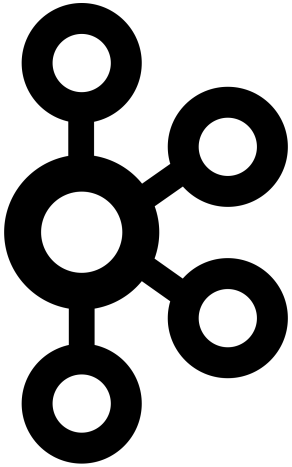


ES & Kafka

Diego Pacheco

About me...

- Cat's Father
- Principal Software Architect
- Agile Coach
- SOA/Microservices Expert
- DevOps Practitioner
- Speaker
- Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

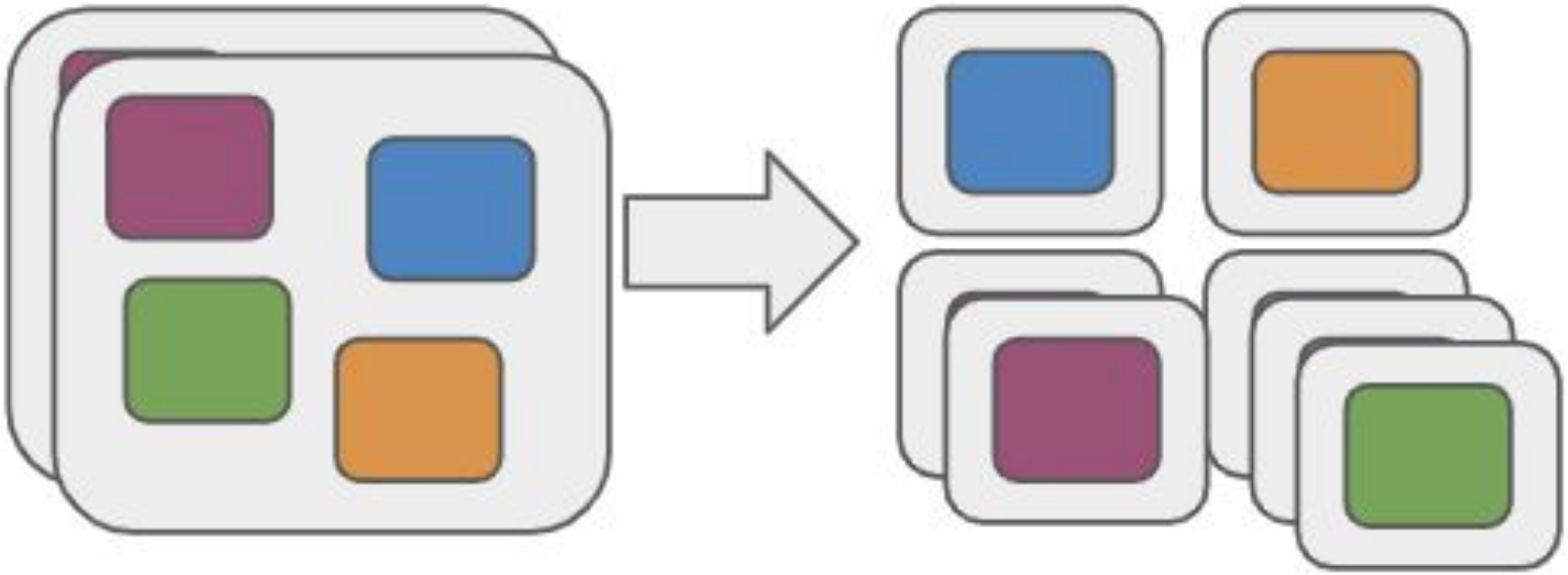
https://diegopacheco.github.io/

ES/CQRS

CQRS

Command Query Responsibility Segregation

#1 Reason why we need ES == Microservices




PRINCIPLES
VALUE
STRATEGY
STATEMENT
MISSION
INTEGRITY
CONDUCT
INNOVATION
STATEMENT
BUSINESS
COMPANY
PLAN
OBJECTIVES
IDEOLOGY
INNOVATION
CUSTOMER
ACCOUNT
VISION
EMPLOYEES
EXPENSE
PRICE
GUIDING
RULE
VISION
COMMERCE
GOALS
EXCELLENCE

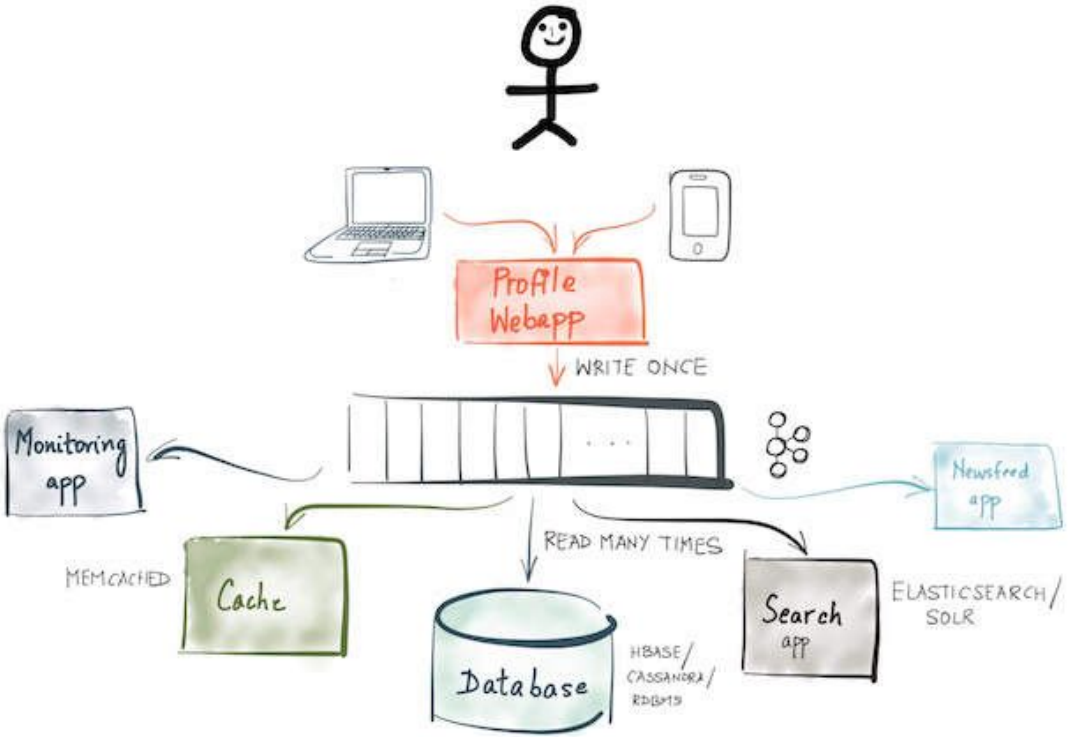
No Places, RAW Events.

