
ONEWAY
SOLUTION

One Way Solution

The Sinks & Serving

Data Engineering – [Day 4]

LUAN MORENO

CEO & Data Architect

Data Engineer & MVP

MATEUS OLIVEIRA

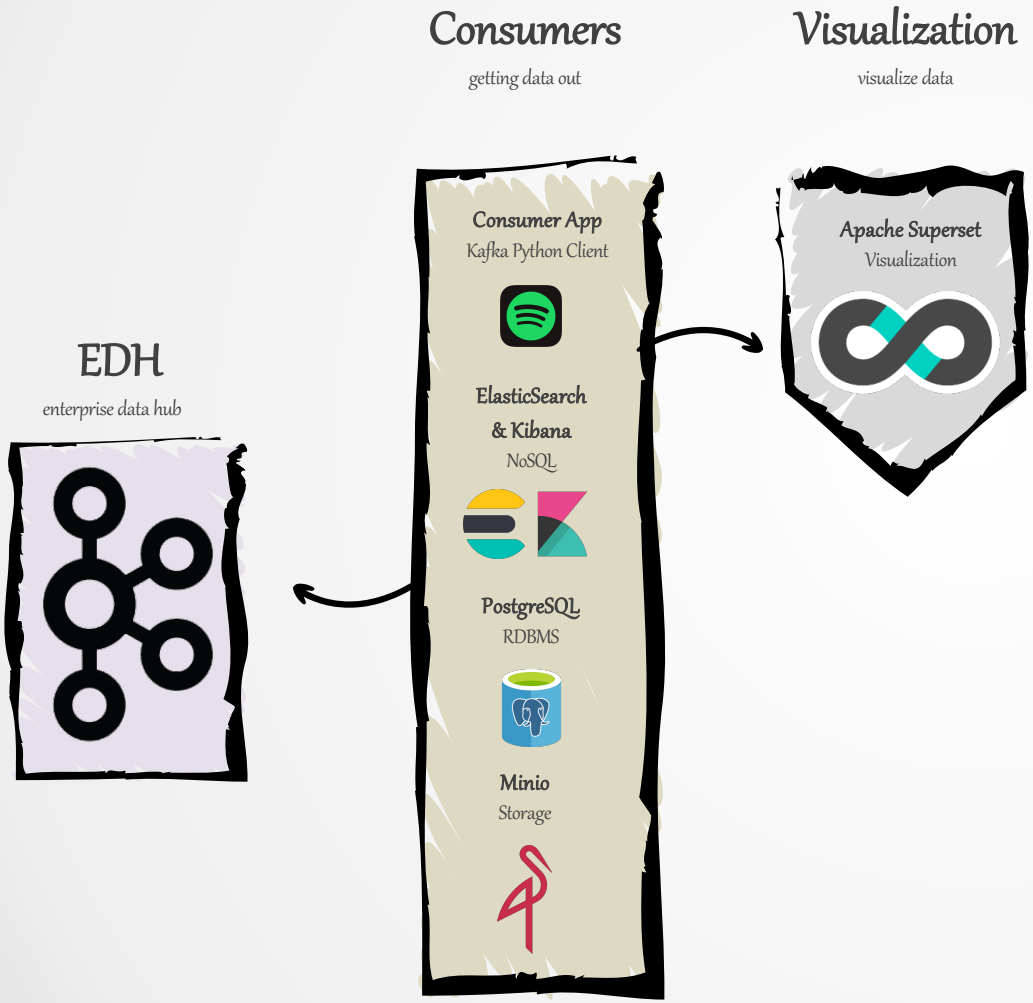
Big Data Architect

Data In-Motion Specialist

Shallow men believe in luck or in circumstance. Strong men believe in cause and effect.

Ralph Waldo Emerson

Use Case ~ [Data Sinks & Serving]



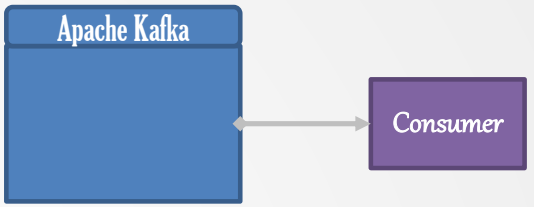

With enough courage, you can do without a reputation.
Margaret Mitchell
quotefancy

Kafka Consumer [Basics]

Consumer API [101]

subscribe to one or more topics and process the stream of records produced to them in an application

- **Consumers** = Subscribers or Readers in a Traditional Messaging Systems
- Reads Messages [Events] from a Topic(s)



Programming Languages [Kafka Clients]

Librdkafka is a C library of the Apache Kafka Protocol, providing Producer, Consumer & Admin Clients. Designed with Message Delivery Reliability and High Performance in mind.

Producers = 1 Million of Messages per Second
Consumers = 3 Million of Messages per Second

Confluent

- C, C++, Java, .Net, Python & Scala

Open Source Community

- Erlang, Groovy, Haskell, Kotlin, Lua, Node.JS, OCaml, PHP, Ruby, Rust, TCL, Swift

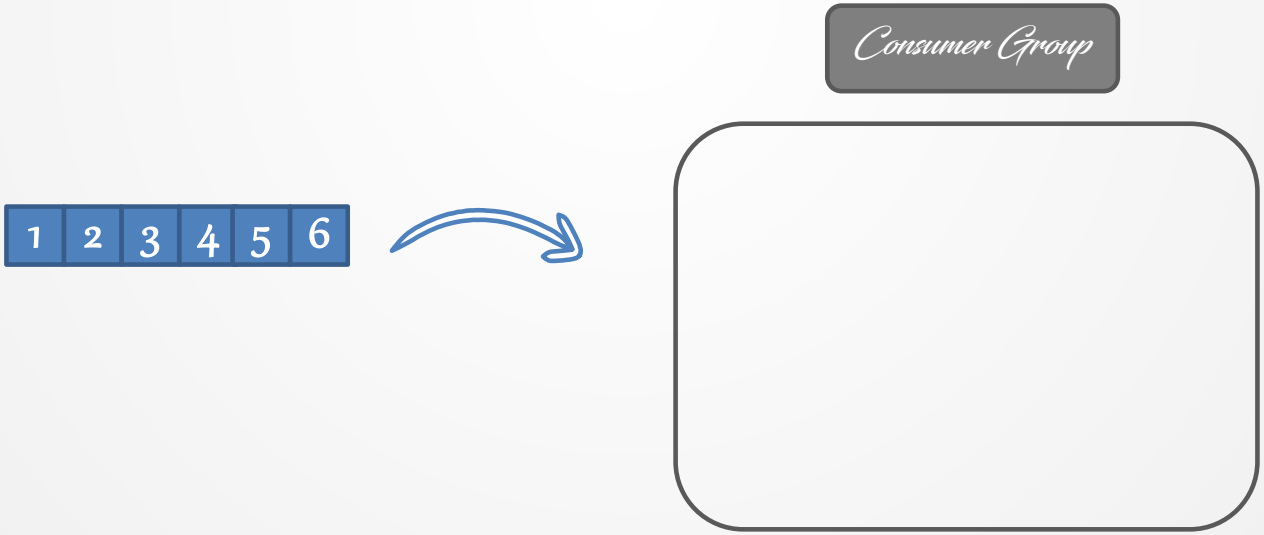
Kafka Consumer & Consumer Groups

Consumer Groups

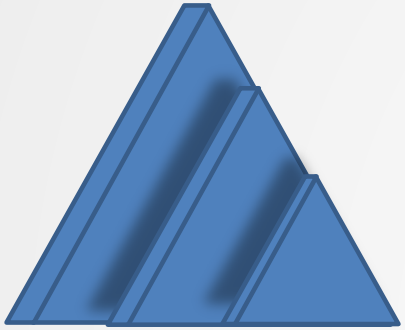
suppose you have an application that needs to read messages from a kafka topic, run some validations against them, and write the results to another data store. in this case your application will create a consumer object, subscribe to the appropriate topic, and start receiving messages, validating them and writing the results. this may work well for a while, but what if the rate at which producers write messages to the topic exceeds the rate at which your application can validate them? if you are limited to a single consumer reading and processing the data, your application may fall farther and farther behind, unable to keep up with the rate of incoming messages. obviously, there is a need to scale consumption from topics. Just like multiple producers can write to the same topic, we need to allow multiple consumers to read from the same topic, splitting the data between them.

Info

Consumer = Reader
Consumer Group = Group of Readers



Kafka Consumer Groups [Case 1]

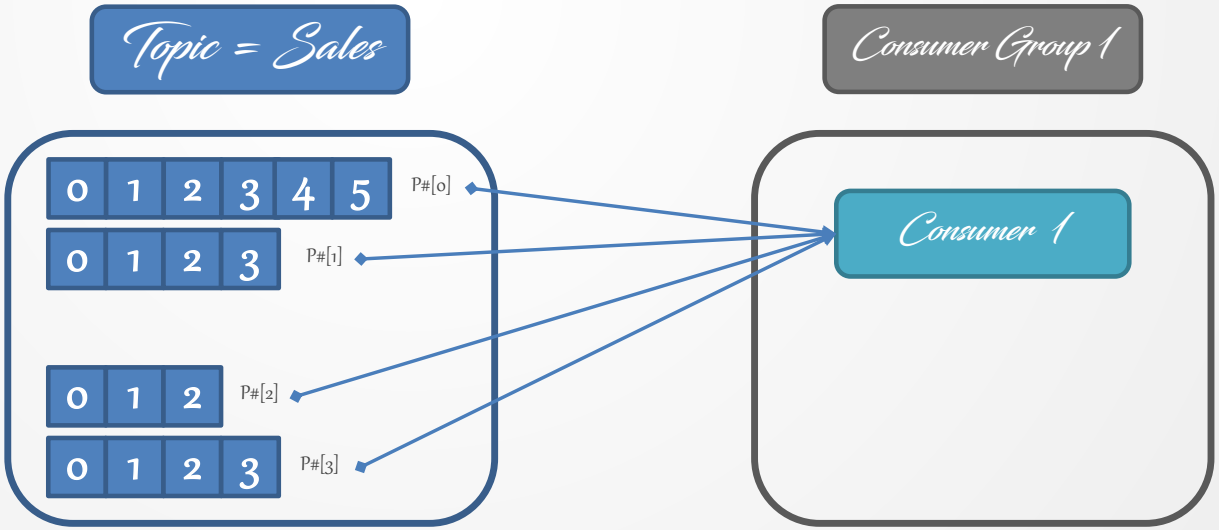


One Consumer Group with 4 Partitions

let's take topic t1 with four partitions. now suppose we created a new consumer, C1, which is the only consumer in group G1, and use it to subscribe to topic T1. consumer C1 will get all messages from all four t1 partitions

