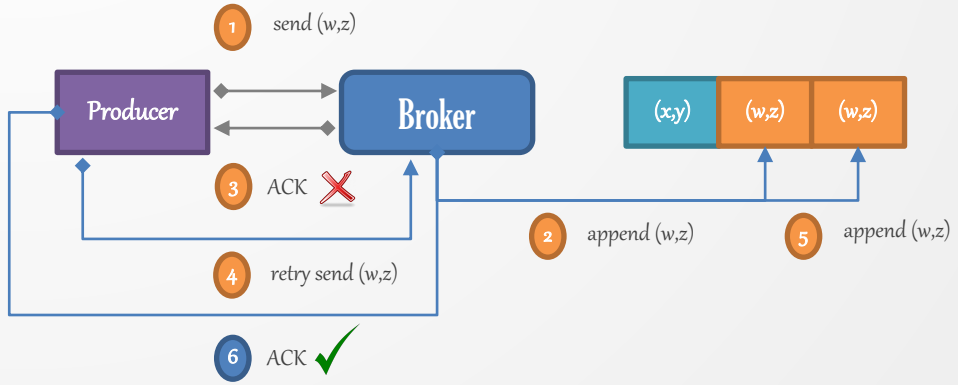
even if a producer retries sending a message, it leads to the message being delivered exactly once to the end consumer. requires cooperation between the systems.

this is exactly what i was waiting for...

Idempotent Producers

can be performed many times without causing a different effect than only being performed once. if retries occurs, same message won't be written again. each message will contain a sequence number which broker will use to dedup any duplicate send.






Freedom lies in being bold.
Robert Frost
quotefancy

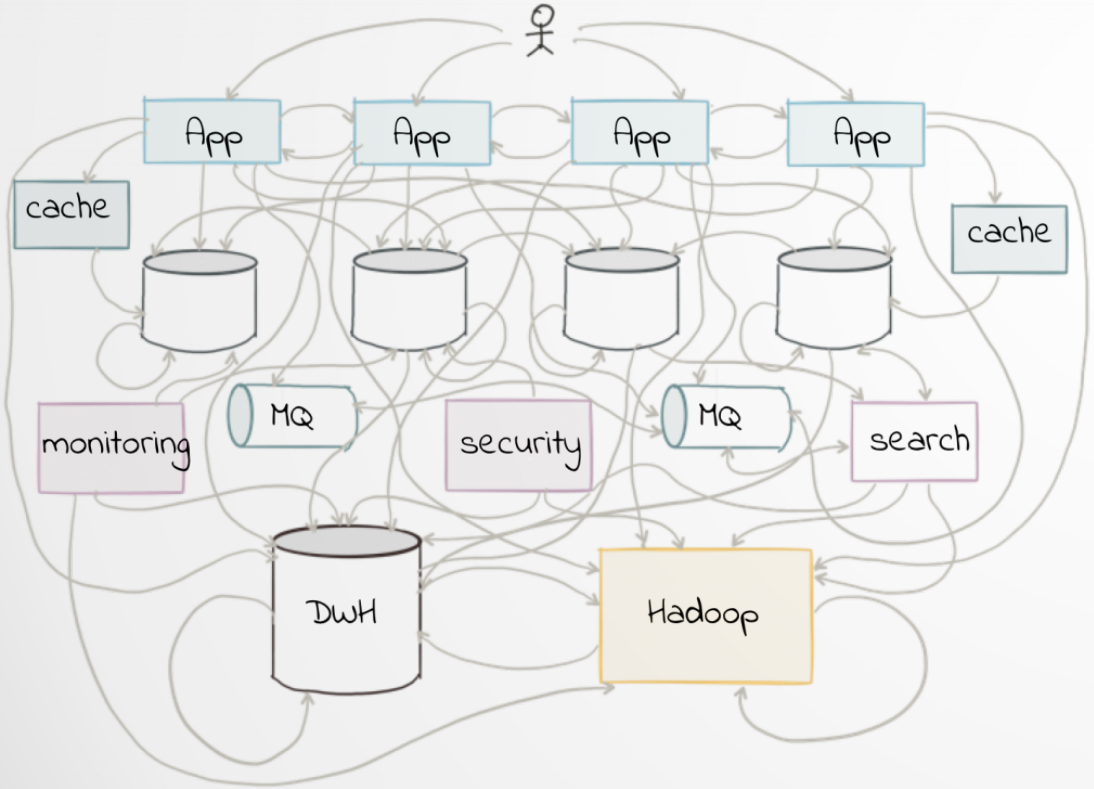
Apache Kafka [De-Facto Streaming Platform]