Immutability == Append ONLY

Segregation == Scalability

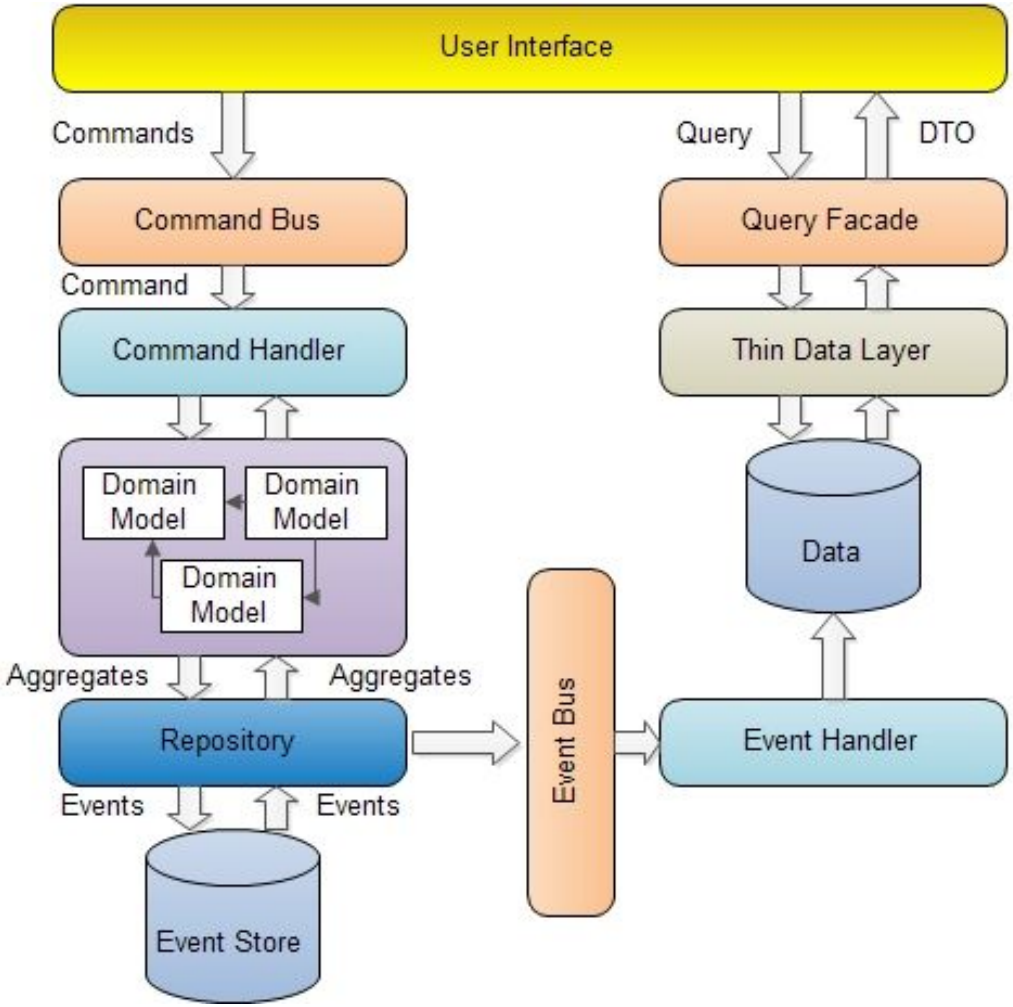
ES

Persists RAW EVENTS.
When there are mutations, NEW events are created.