Info

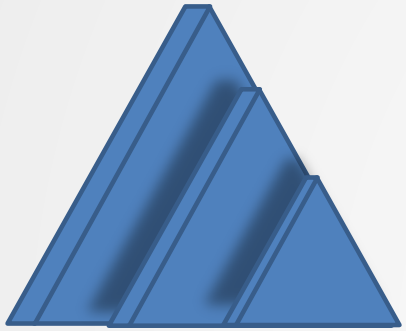
unable to keep up with the rate of incoming messages, always a need to scale consumption from topics



Kafka Consumer Groups [Case 2]

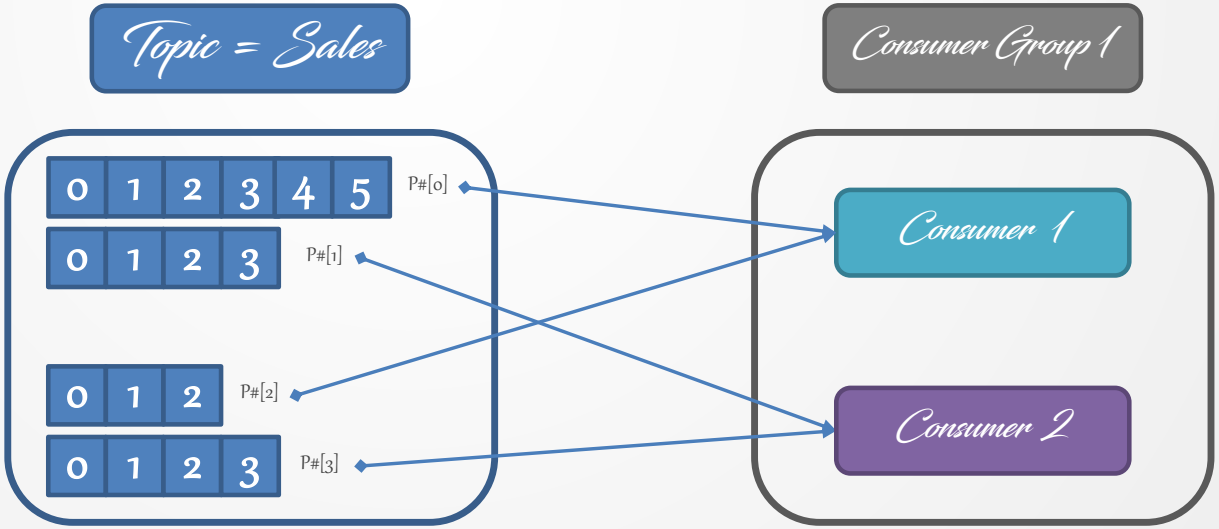
Info

read messages faster since there is now two consumers requesting data from the same topic in different partitions



Four Partitions Split to Two Consumer Groups

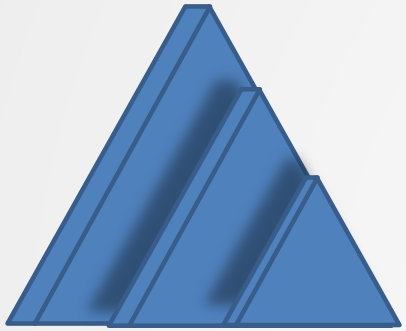
if we add another consumer, C2, to group G1, each consumer will only get messages from two partitions. perhaps messages from partition 0 and 2 go to C1 and messages from partitions 1 and 3 go to consumer C2



Kafka Consumer Groups [Case 3]

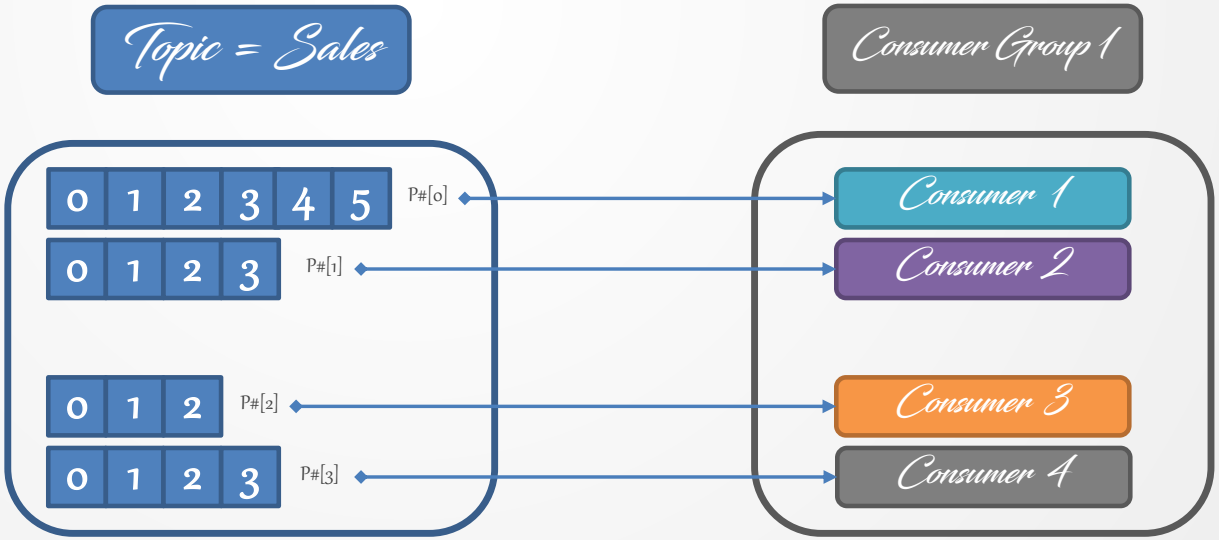
Info

every consumer now can pull data from a partition making the read lot faster

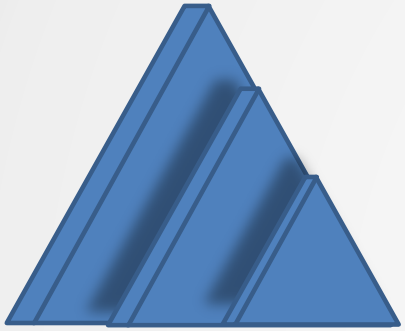


Four Consumer Groups to One Partition Each

if G1 has four consumers, then each will read messages from a single partition



Kafka Consumer Groups [Case 4]

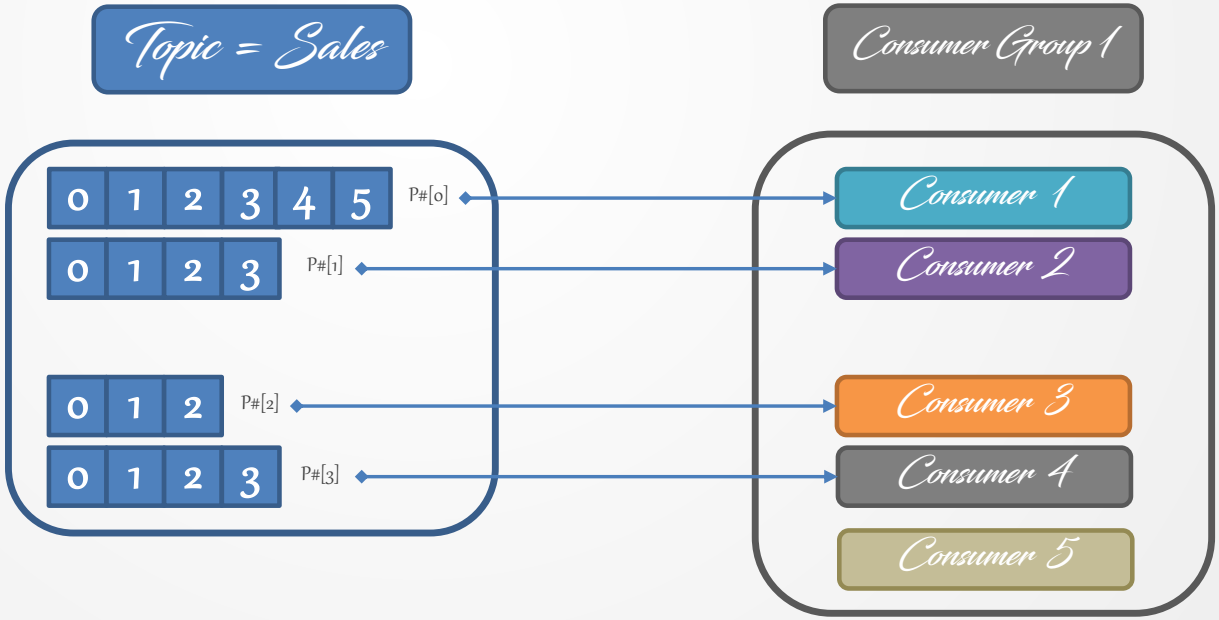


More Consumers Groups than Partitions

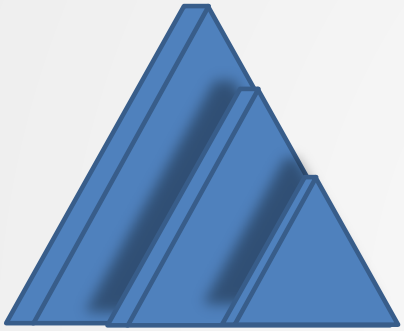
if we add more consumers to a single group with a single topic than we have partitions, some of the consumers will be idle and get no messages at all