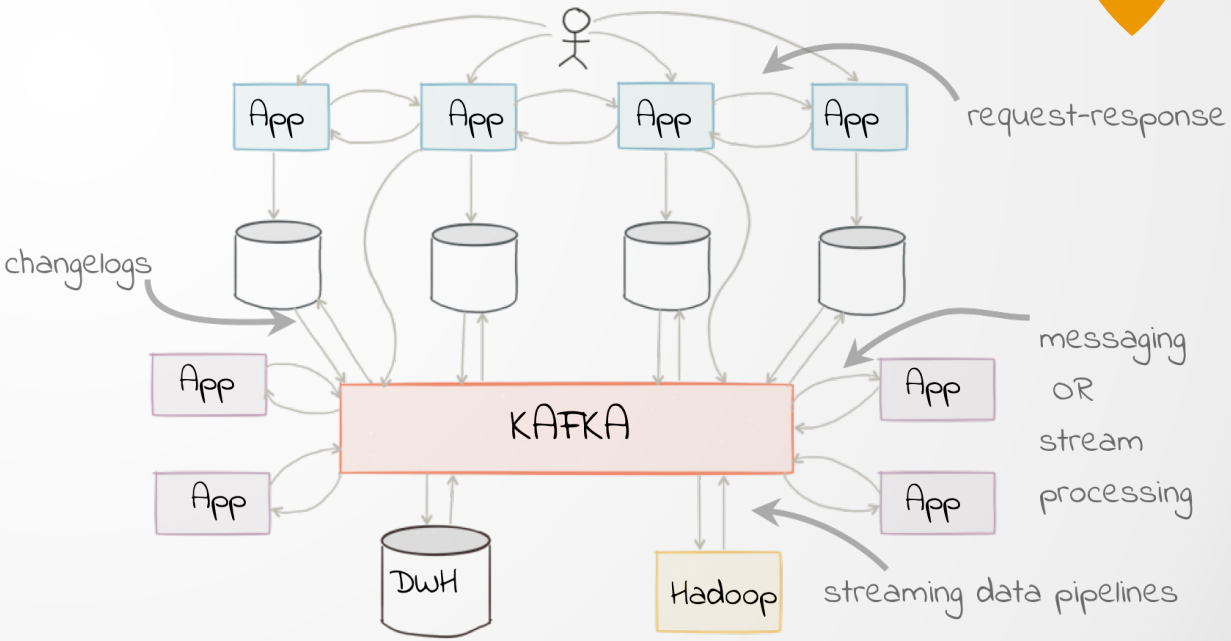
< Apache Kafka

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

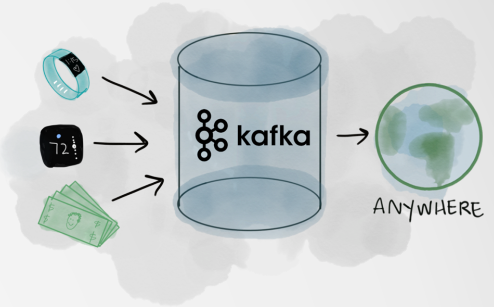
> Apache Kafka

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





Apache Kafka [Fundamentals]



Apache Kafka

- Open-Source Software – [Scala & Java]
- 7,500 Companies using Apache Kafka
- + 180 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

History

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

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]

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]

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

Apache Kafka [The Log Structure]

Logs in a Database

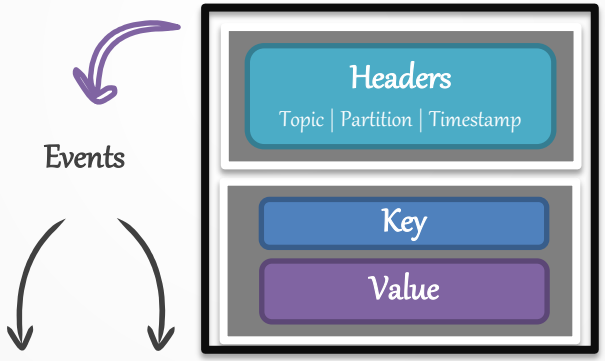
- Data Structured & Indexes
- Immediately Persisted on Disk
- ACID [Atomic, Consistency, Isolation & Durability]
- Master & Slave [Sync of Log File]

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]

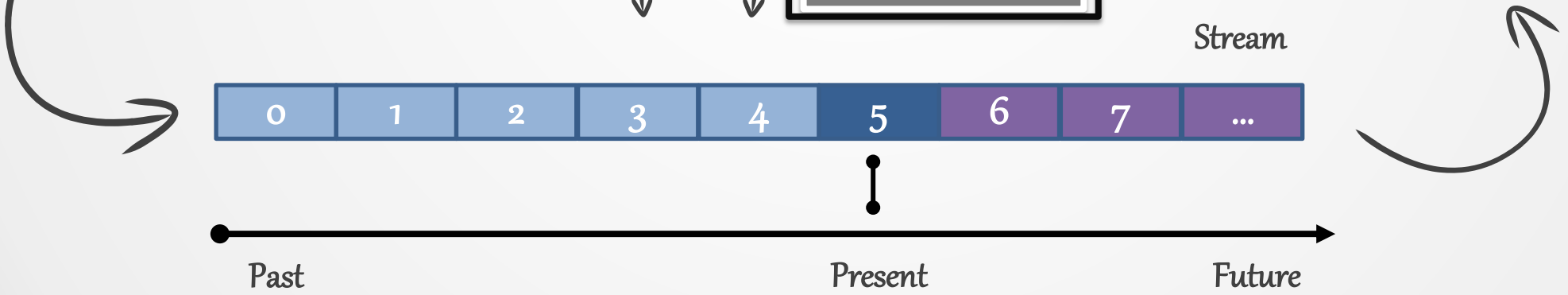
Writing into Log

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



Reading from Log

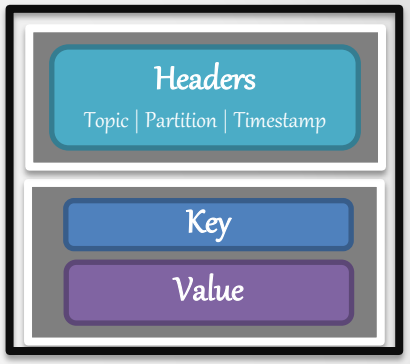
- Passthrough JVM [Java Virtual Machine]
- Copy from Disk Buffer to Network Buffer [Zero Copy]
- Sequential Reads
- Batch-Operation for Reading Logs



Apache Kafka [Topic, Partition & Offset]