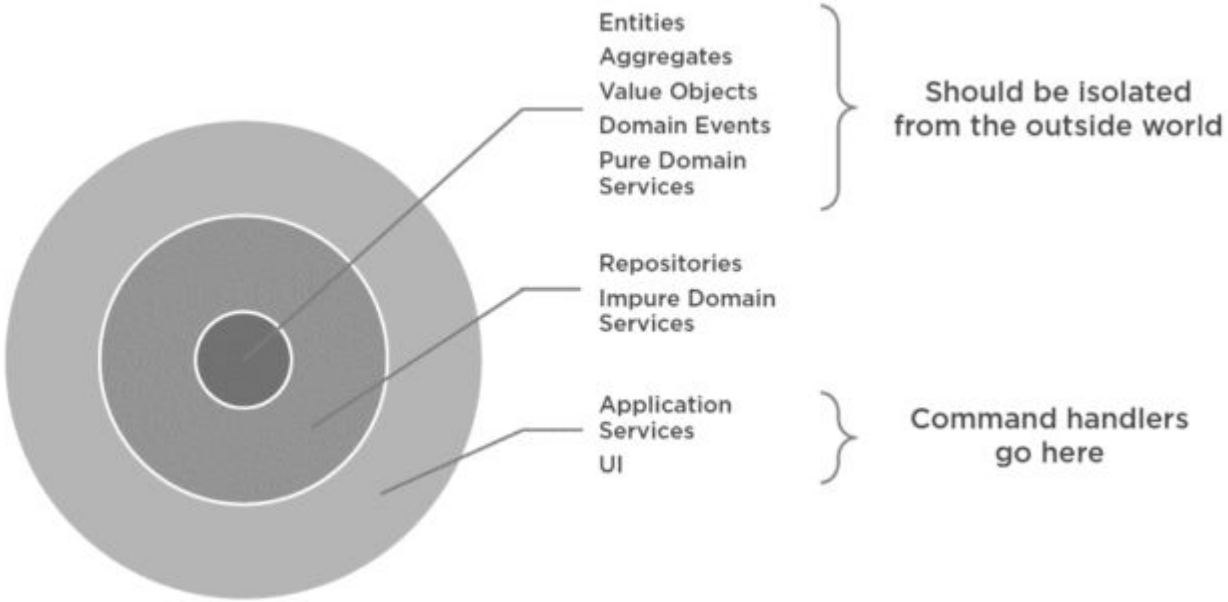


CQRS

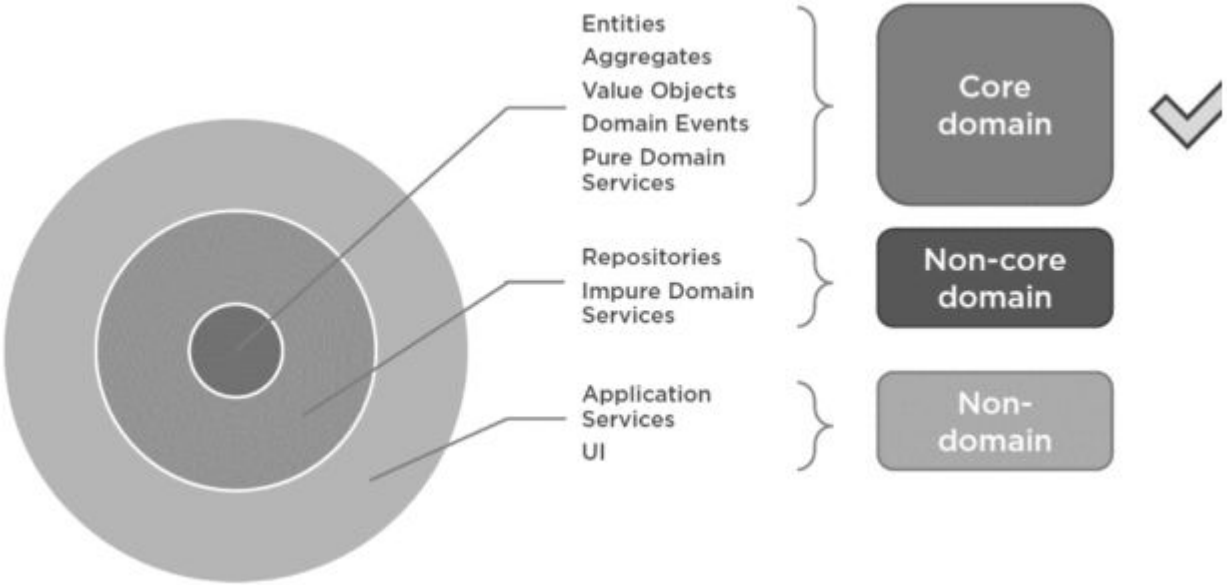
IT's about scaling READ and WRITES, basically in different RELATIONAL DBS.



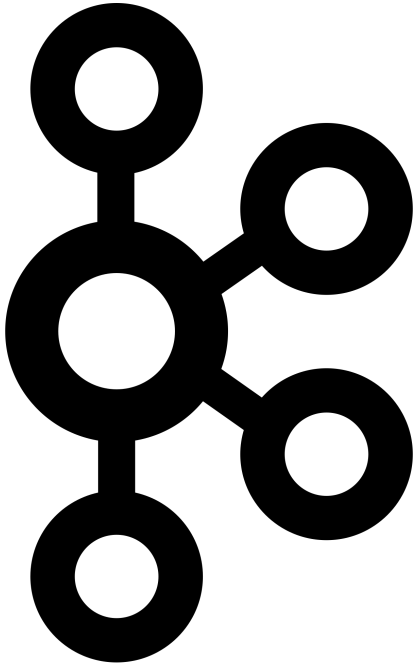
CQRS / ES :: Design



CQRS / ES :: Design



KAFKA



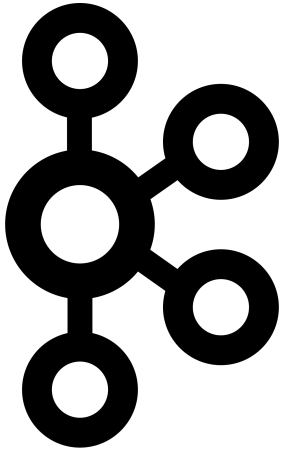
7 Trillion

messages per day.