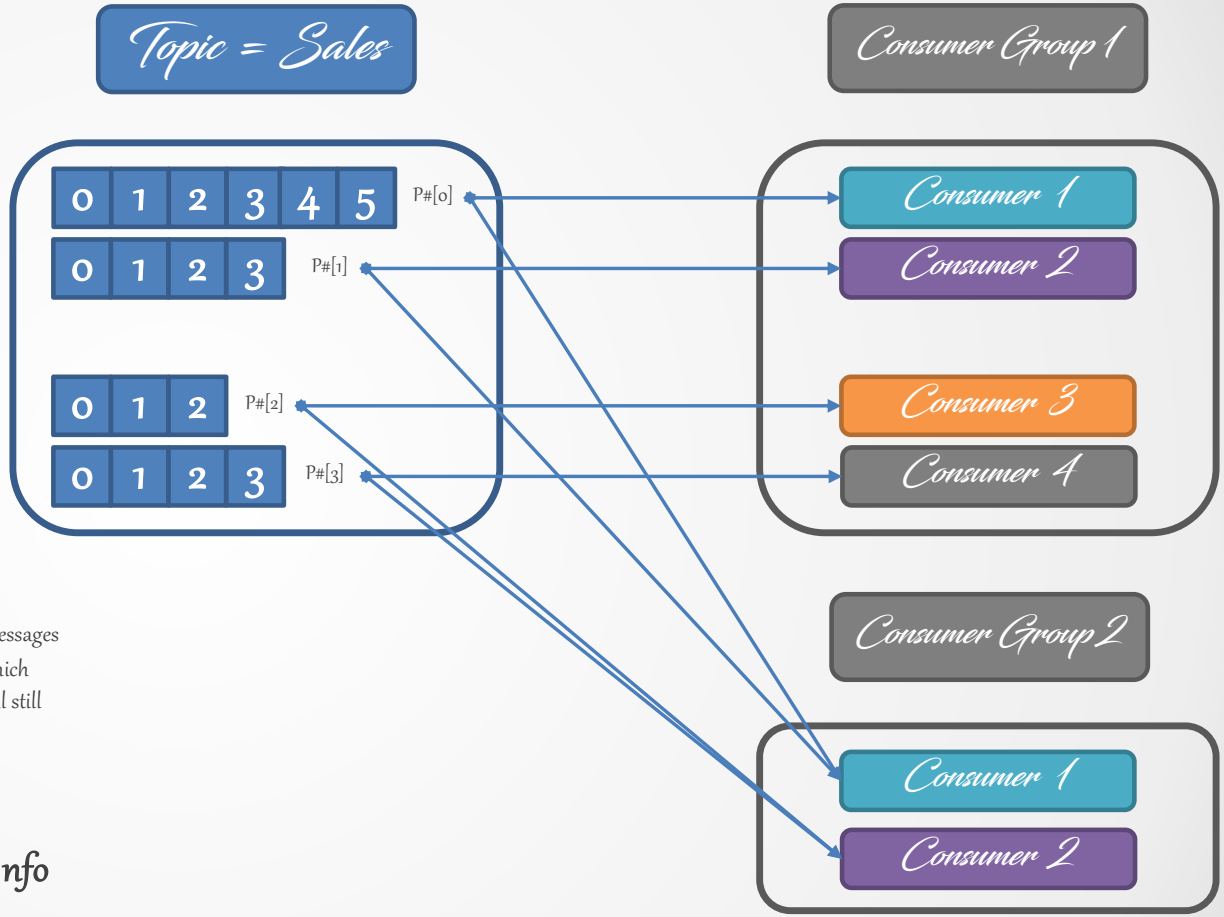
Info

there is no point in adding more consumers than you have partitions in a topic, some consumers will get idle



Kafka Consumer Groups [Case 5]





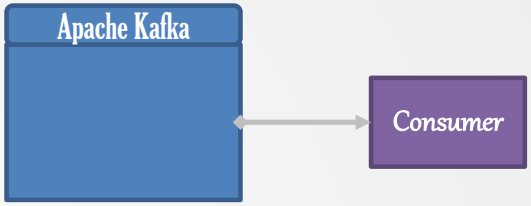
Adding New Consumer Group

if we add a new consumer group G2 with a single consumer, this consumer will get all the messages in topic T1 independent of what G1 is doing. G2 can have more than a single consumer, in which case they will each get a subset of partitions, just like we showed for G1, but G2 as a whole will still get all the messages regardless of other consumer groups.

Info

new consumer groups won't impact others that are consuming from the same topic. consumer groups are independent of each other

Kafka Consumer [Dissecting Read Process]



Kafka Consumer

the first step to start consuming records is to create a KafkaConsumer instance. creating a KafkaConsumer is very similar to creating a KafkaProducer - you create an instance with the properties you want to pass to the consumer. we will discuss all the properties. to start we just need to use the three mandatory properties: bootstrap.servers, key.deserializer, and value.deserializer.

1 Initial Configuration

start by adding the mandatory properties

1) Bootstrap.Servers
2) Key.Deserializer
3) Value.Deserializer

Recommendation = Group.Id

2 Topic Subscription

subscribe to one or more topics (use regex expression) for that

1) Topic Name

3 Poll Loop

the heart of the consumer API is a simple loop for polling data, it handles automatically

1) Coordination
2) Partition Rebalance
3) Heartbeats
4) Data Fetching

infinite loop = long-running application that polls from kafka.

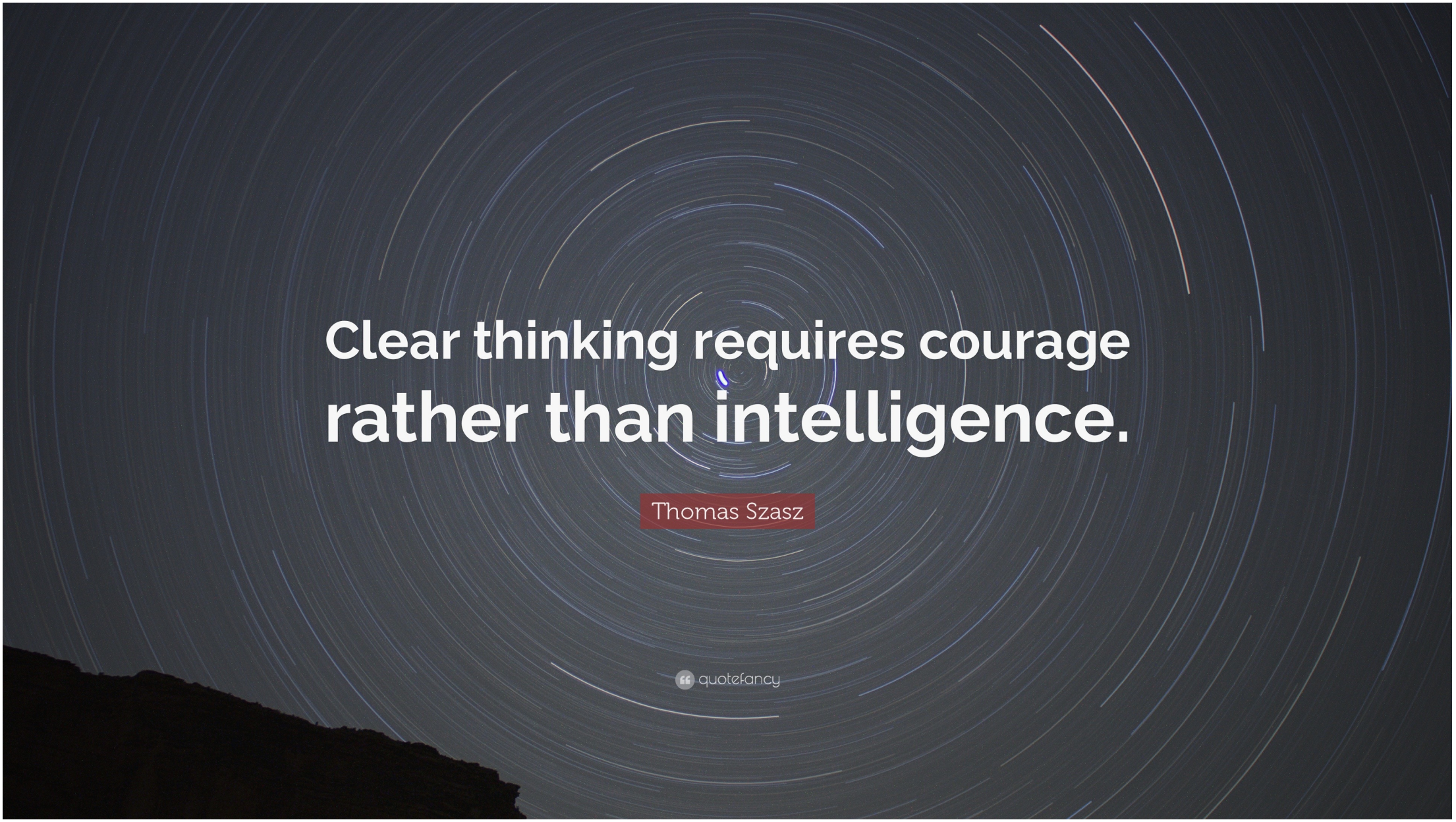
poll() = needs to keep polling data otherwise will be considered dead

first time consumer calls a poll() with a new consumer, it is responsible to

1) Find Group Coordinator
2) Join Consumer Group
3) Receive Partition Assignment
4) One Consumer per Thread

Building a Kafka Consumer Application using Python

1. Consumer Settings
2. Kafka Consumer API
3. Consume Data


Clear thinking requires courage
rather than intelligence.
Thomas Szasz
quotefancy

Kafka Consumer Groups & Partition Rebalance

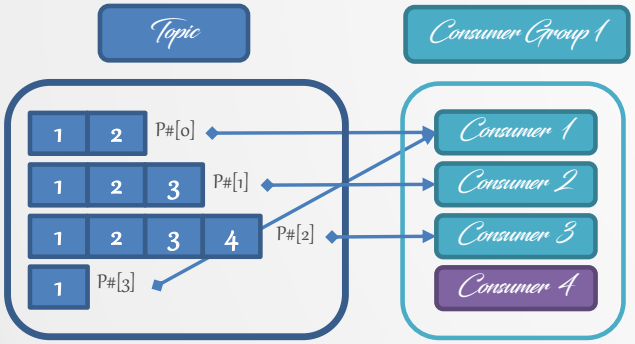
Rebalance

moving partition ownership from one consumer to another is called a rebalance. rebalances are important because they provide the consumer group with high availability and scalability (allowing us to easily and safely add and remove consumers), but in the normal course of events they are fairly undesirable. during a rebalance, consumers can't consume messages, so a rebalance is basically a short window of unavailability of the entire consumer group. in addition, when partitions are moved from one consumer to another, the consumer loses its current state; if it was caching any data, it will need to refresh its caches - slowing down the application until the consumer sets up its state again