Partitioning Strategy

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

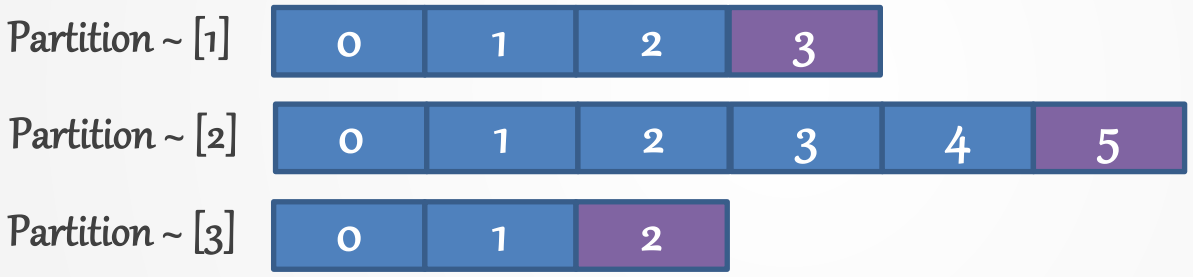


Topic

- 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 [Broker]

Topics

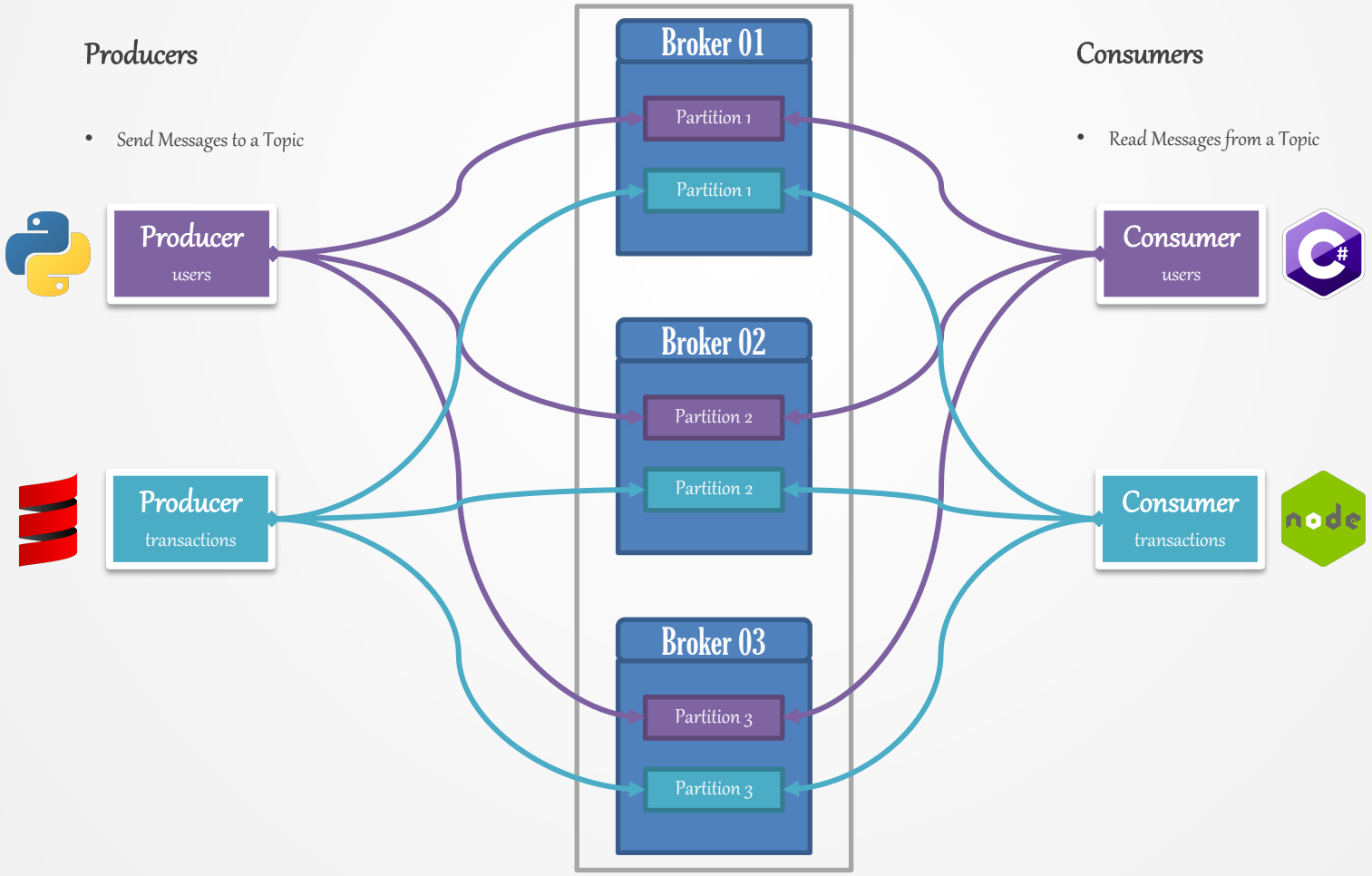
- Stores Stream of Records
- Users & Transactions
- Partition = 3

users

transactions

Brokers

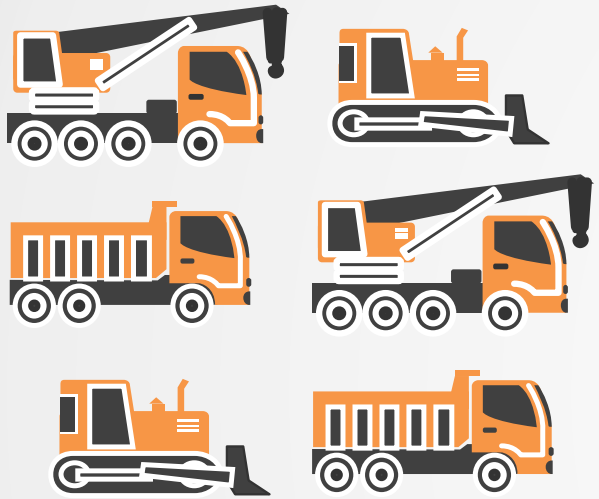
Receives & Stores Messages in a Topic
Manage Multiple Partitions
Messages Identified by Offset