* 2019 numbers

Use Cases

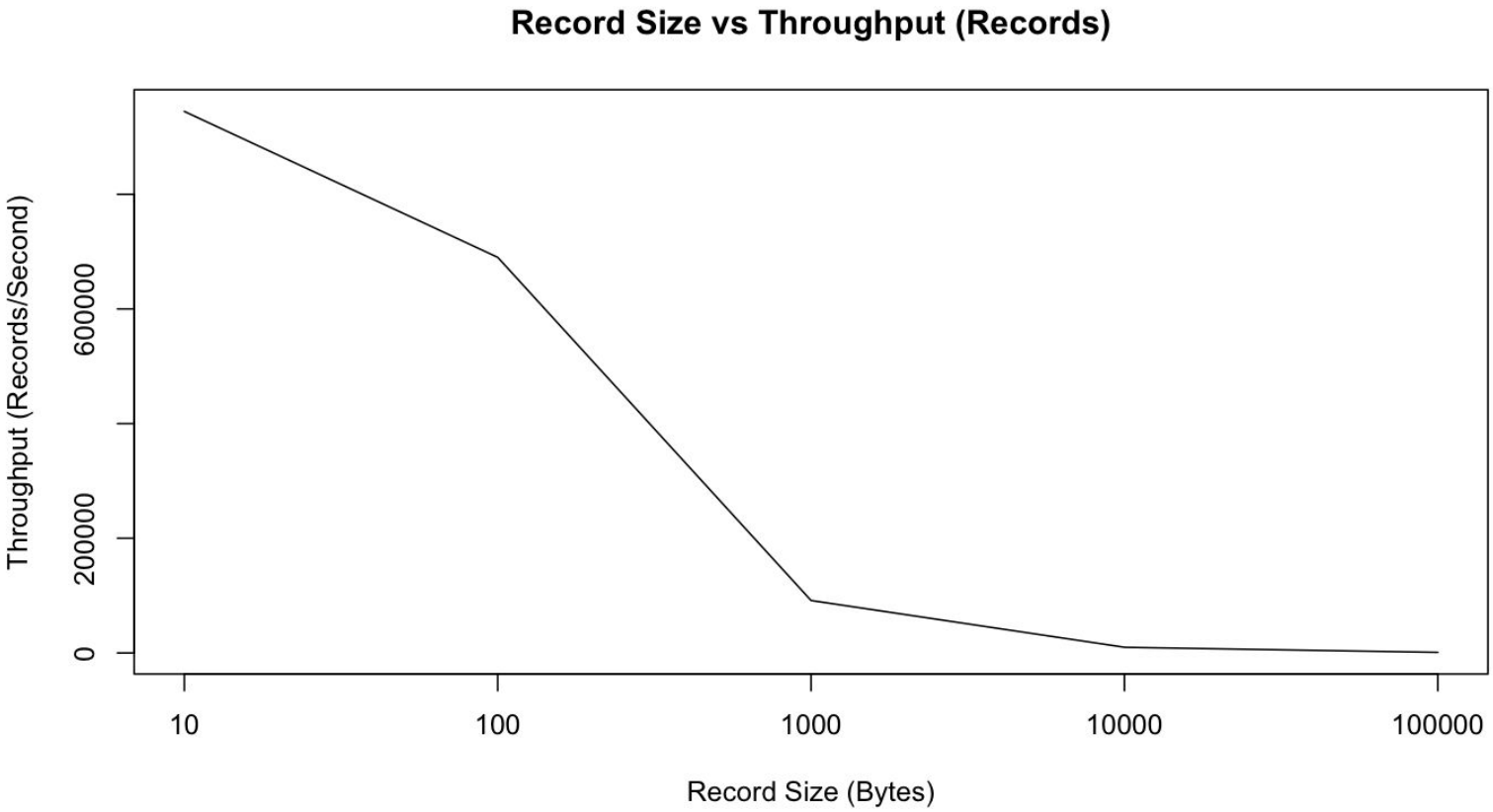
- Messaging
- Web site tracking activity
- Metrics
- Stream Processing
- Event Sourcing
- Commit Log