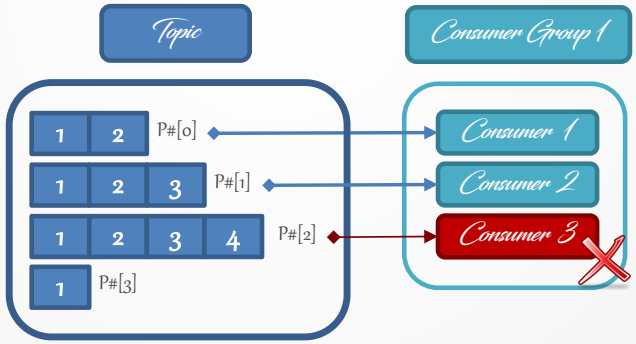
Info

- Share Ownership of Partitions ~ Topics
- Rebalance is a Window of Unavailability

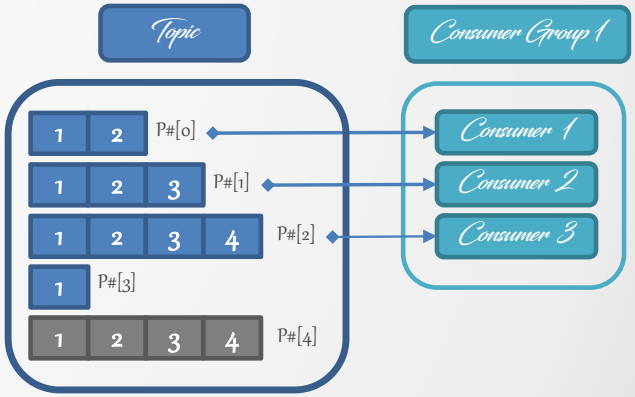
1. Add New Consumer
2. Consumer Shuts Down or Crashes
3. Topic is Modified = New Partitions Added



1 adding new consumer into the consumer group triggers a rebalance

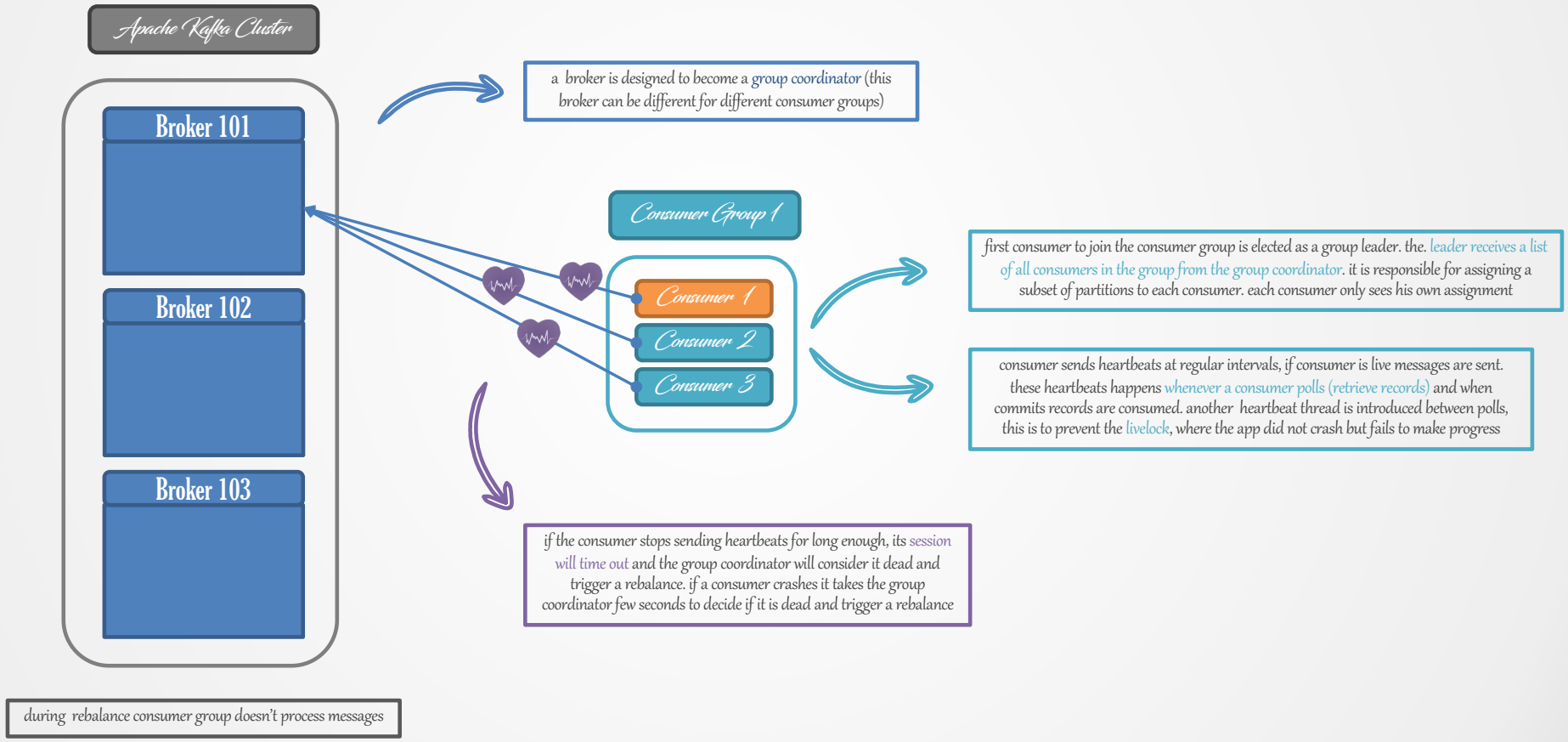


2 consumer shuts down or crashes triggering a rebalance process



3 adding a new partition into the topic triggers a rebalance

Kafka Consumer Groups & Membership



Kafka Consumer Groups [Commits & Offsets]

Commits & Offsets

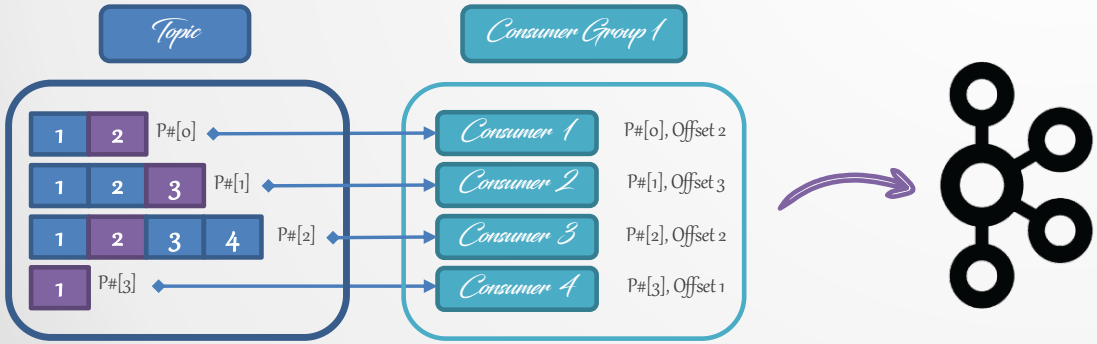
whenever we call poll(), it returns records written to kafka that consumers in our group have not read yet. this means that we have a way of tracking which records were read by a consumer of the group. As discussed before, one of kafka's unique characteristics is that it does not track acknowledgments from consumers the way many jms queues do. instead, it allows consumers to use kafka to track their position (offset) in each partition.

1 Commit

consumers use kafka to track their position, the action of updating the current position is what is call commit

2 Offset

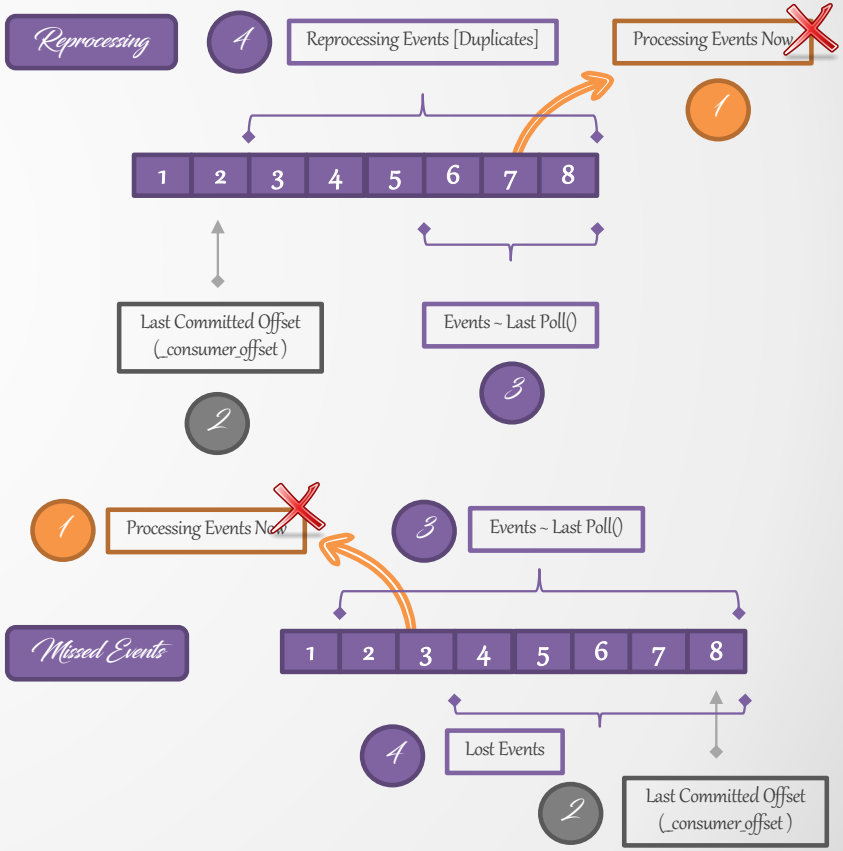
it produces a message to kafka, to a special _consumer_offset topic, with the committed offset for each partition



3 Rebalance

consumer will read latest committed offset of each partition

1) App Crash or Unexpected Shutdown
2) New Consumer into Consumer Group
3) Topic Changes



Kafka Consumer Groups [Commit Offset Options]

Modes

ways of committing offsets

Automatic Commit [Enable. Auto.Commit = True]

every five seconds the consumer will commit the largest offset your client received from poll(). the seconds can be controlled by setting Auto.Commit.Interval.ms