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 [Use-Case] ~ Monitoring and Tracking System



GPS

Global Positioning System
Truck Location Data
Every 20 Seconds

Location Dashboard

Report & Dashboards
Location of Trucks

Notification Service

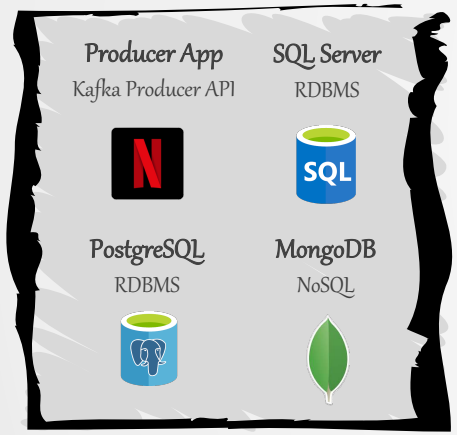
Report & Dashboards
Location of Trucks
Weather Conditions & Fuel Status

Topic

Topic Name – gps_location_trucks
Data – truck_id | latitude | longitude
Partitions – 5 [Ordered]
Key – truck_id

Apache Kafka [Use-Case] ~ Netflix

Producers → Enrichment → Consumers



Info

- 35 + Fronting Apache Kafka Clusters
- 3.000 + Instances Provisioned
- Replication Factor of 2
- Retention Period of 8 ~ 24 Hours

Info

- 12 Consumer Apache Kafka Clusters
- 900 + Instances Provisioned
- Replication Factor of 2
- Retention Period of 2 ~ 4 Hours

Ingestion Lane

- 700 Billion Events per Day
- 13 Million Events & 29 GB per Second
- Hundreds of Event Types

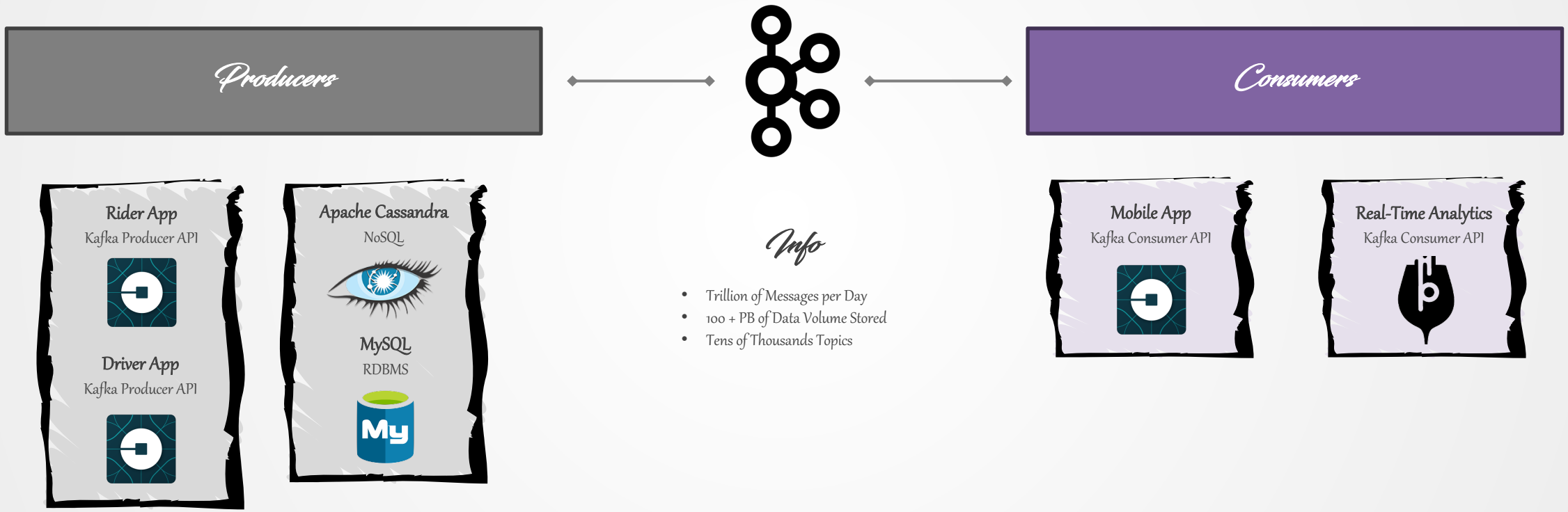
Enrichment Lane

- Business Logic on Stream Processor
- Microservices Pattern Need Full View of a DataSet
- Used for GraphQL & GRPC Systems

Consumption Lane

- Consuming Microservices Reading from Topics
- Spring Kafka Connectors
- Use of Confluent Schema Registry ~ Apache Avro

Apache Kafka [Use-Case] ~ Uber



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 = 2.8**

Apache Kafka

- Open-Source Stream-Processing Software
- Publish & Subscribe
- Process & Store
- **Current Version = 2.8**