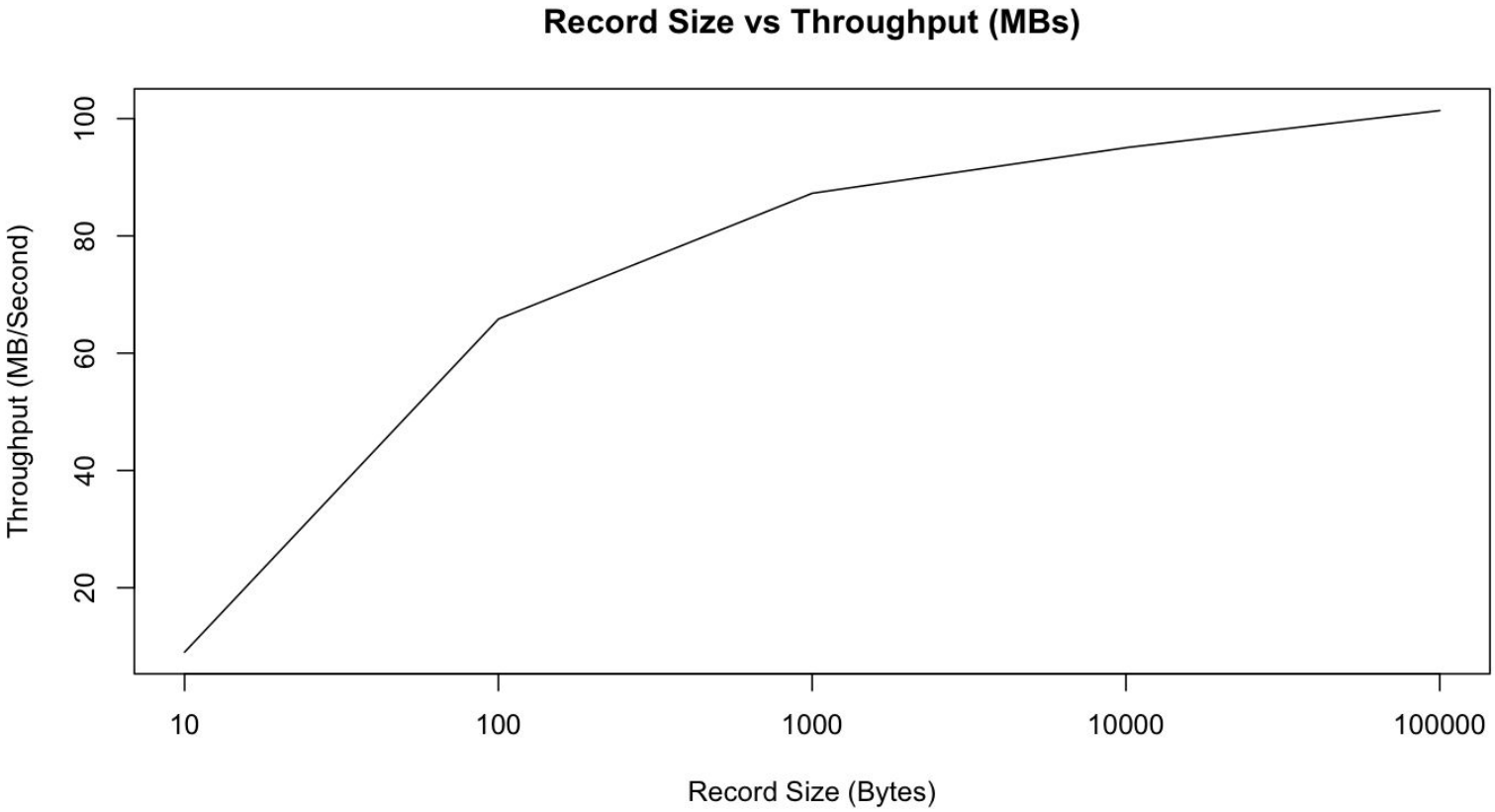
Performance

- 2M writes per second
  - Intel Xeon 2.5 GHz processor with six cores
  - Six 7200 RPM SATA drives
  - 32GB of RAM
  - 1Gb Ethernet
- Single producer thread, no replication
  - 821,557 records/sec
  - (78.3 MB/sec)
- End-to-end Latency
  - 2 ms (median)
  - 3 ms (99th percentile)
  - 14 ms (99.9th percentile)

Performance



Performance

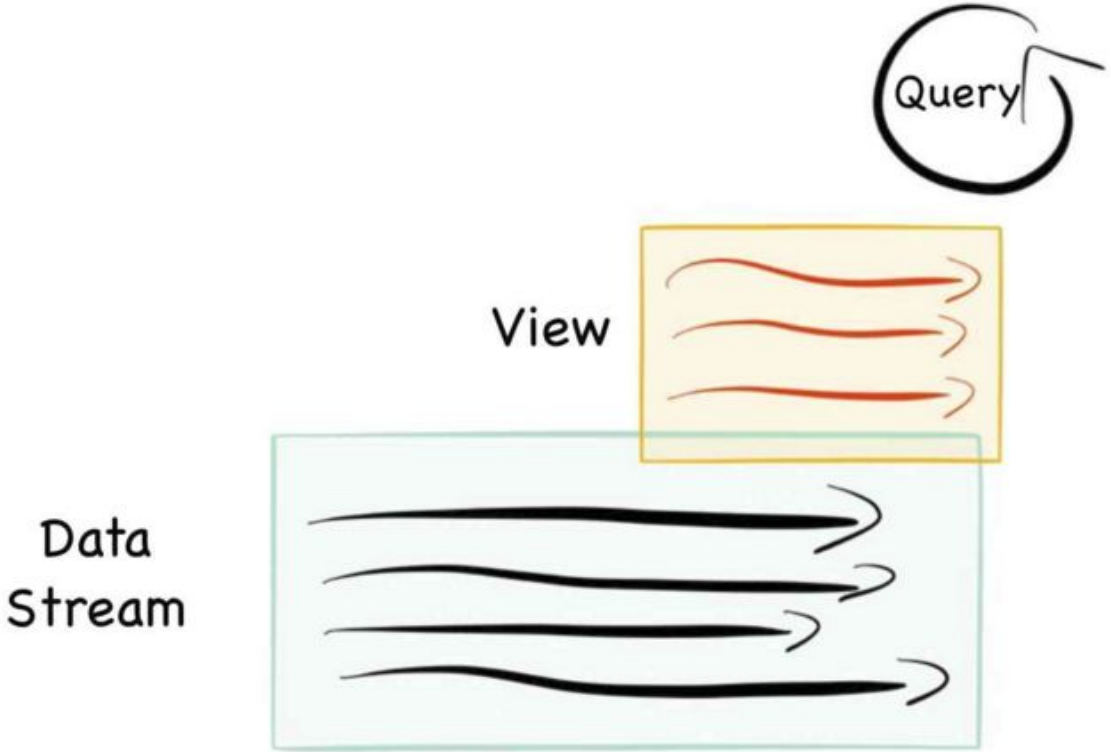


The truth is the log.

The database is a cache of a subset of the log.

—Pat Helland
Immutability Changes Everything
http://cidrdb.org/cidr2015/Papers/CIDR15_Paper16.pdf