- does not give to developer enough control to avoid duplicate messages

Commit Current Offset [Enable. Auto.Commit = False]

used to reduce the possibility of missing messages and to reduce the number of duplicated messages during rebalancing. committed explicitly only and use commitSync(), throw an error if commit fails

- gives total control over the commit although it's a sync process (limit throughput)

Async Commit

another option is the async option where instead of waiting to respond to a commit, we just send the request and continue to process upcoming messages, problem is that commitAsync() won't retry, possibility to add a callback to log errors

- it's faster but does not allow retry, if the commit fails can leads to duplicated messages

Combine Synchronous & Asynchronous Commits

common pattern to combine the both methods before shutdown, if everything is fine the Async() is called since it's faster but if fails the Sync() is invoked and will retry until it succeeds or suffers unrecoverable failure

- speediness for normal writes and security whenever a failure or wait happens

commit current offset

```
while (true) {
  ConsumerRecords<String, String> records = consumer.poll(100);
  for (ConsumerRecord<String, String> record : records)
  {
    System.out.printf("topic = %s, partition = %s, offset = %d, customer = %s, country = %s\n",
    record.topic(), record.partition(),
    record.offset(), record.key(), record.value());
  } try {
      consumer.commitSync();
  } catch (CommitFailedException e) {
      log.error("commit failed", e) }
}
```

async commit

```
while (true) {
  ConsumerRecords<String, String> records = consumer.poll(100);
  for (ConsumerRecord<String, String> record : records)
  {
      System.out.printf("topic = %s, partition = %s, offset = %d, customer = %s, country = %s\n",
      record.topic(), record.partition(), record.offset(), record.key(), record.value());
  }
consumer.commitAsync();
}
```

sync & async commit

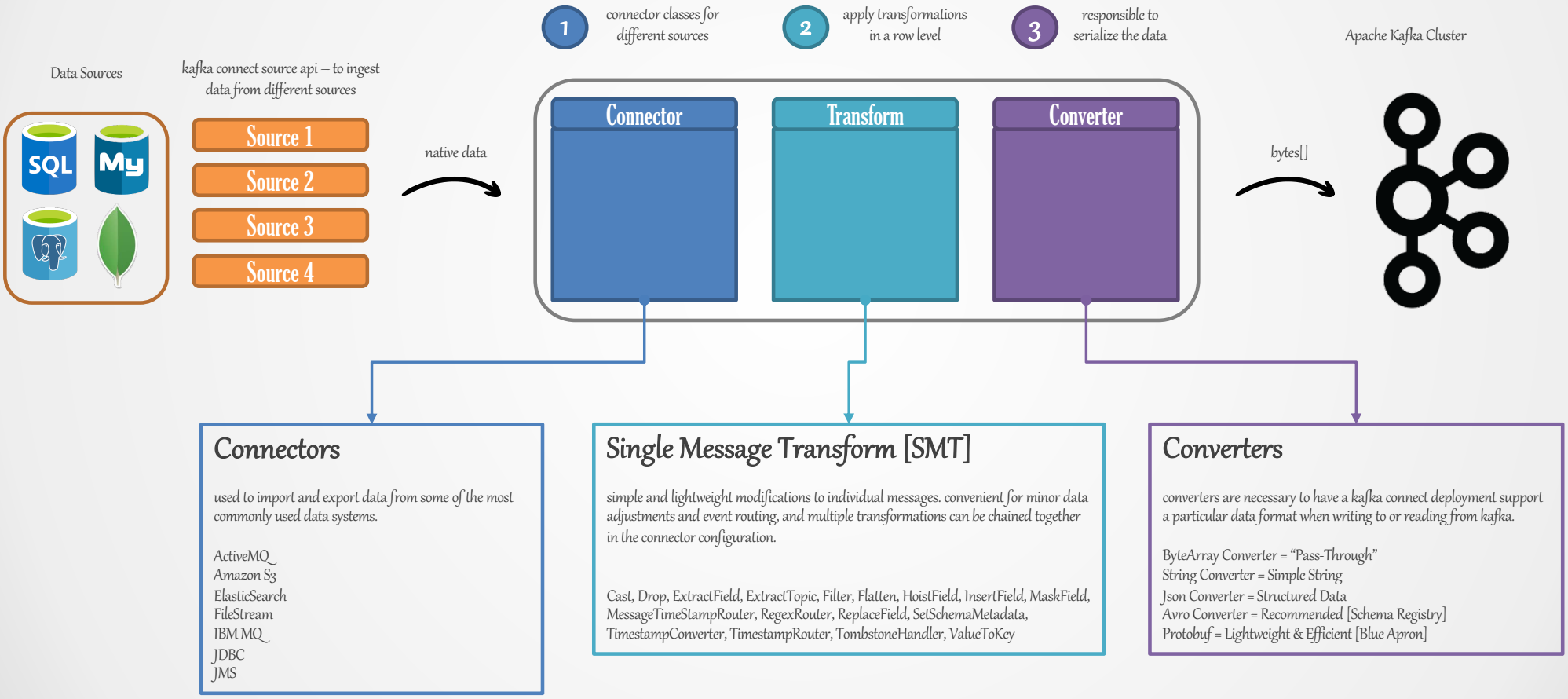
```
try {
      while (true) {
        ConsumerRecords<String, String> records = consumer.poll(100);
        for (ConsumerRecord<String, String> record : records) {
          System.out.printf("topic = %s, partition = %s, offset = %d, customer = %s, country = %s\n",
          record.topic(), record.partition(),
          record.offset(), record.key(), record.value());
        }
        consumer.commitAsync();
      }
  } catch (Exception e) { log.error("Unexpected error", e);
  } finally {
      try {
        consumer.commitSync();
      } finally {
        consumer.close(); }
  }
```

Scaling-Out Kafka Consumer Application

1. Spin Up Python App Consumer
2. Verify Consumer Group ~ Scaling App


Courage is never to let your actions be influenced by your fears.
Arthur Koestler
quotefancy

Kafka Connect [Data Movement]



Connectors

used to import and export data from some of the most commonly used data systems.

ActiveMQ
Amazon S3
ElasticSearch
FileStream
IBM MQ
JDBC
JMS

Single Message Transform [SMT]

simple and lightweight modifications to individual messages. convenient for minor data adjustments and event routing, and multiple transformations can be chained together in the connector configuration.

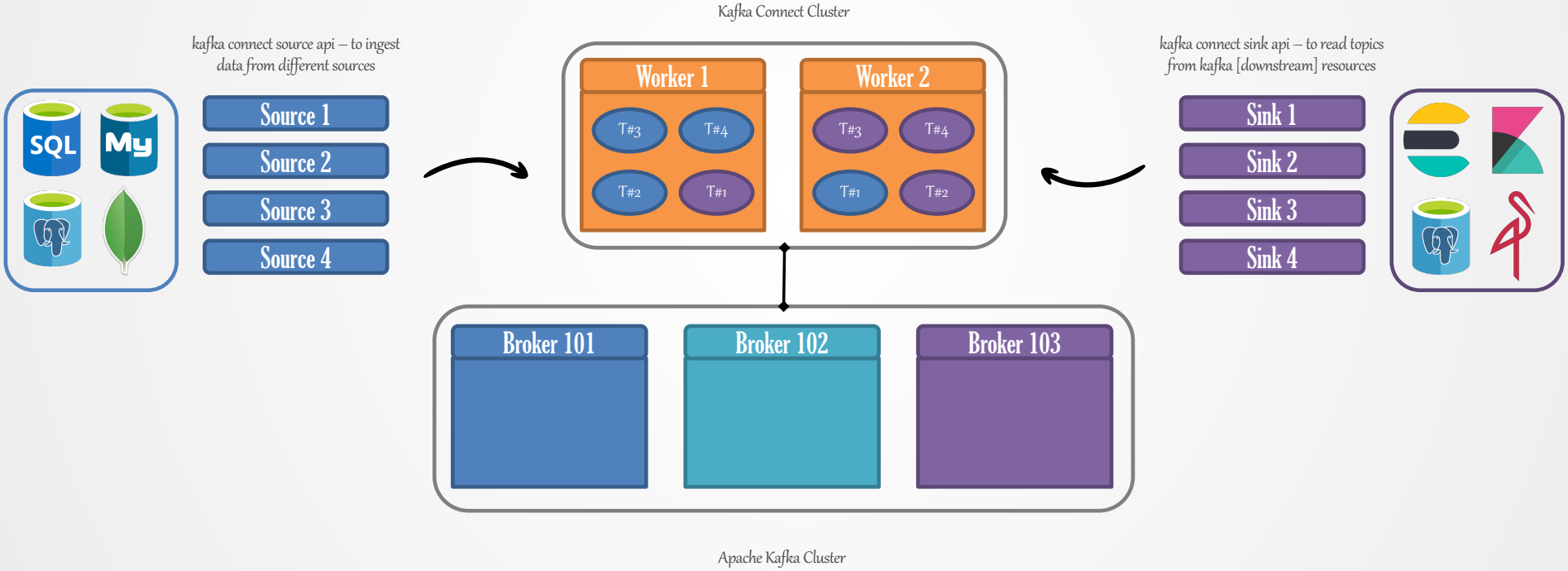
Cast, Drop, ExtractField, ExtractTopic, Filter, Flatten, HoistField, InsertField, MaskField, MessageTimeStampRouter, RegexRouter, ReplaceField, SetSchemaMetadata, TimestampConverter, TimestampRouter, TombstoneHandler, ValueToKey

Converters

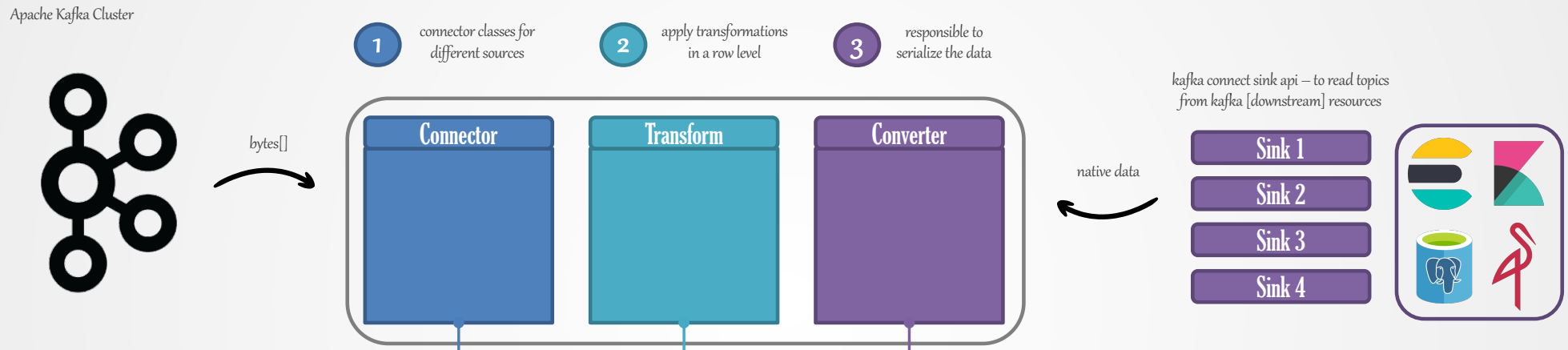
converters are necessary to have a kafka connect deployment support a particular data format when writing to or reading from kafka.