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 = 6.2.0**
- **Current Kafka Version = 2.8**

Amazon MSK

- Managed Streaming for Apache Kafka
- Fully Compatible & Managed
- Elastic Stream Processing [Apache Flink]
- High Available & Secure
- Process & Store
- **Current Version = 2.8**

Azure HDInsight

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

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**

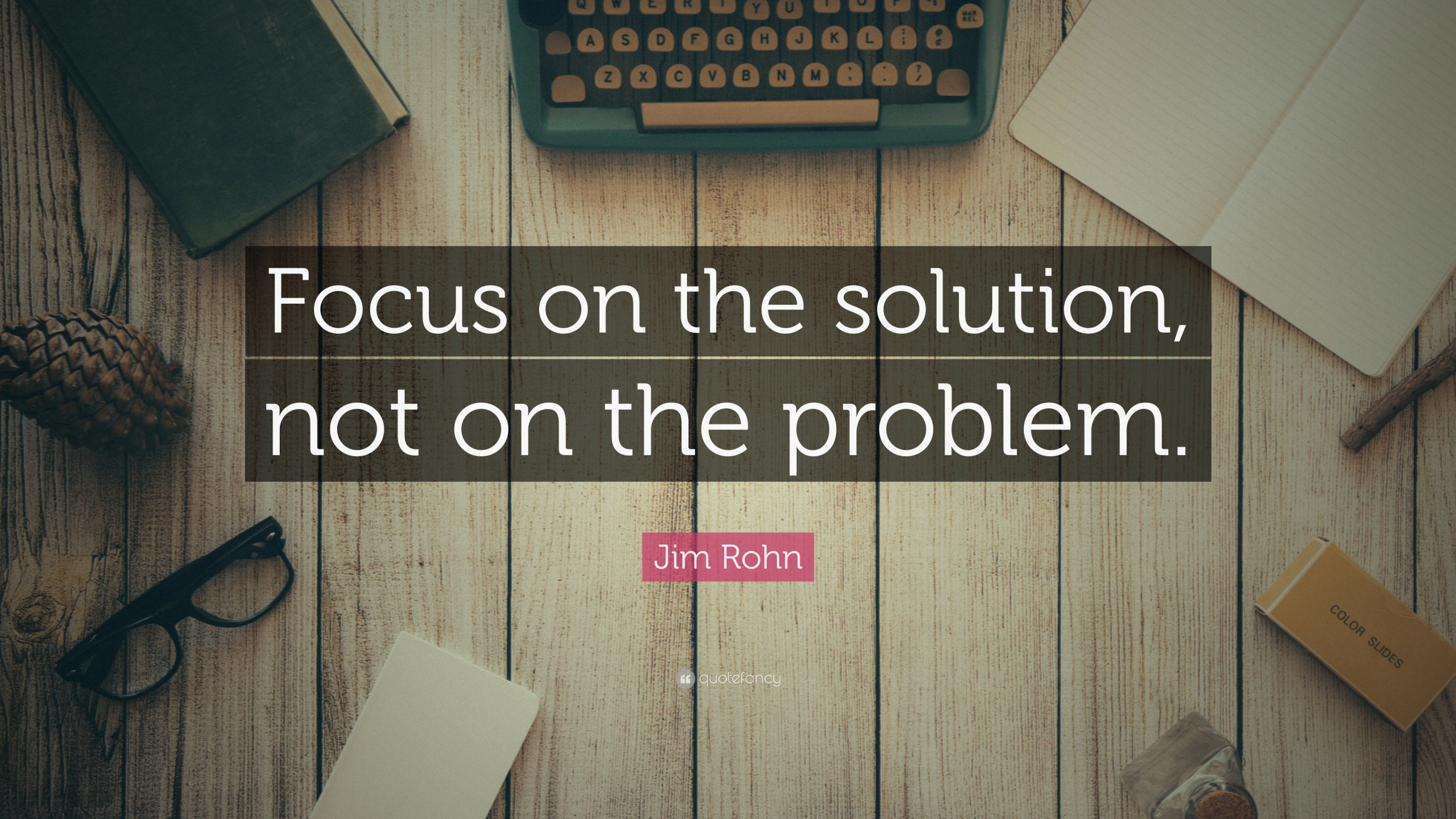
Total Annual Cost for Apache Kafka

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

Apache Kafka on Kubernetes using [Strimzi]