Talk our own data model



Kafka: Numbers at LinkedIN 2019/Q1

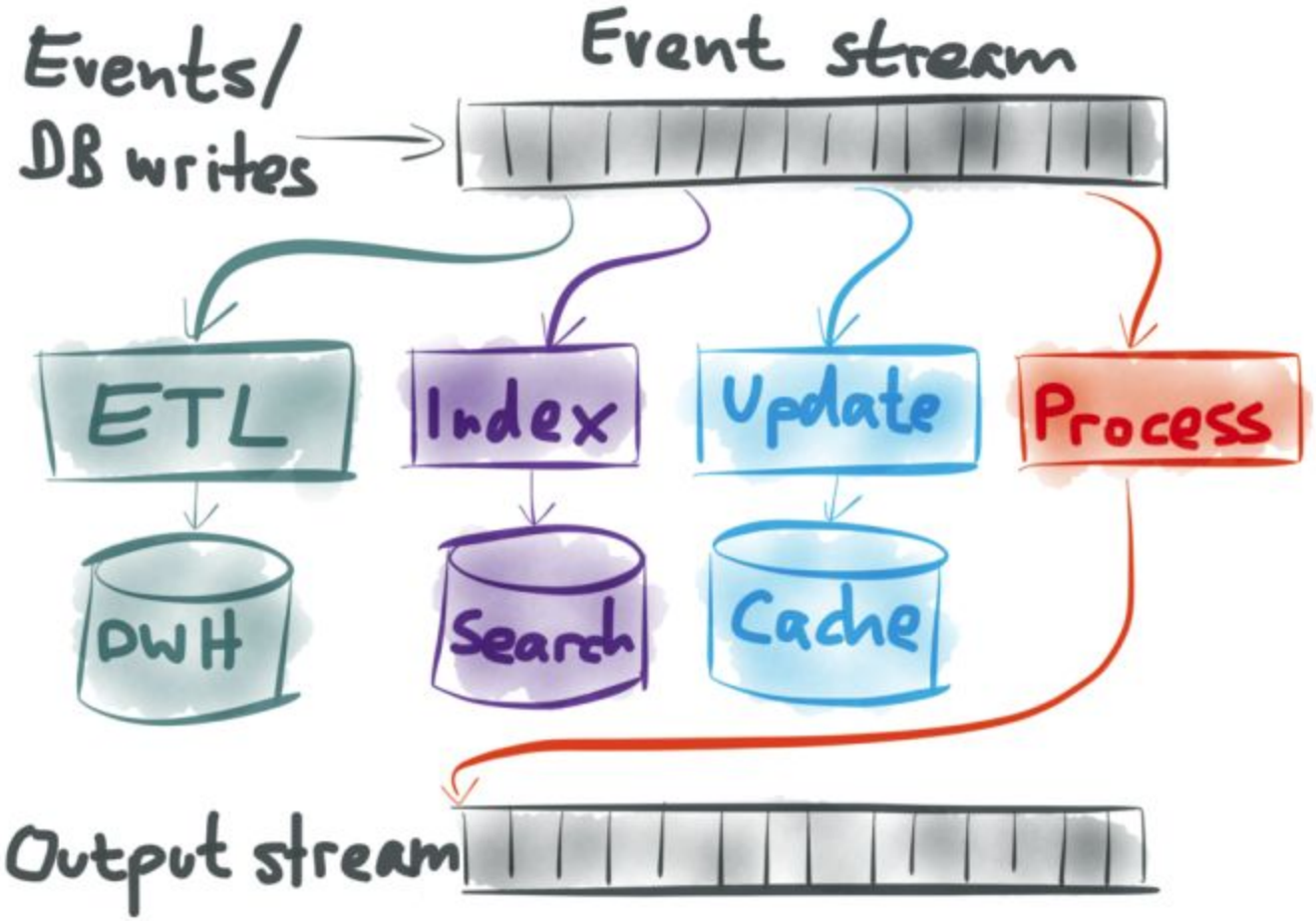
Big Scale to Operate

2T
messages/day

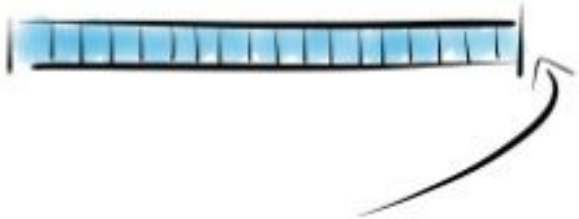
40+
Kafka src clusters
in different DCs

100+
pipelines

Kafka: As ES Solution

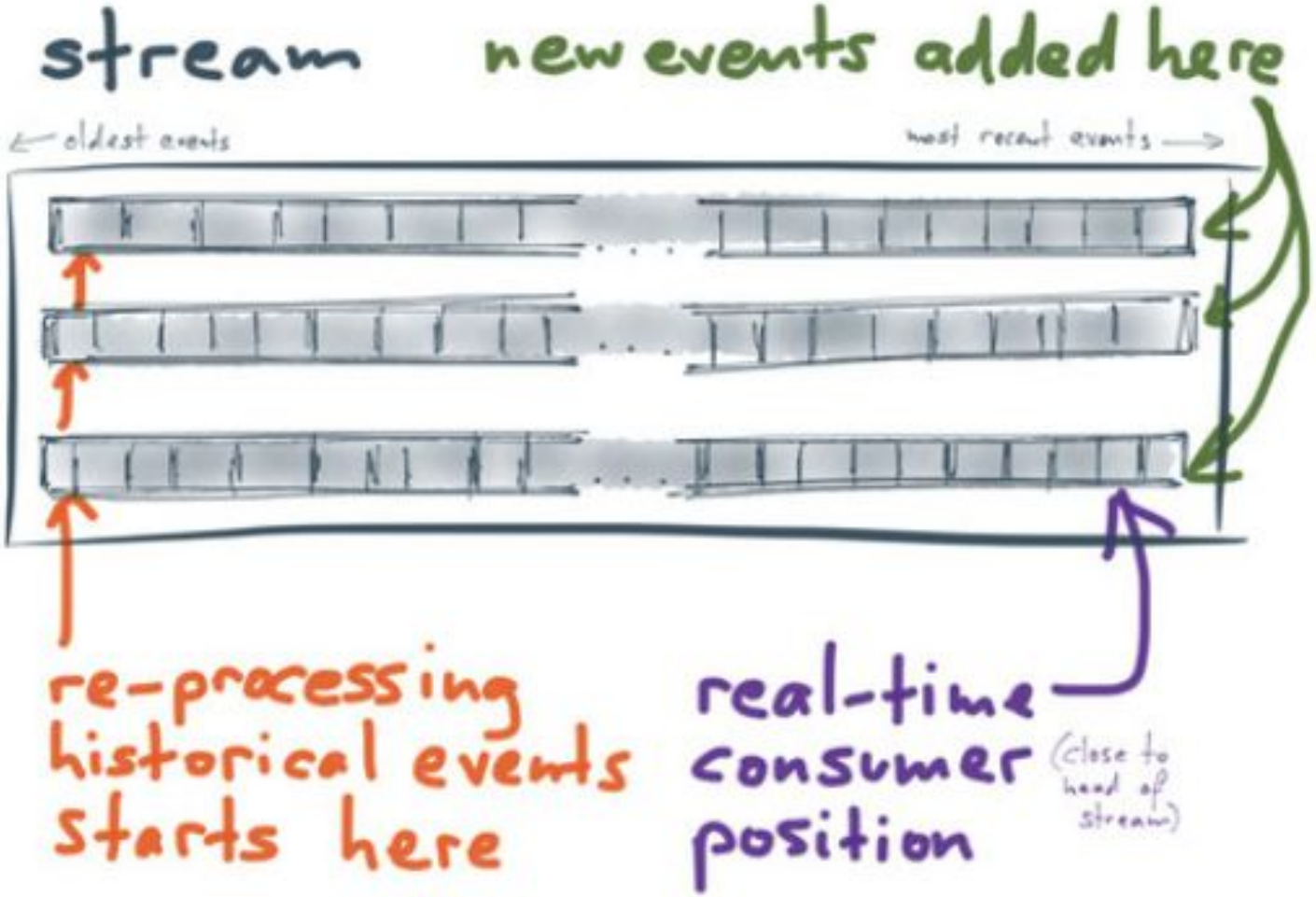


KAFKA's Distributed Log

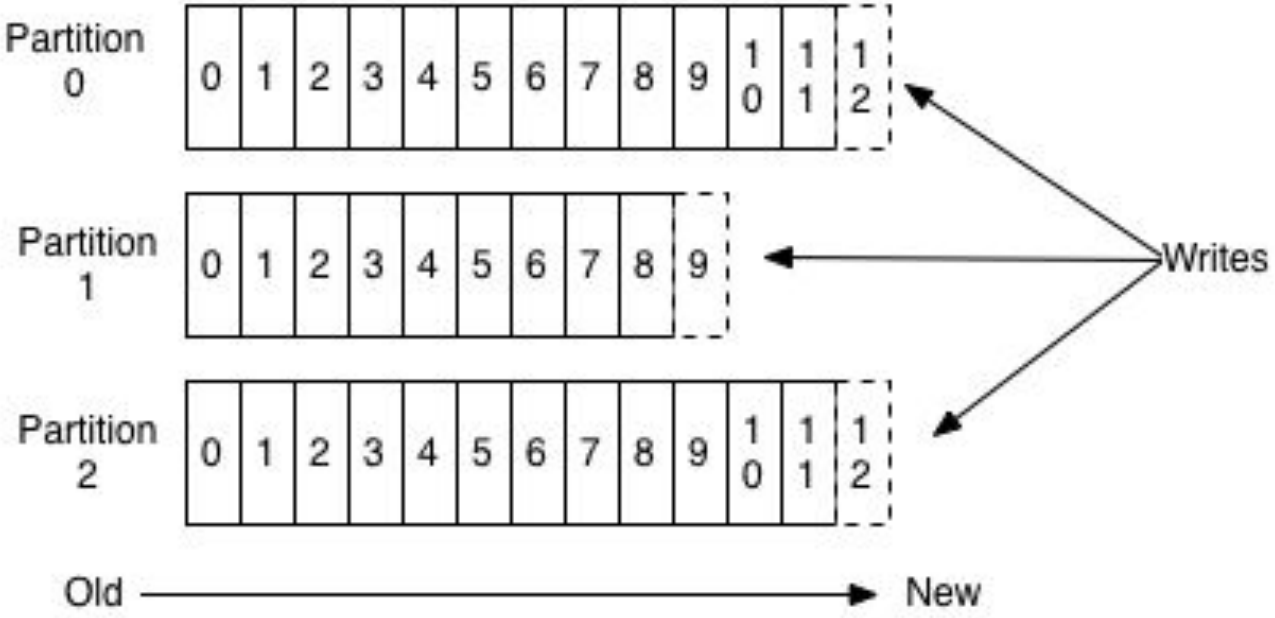


Append Only

Linear Scans

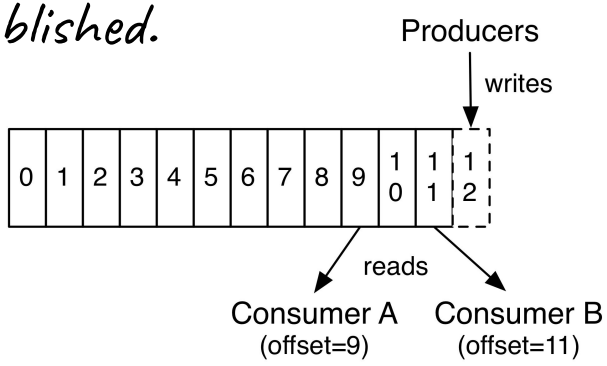

stream
new events added here
← oldest events
most recent events →
re-processing historical events starts here
real-time consumer position
(close to head of stream)

Anatomy of a Topic



Topics & Partitions

- Topic = Category or Field Name where records are published.
- Zero, 1 or Multiple-subscribers.
- Each topic has a partitioned log (partitions).
- Each Partition:
  - Ordered
  - Immutable
  - Continually appended
  - Unique Sequential ID called *offset*
- Kafka cluster durable persit all logs (no matter if consumed or not)



Topics & Partitions

- Kafka persist published records for period of time(configured retention)
- Kafka cluster performance is equivalent constant of data size.
- Storing data for a long time is not a problem.
- Consumer need to keep track of offset.
- Consumer can consume records in any order.
- Partition on log enable scaling beyond a single server size.
- A Topic may have any partitions so it handle arbitrary size of data.
- Partition acts as unit of parallelism.