ByteArray Converter = "Pass-Through"
String Converter = Simple String
Json Converter = Structured Data
Avro Converter = Recommended [Schema Registry]
Protobuf = Lightweight & Efficient [Blue Apron]

Kafka Connect [Sources & Sinks]



Kafka Connect Sink [Movement]



Connectors

used to import and export data from some of the most commonly used data systems.

ActiveMQ_
Amazon S3
ElasticSearch
FileStream
IBM MQ_
JDBC
JMS

Single Message Transform [SMT]

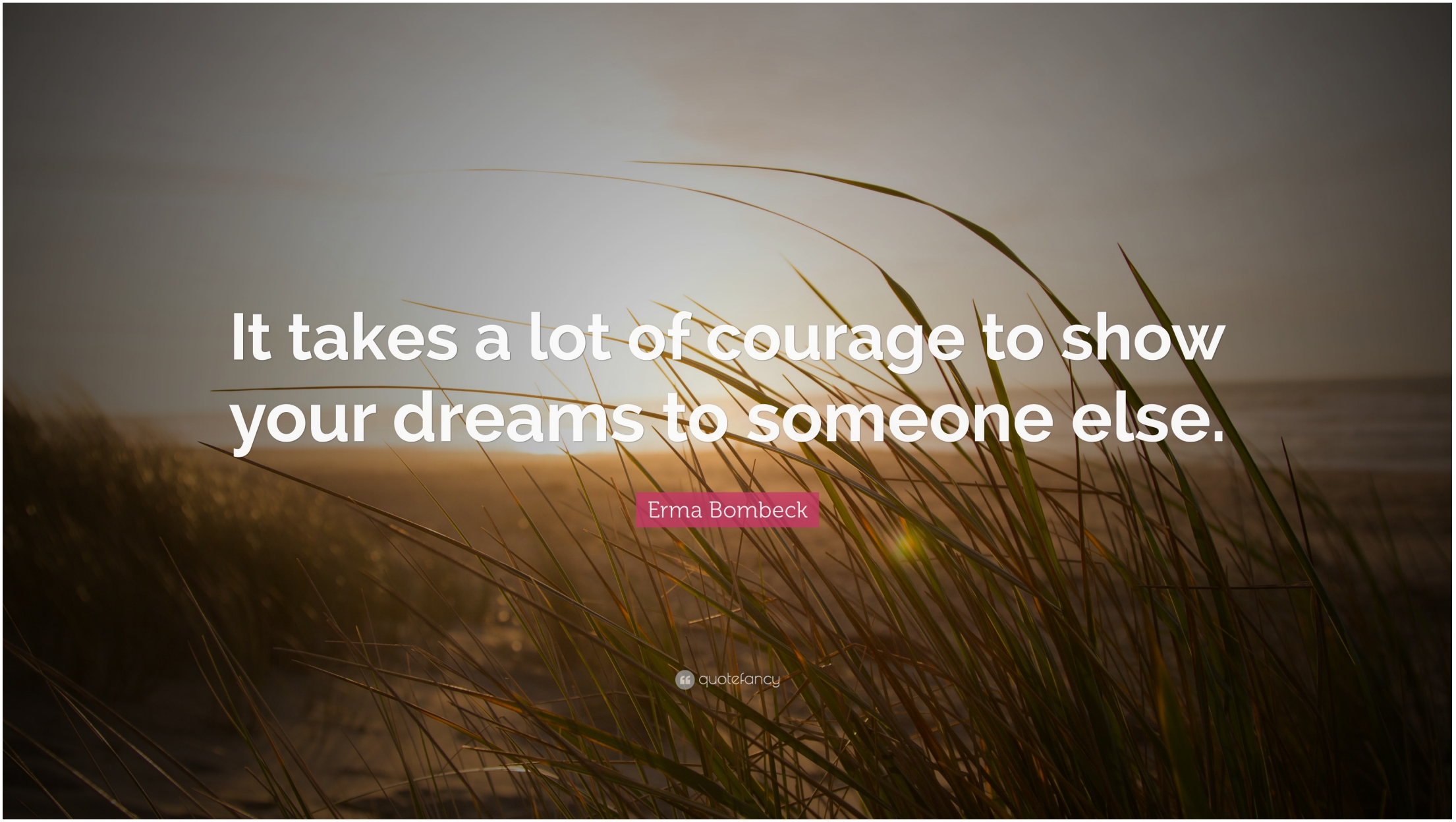
simple and lightweight modifications to individual messages. convenient for minor data adjustments and event routing, and multiple transformations can be chained together in the connector configuration.

Cast, Drop, ExtractField, ExtractTopic, Filter, Flatten, HoistField, InsertField, MaskField, MessageTimeStampRouter, RegexRouter, ReplaceField, SetSchemaMetadata, TimestampConverter, TimestampRouter, TombstoneHandler, ValueToKey

Converters

converters are necessary to have a kafka connect deployment support a particular data format when writing to or reading from kafka.

ByteArray Converter = "Pass-Through"
String Converter = Simple String
Json Converter = Structured Data
Avro Converter = Recommended [Schema Registry]
Protobuf = Lightweight & Efficient [Blue Apron]


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

Kafka Connect [ElasticSearch] ~ Sink

ElasticSearch Service Sink Connector for Confluent Platform

the service moves data from apache kafka to elasticsearch. it writes data from a topic to an index in elasticsearch. all data for a topic have the same type in elasticsearch. this allows an independent evolution of schemas for data from different topics. this simplifies the schema evolution because elasticsearch has one enforcement on mappings; that is, all fields with the same name in the same index must have the same mapping type.

ElasticSearch

- Distributed Open-Source Analytics Engine
- Built-On Apache Lucene ~ Released in 2010
- Simple REST APIs
- Powerful Search Engine for = App, Website, Logging, Business

Features

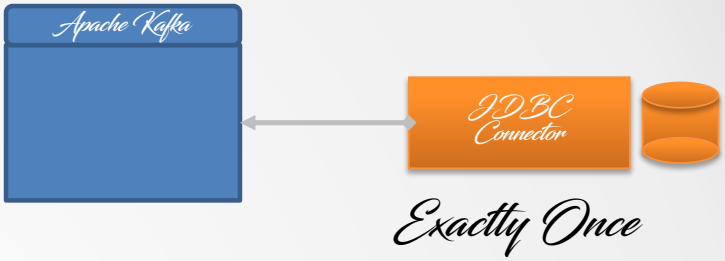
Mapping Inference
- Infer Mappings from Connect Schemas

Schema Evolution
- Backward, Forward & Fully Compatible Schema

Delivery Semantics
- Batching & Pipelined Writes ~ Boost Throughput
- Concurrent Processing of Multiple Batches

Automatic Retries
- Retries Requests Automatically

Kafka Connect [JDBC Sink Connector]



JDBC Connector [Source & Sink] for Confluent Platform

you can use the kafka connect jdbc source connector to import data from any relational database with a jdbc driver into apache kafka® topics. you can use the jdbc sink connector to export data from kafka topics to any relational database with a jdbc driver. the jdbc connector supports a wide variety of databases without requiring custom code for each one.

Features

Data Mapping
- Apache Avro & Schema Registry
- JSON with Embedded Schema

Key Handling
- PK.Mode = None, Kafka [Topic, Partition, Offset], Record Key & Record Value

Delete Mode
- Consume Tombstone Record [Key with Null Value & PK.Mode = Record Key

Idempotent Writes
- Insert Mode = Insert or Upsert

Auto-Creation & Auto-Evolution
- Auto.Create = Automatic Table Creation
- Auto.Evolve = Alter Table Statement

Relational Database Systems [RDBMS]

- Microsoft SQL Server
- PostgreSQL
- Oracle Database
- IBM DB2
- MySQL Server
- SAP HANA
- SQLite Embedded Database

Kafka Connect [MongoDB] ~ Sink

MongoDB Connector

The MongoDB Connector Officially Supported By MongoDB, Can Provide ~ Source and Sink Connectors Install One Connector and Ingestion and Output in MongoDB Database

NoSQL ~ MongoDB

- CAP Base Theorem
- Document Database [Json]
- Fast Development Time
- High Performance
- Rich Query Language
- High Availability
- Horizontal Scalability
- Storage Engines – WiredTiger & In-Memory

Features

Unique Document ID Strategy

- document.id.strategy
- document.id.strategy.overwrite.existing
- delete.on.null.values

Schemas:

- Apache Avro & JSON Format

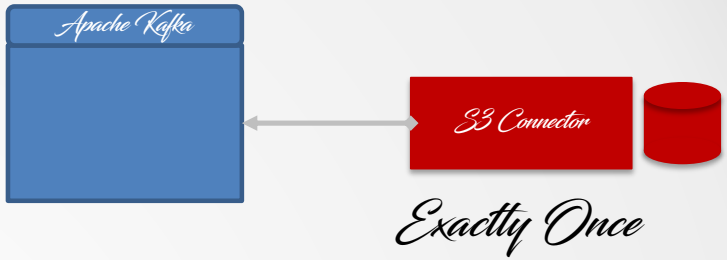
Dead Letter Queue Configuration

- Specifies Invalid Messages

Enable Mapper

- Setting namespace.mapper Using Class: com.mongodb.kafka.connect.sink.namespace.mapping.FieldPathNamespaceMapper

Kafka Connect [S3] ~ Sink



Amazon S3 Sink Connector for Confluent Platform

the S3 connector, currently available as a sink, allows you to export data from kafka topics to S3 objects in either avro, json and parquet formats. in addition, for certain data layouts, S3 connector exports data by guaranteeing exactly-once delivery semantics to consumers of the S3 objects it produces.