Producing & Connecting to Data Sources

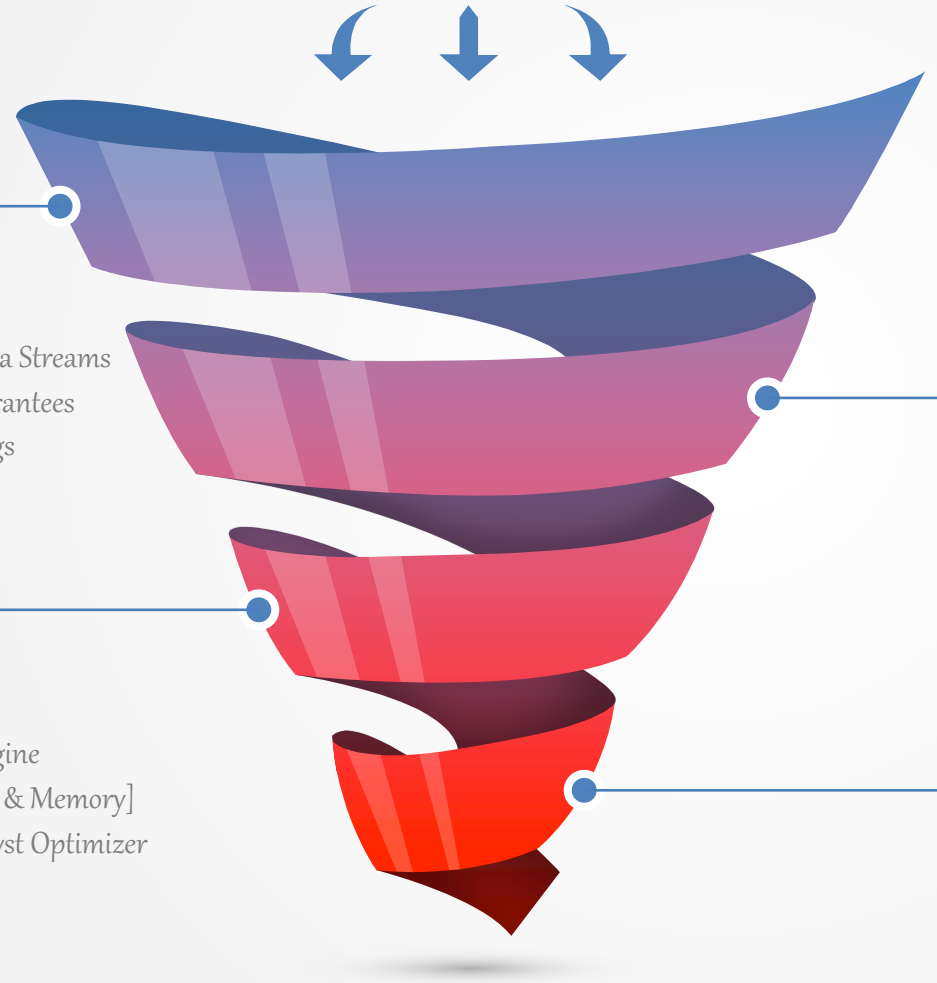
http://bit.ly/owshq-kafka


Focus on the solution,
not on the problem.
Jim Rohn
quotefancy

Structured Streaming

Philosophy

Treat Data Streams = **Unbounded Tables**
Incremental Query over Streams



Guarantees

Scalable & Fault-Tolerant
Incremental & Continuous Data Streams
End-to-End Exactly-Once Guarantees
Checkpoint & Write Ahead Logs

Seamless

Express Streaming Computation = Batch Computation

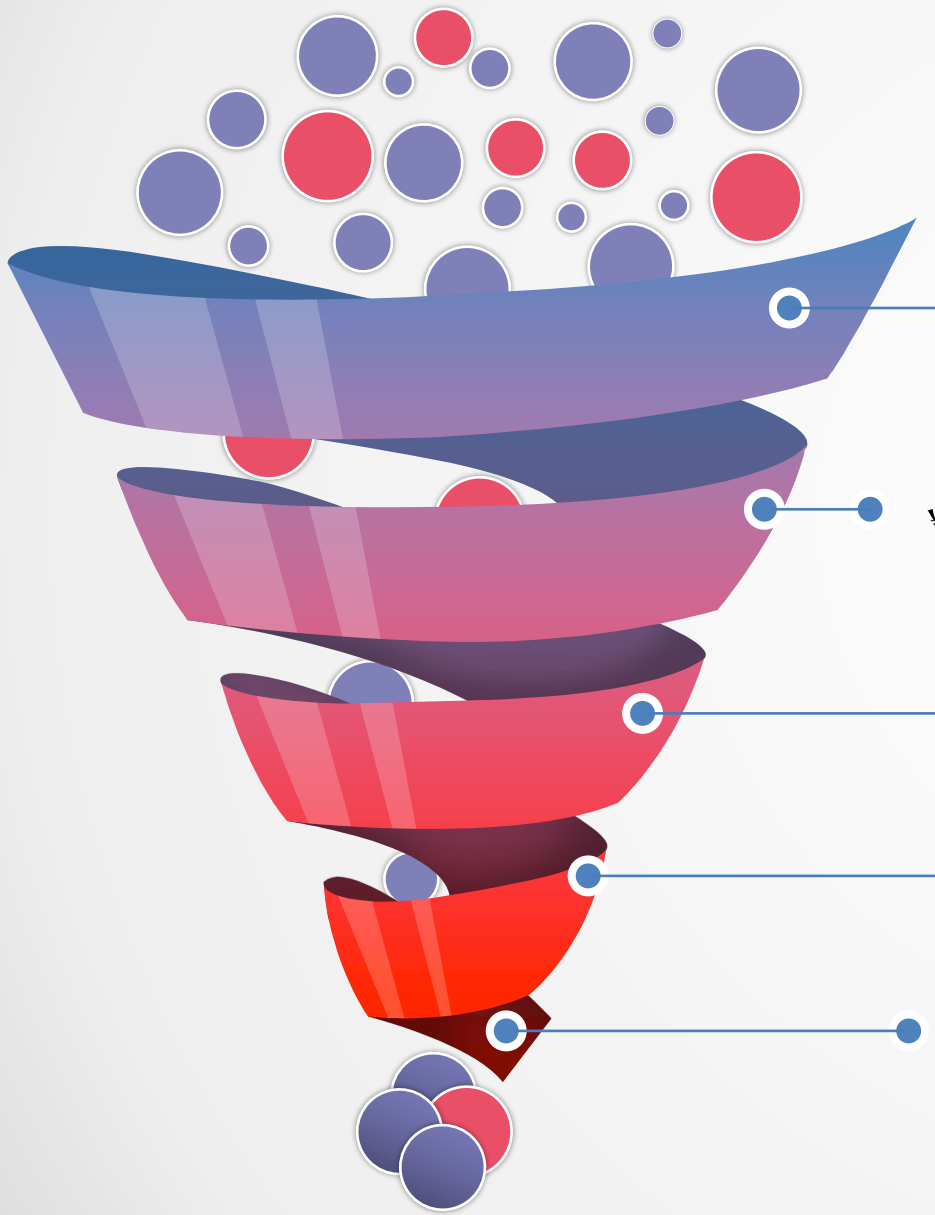
Engine

Built on Spark SQL Engine
Project Tungsten [CPU & Memory]
Optimized using Catalyst Optimizer