Features

Exactly Once Delivery
- Using a Deterministic Partitioner
- Eventual Consistency

Pluggable Data Format with or without Schema
- Apache Avro, Apache Parquet & JSON Format

Pluggable Partitioner
- Default, Custom & Time Based

Non-AWS Object Storage Support
- Minio

Partitioning Records
- Default
- Field
- Time
- Daily
- Hourly

MinIO

- High Performance Kubernetes Native Object Storage
- Designed ~ S3 API
- 100% Open-Source
- Erasure Coding [Written in Assembly]
- Encryption
- Identity Management
- Continuous Replication
- Federation
- Multi-Cloud Gateway [S3, Blob Storage & GCS]

Kafka Connect Sink [Deploying Connectors]

1. Meet Data Sinks
2. Deploying Connectors

MDW & OLAP Systems

Cloud-Based Data Warehouses Systems

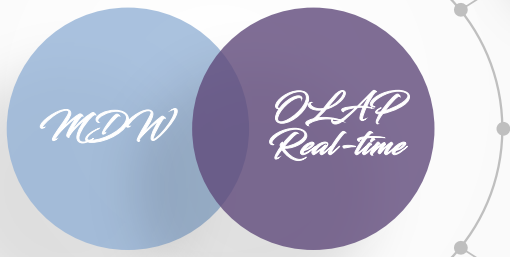
- Amazon Redshift
- Azure Synapse Analytics
- Google BigQuery
- Snowflake

Physical Hardware

- Massively Parallel Processing [MPP]
- "Loosely Coupled" & "Shared Nothing"

Columnar Data Storage Type

- Batch-Processing Mode
- Compression Benefits
- I/O Reduction Operations
- Index Bitmap



Real-Time Ingest

- Exactly Once Semantics
- Directly Source from Apache Kafka as Consumer
- Designed for Streaming Technologies

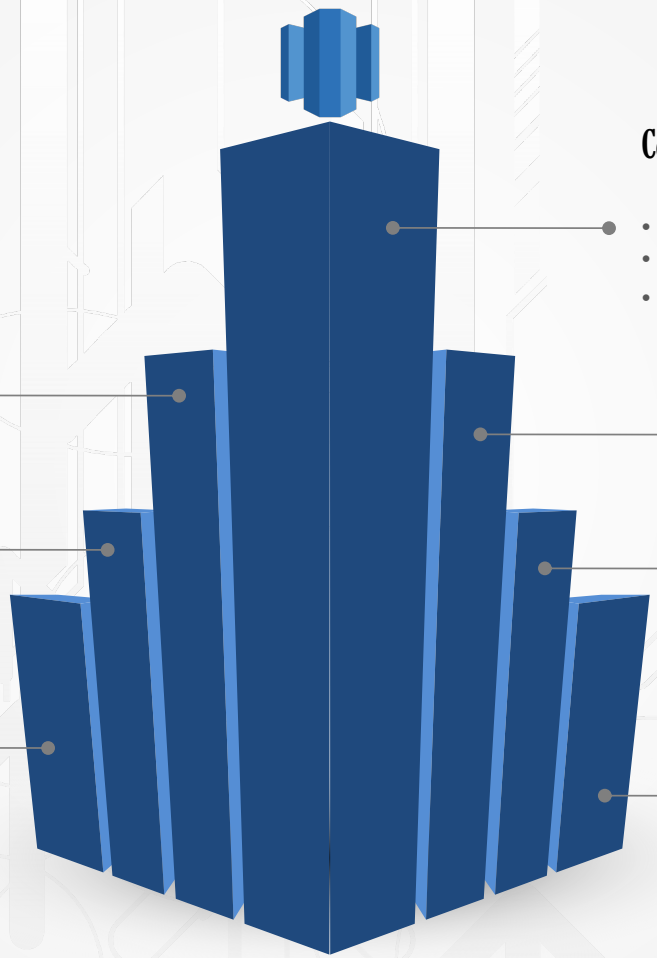
Users

- SQL using Apache Calcite
- API using RestAPI Calls

Key Capabilities

- Optimized for Multiple Connections
- Columnar-Oriented Storage
- Horizontally Scale & Fault-Tolerant
- Joins using Presto & Trino

Amazon Redshift



Integrations

- AWS S3 Data Lake
- AWS Glue
- Amazon Kinesis Firehose
- Amazon QuickSight
- Database Migration Service

Faster Performance

- Massively Parallel Processing
- Machine Learning
- Result Caching

Deploy & Manage

- Automated Provisioning
- Automated Backups
- Fault Tolerant
- Flexible Querying
- Third-Party Tools Integration

Cost-Effective

- Pay as you Go
- Predictable Cost
- Node Type

Amazon Redshift

Most Popular Cloud Data Warehouse

> 15K Customers using Amazon Redshift

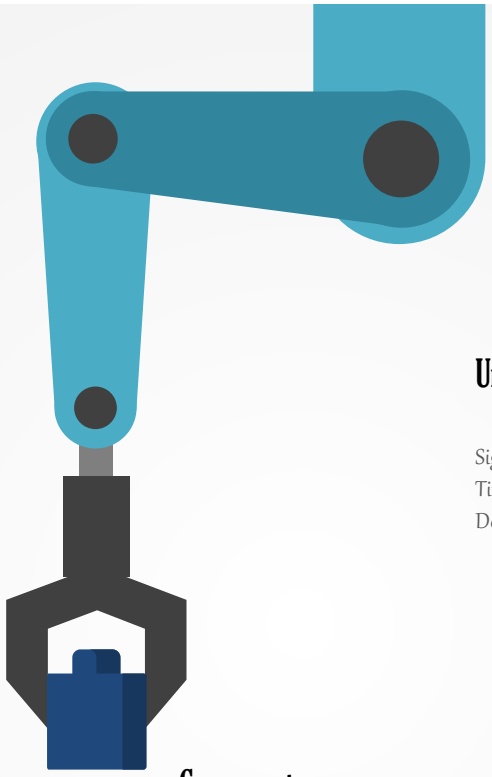
Scalability

- Petabyte-Scale
- Exabyte-Scale Data Lake Analytics with Spectrum
- Limitless Concurrency

Features

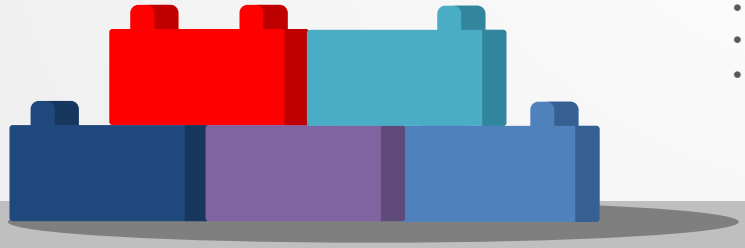
- Query and Export Data
- Federated Query
- Redshift ML
- Redshift Partner Console Integration

Azure Synapse Analytics



Features

- EDW - Azure SQL Dw Engine
- Data Lake Exploration
- Languages – T-SQL, Python, Scala, Spark SQL & .NET
- Orchestration – Azure Data Factory
- Streaming Ingestion & Analytics
- Integrated AI & BI



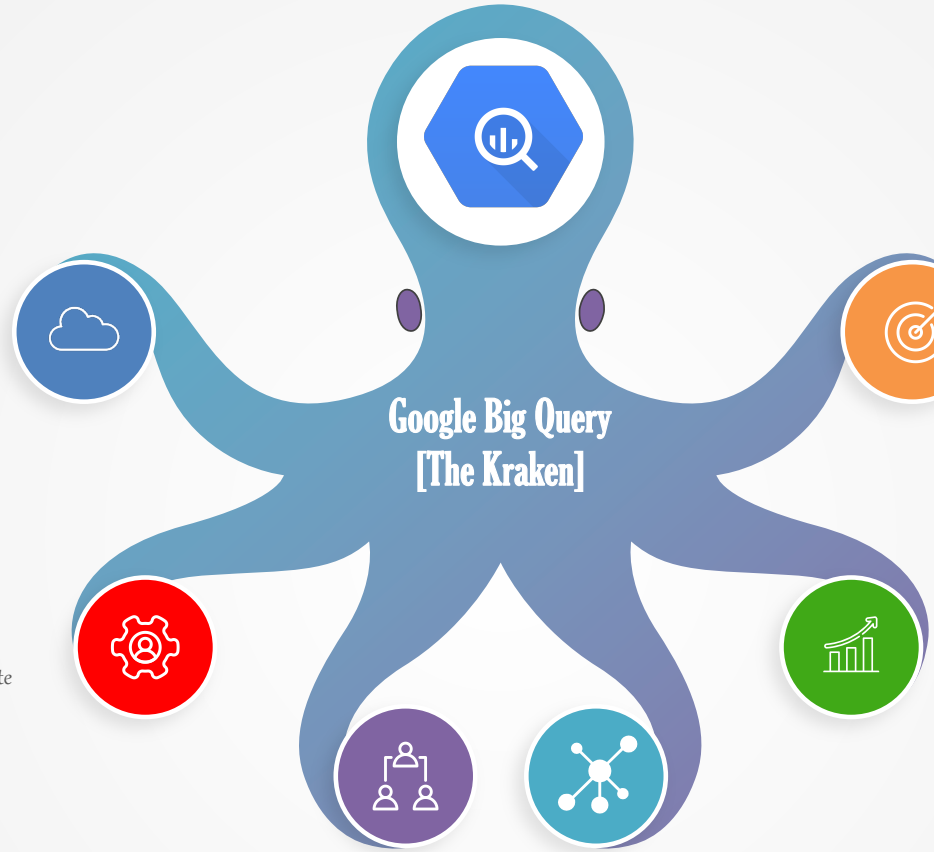
Components

- Built-In SQL Pool
- DW using - SQL Dw
- Apache Spark Pools
- Data Integration & Studio

Unified Experience

Significantly Reduce Project Development Time with a Unified Experience for Developing End-to-End Analytics Solutions

Google BigQuery



Google's Serverless Offering

- Automatic Resource Provisioning
- SaaS Offering for Dw

Foundation for BI & AI

- EDW Google's Offering
- Integration, Transformation & Analyzes
- TensorFlow & BigQuery ML

Storage & Computation

- Separated Storage & Compute
- Choose Storage Tier
- Control Costs

Real-Time Analytics with SQL

- High-Speed Streaming Insertion API
- Standard ANSI:2011 SQL Support
- ODBC & JDBC Drivers

Programmatic Interaction

- REST API
- Java
- Python
- Node.js
- C#
- Ruby
- PHP

Big Data Ecosystem Integration

- Cloud DataProc
- Cloud DataFlow
- Apache Big Data Ecosystem
- Apache Hadoop & Apache Beam

Snowflake

Fast and Easy SQL Analytics

- In Seconds, Provision Compute Clusters Ranging in Size from Extra-Small ~ 6XL
- Fully ANSI SQL Compatible, with Native Support for Semi-Structured Data
- Optimized Direct Connectors for Popular BI and Analytics Tools

Single Source for All Data

Creates a Single, Governed, and Immediately Queryable Source for Effectively All Your Data, Including JSON and XML, with Near-Unlimited and Cost-Effective Storage

Use All of Your Data

- Near-Unlimited, Low-Cost Cloud Storage with 2-3x Compression
- A Single Copy of Your Data Available Immediately Everywhere
- Native Support for Geospatial Data and Analysis

Managed Service

- Automatic Query Caching, Planning, Parsing, and Optimization
- Automatic Updates with No Scheduled Downtime
- Cross-Cloud Data Replication for Seamless, Global Data Access

Accelerate BI & Analytics for All Users

Support a Virtually Unlimited Number of Concurrent Users and Queries with Near-Unlimited, Dedicated Compute Resources. Query Semi-Structured Data Directly with SQL, and with Your Favorite Business Intelligence and Machine Learning Tools

Security, Governance & Privacy

- Automatic Data Encryption At-Rest and In-Transit
- Dynamic Data Masking and Tokenization
- Leverage Functionality ~ GDPR and CCPA Compliance
- Certifications for SOC2 Type 2, ISO 27001, PCI, HIPAA, FedRAMP

Apache Pinot

Characteristics

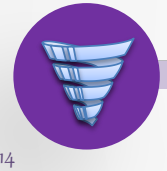
- Blazing Fast OLAP Engine
- LinkedIN, Uber, Target, Slack, Stripe
- Designed for Real-Time Analytics

History

- Pinot Noir ~ Name of Grape for Wine
- Developed Internally at LinkedIN in 2014
- Open-Sourced in 2015
- Entered in Apache Incubator in 2018

Key Capabilities

- Columnar-Oriented Storage
- Pluggable Indexing Technologies
- Horizontally Scale & Fault-Tolerant
- Performs Anomaly Detection using ThirdEye
- Joins using Presto & Trino



Apache Pinot

Realtime Distributed OLAP DataStore, Designed for Answering OLAP Queries with Low-Latency

Performance

- 100K + Queries per Second at ms Latency
- Ingestion of Million of Events per Second
- 50+ User-Facing Analytics

Users

- PQL ~ SQL using Apache Calcite
- API ~ RestAPIs for Broker & Controller
- External Clients ~ JDBC, Java, Python & Go

Core Components

- Controller (Helix & Zookeeper) ~ Manage State & Health
- Broker ~ Route Queries
- Server ~ Host Segments (Data) ~ Realtime & Offline Tables
- Minion ~ Purge Data

Data Virtualization

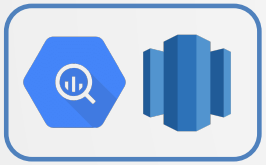
Access Data in Any Source, Shape or Structure without Copying from Source

Processing Query Engine

Connects, Discover, Plan and Analyze Different Data Stores for Query at Scale

Modern Data Warehouses

Redshift, BigQuery

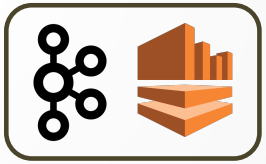


Real-Time OLAP

Apache Druid & Apache Pinot

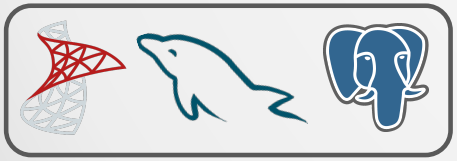
Real-Time Ingestion

Apache Kafka, Kinesis



Relational Databases

SQL Server, MySQL, Postgres



NoSql

Cassandra, MongoDB, Redis, ElasticSearch

Data Lake

HDFS, S3, Blob Storage, GCS & MinIO [S3*]

Trino

Fastest Bunny on Forest, Eating Any Source of Data

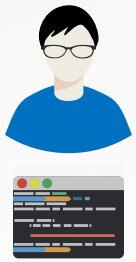
Trino

- Speed & Scale
- Simplicity
- Versatile
- In-Place Analysis
- Query Federation
- Runs Everywhere
- Trusted
- Trino Software Foundation [OSS]

Designed For

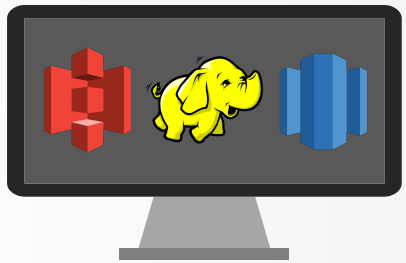
- Access Data using SQL
- Data Warehousing
- Analytics
- OLAP

Trino is a Tool Designed to Efficiently Query Vast Amounts of Data using Distributed Queries in a Terabytes & Petabytes Range



Query Execution Model

Turn SQL Statements into Queries, Executes Across a Distributed Cluster of Coordinator and Workers

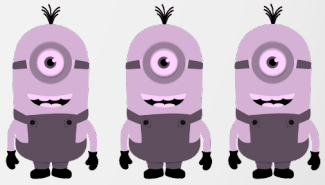


Coordinator a.k.a The Brain

Parsing Statements, Planning Queries, & Managing Trino Worker Nodes

Workers a.k.a The Slave

Responsible for Executing Tasks & Processing Data. Fetch Data from Connectors and Exchange Intermediate Data in Parallel

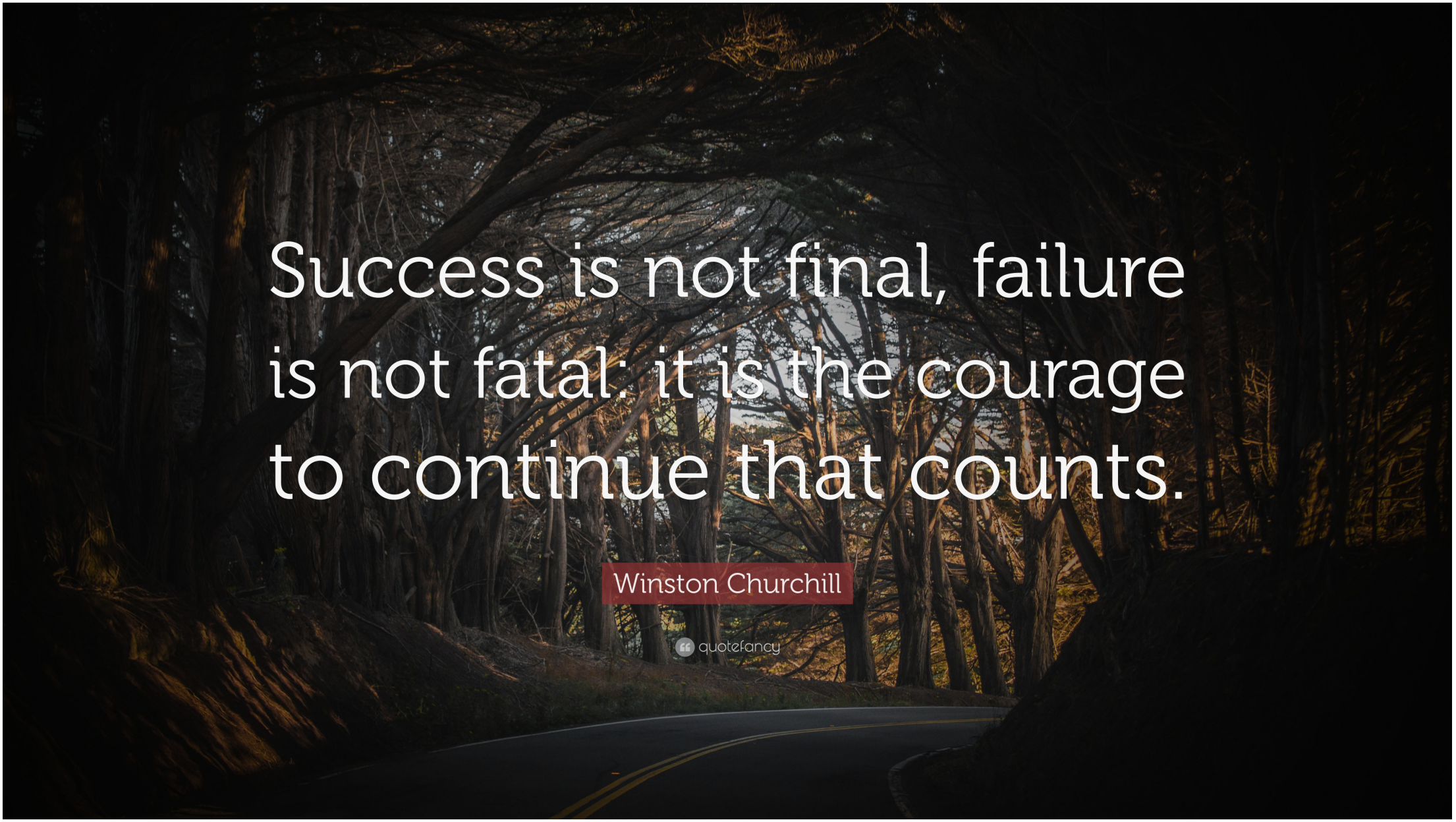


Features

- Memory Management & Spilling using Compression and Encryption
- Resource Groups
- Distributed Sort
- Dynamic Filtering ~ Query Performance, Network Traffic Reduction & Remote Load Reduction
- Cost Based Optimization - CBO
- Query Pushdown

Apache ORC ~ Optimized Row Columnar

- Smallest, Fastest Columnar Storage for Apache Hadoop Workloads
- ACID Support
- Built-In Indexes
- Complex Types
- Integration with Apache Spark & Trino


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


ONEWAY
SOLUTION