Processing

Uses Micro-Batch Processing Engine
100 milliseconds with Exactly-Once Latency

Anatomy of a Streaming Query



Source

```
spark.readStream
.format("kafka")
.option("subscribe", "input")
.load()
```

Transformation

```
.groupBy('value.cast("string") as 'key)
.agg(count("*") as 'value)
```

Sink

```
.writeStream
.format("kafka")
.option("topic", "output")
```

Trigger

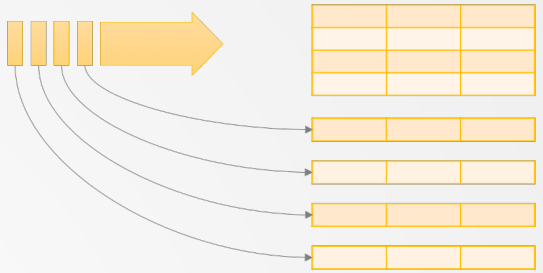
```
.trigger("1 minute")
.outputMode("update")
```

Checkpoint

```
.option("checkpointLocation", "…")
.start()
```

Data Streams

Unbounded Table



data stream as an unbounded table

new data in the data stream = new rows appended to an unbounded table

Structured Streaming [Use-Case] ~ Near Real-Time ETL

Source
Data Lake

Source
Apache Kafka

Structured Streaming

Bronze
Raw Data [Unstructured]

Stage 1
convert unstructured data to
structured tabular data
Latency of ***few minutes***

Structured Streaming

Silver
Clean & Transform Data

Stage 2
process, transform and save data in a storage
for **data skipping & optimization** purpose

Structured Streaming

Gold
Cleaned & Curated Data

Stage 3
query latest structured data
interactively or with periodic jobs

Sink
Apache Kafka

Sink
Data Warehouse

Building a Data Lakehouse using Near Real-Time ETL [Data Lake & Apache Kafka]


A goal without a plan is just a wish.
Antoine de Saint-Exupéry
quotefancy

Use-Case: Microsoft Azure

Azure Event Hubs

Simple, Secure, & Scalable Real-Time Data Ingestion

- AMQP
- HTTPS
- Apache Kafka

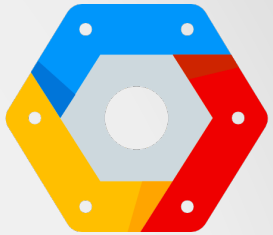
Azure Stream Analytics

Serverless Real-Time Analytics [SQL Interface]
Machine Learning

Azure Synapse Analytics

Fast, Flexible, & Secure Cloud Data Warehouse for Enterprises
SQL & PolyBase Features with Fast Loading Operations

Use-Case: Google Cloud Platform [GCP]



Google Pub/Sub

Global Messaging & Event Ingestion
Scale without Provisioning, Partitioning, or Load Isolation
Expand Pipelines to New Regions Simply with Global Topics

Cloud DataFlow

Simplified Stream & Batch Data Processing
Apache Beam [Java | Python | SQL]

Google BigQuery

ServerLess [SaaS], Highly-Scalable, & Cost-Effective Cloud Dw
In-Memory BI Engine & ML
Gartner 2019 – Magic Quadrant for Data Management Solutions

Use-Case: Amazon AWS

Amazon Kinesis

Easily Collect, Process & Analyze Streams in Real-Time

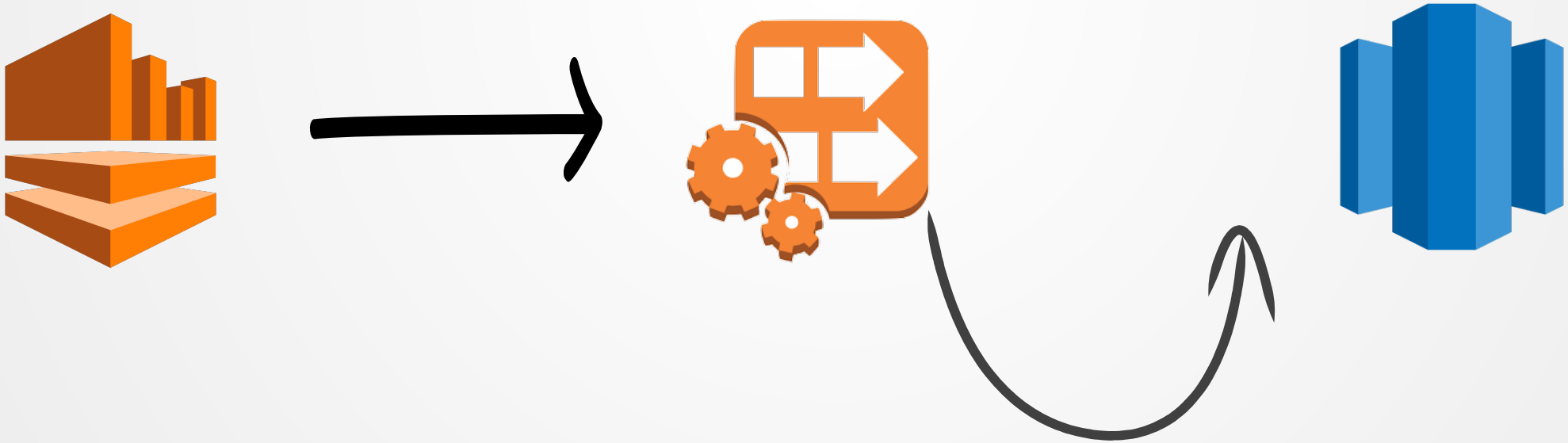
- Kinesis Data Streams
- Kinesis Data Firehose

Amazon Kinesis Data Analytics

Analyze Streaming Data
Query Streams of Data

Amazon Redshift

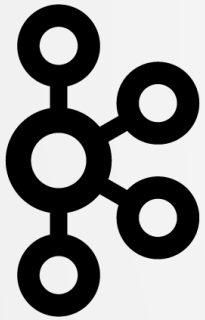
Fast, Simple, Cost-Effective Modern Data Warehouse
MPP | ML | Result Caching & S3 Query Access



Use-Case: Open-Source Software [OSS]

Apache Kafka

Real-Time Data Pipelines & Streaming Apps
Horizontally Scalable, Fault-Tolerant & Wicked Fast

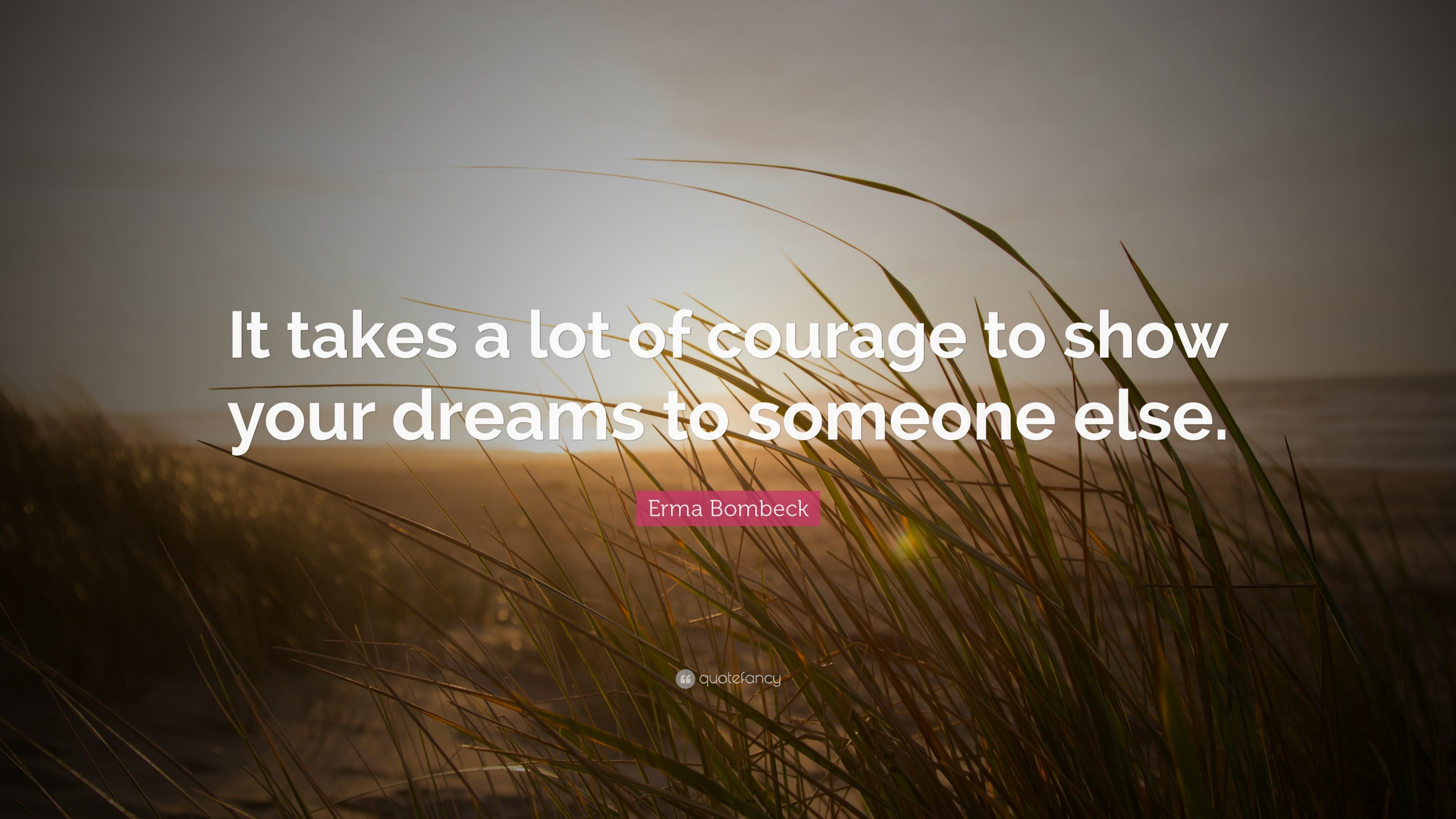


Apache Spark

Unified Analytics Engine for Large-Scale Data Processing
Speed, Easy to Use, Generality & Runs Everywhere

Apache Pinot

Real-Time Distributed OLAP DataStore, Designed to Answer
OLAP Queries with Low Latency


It takes a lot of courage to show your dreams to someone else.
Erma Bombeck
quotefancy

ONEWAY

SOLUTION