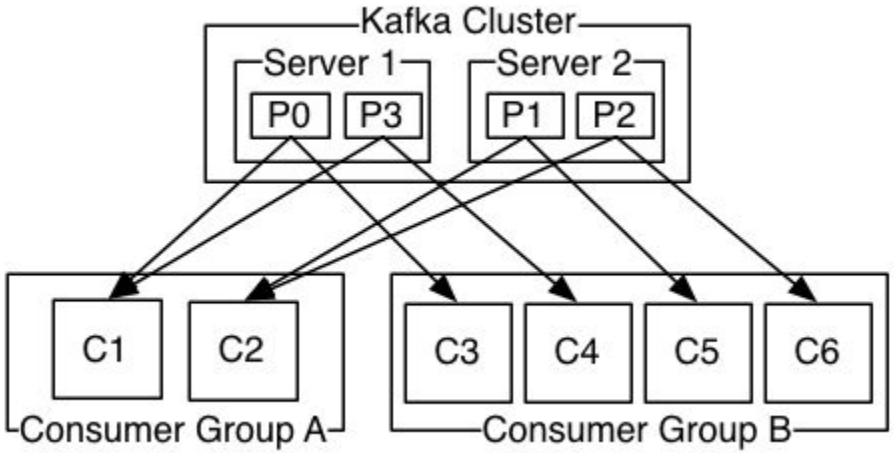
Topics & Partitions: Distribution,Failover, Replication

- Partitions are distributed of several server at kafka cluster
- Each partition is replicated across configurable N number of servers.
- Each partition has at least 1 server "leader" and 0..N "followers".
- IF Leader fails, one of the followers become the new leader automatically.
- Kafka Mirror maker provides geo-replication for kafka clusters.
- Mirror maker: replication across clusters, data centers, regions.
- Can be used as Active/Passive for backup or active/active for data locality.

Producers & Consumers

- Producers can publish data in any topic.
- Producer choose each record goes to each partition.
- It can be done via Round-robin for LB.
- It can be done via semantic partitioning function (key based).
- Consumers are labeled in a "Consumer Group".
- Consumer instances can be separated process/machines.
- Load-balancer: happens IF all consumer instances have the same consumer group.
- Broadcast: happens if all consumers has different instances.

Producers & Consumers: Publish Subscriber Semantics



- ❑ <u>Load-Balancer</u>: Same Consumer group
- ❑ <u>Broadcast</u>: Different Consumer Group

Producers & Consumers

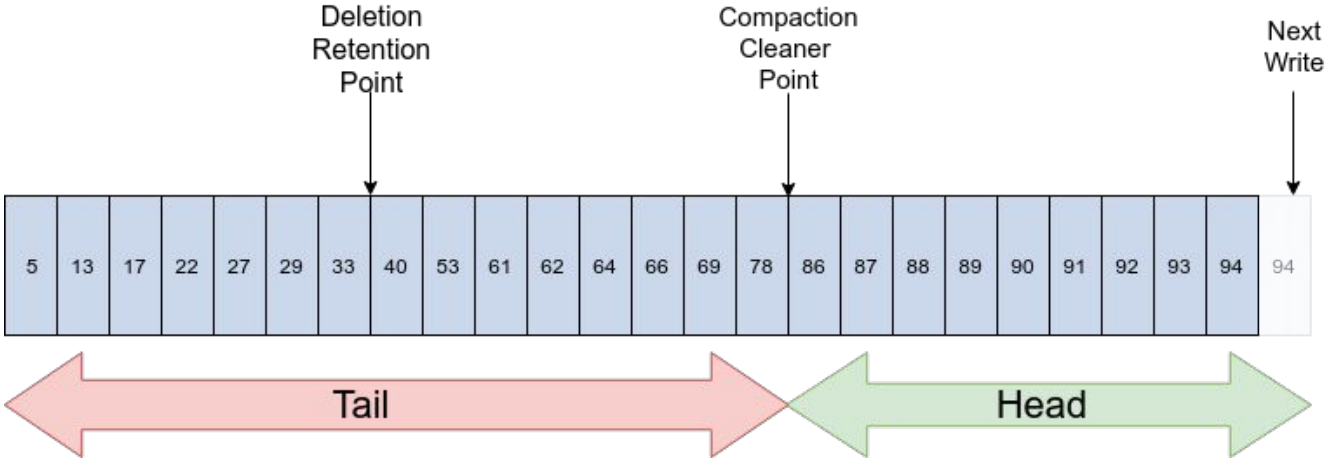
- Kafka only provides a total order within a partition.
- No total ordering is guaranteed between partitions.
- Total Order of records is required? It's Possible. So:
  - 1 single topic
  - 1 single partition
  - 1 single consumer process per consumer group
- Kafka can do multi-tenancy configuring which topics can produce or consume data, there are support for quotas.

Kafka Guarantees

- Messages sent by a producer to a topic/partition will be appended in order.
- Producer send: M1 them M2. M1 will have a lower offset than M2.
- Consumer will see the records in order as they are in the log.
- Topic with replication factor N will tolerate up to N-1 server failures without losing records committed to the log.
- Kafka can do Queue / Publisher Subscriber at Scale with multiple subscribers.
- Kafka has stronger guarantee than regular messaging solution(parallelism).
- Data is written in disk by kafka: replicated for fault tolerance.
- Kafka scales well with 50kb or 50TB of storage.

Kafka Guarantees

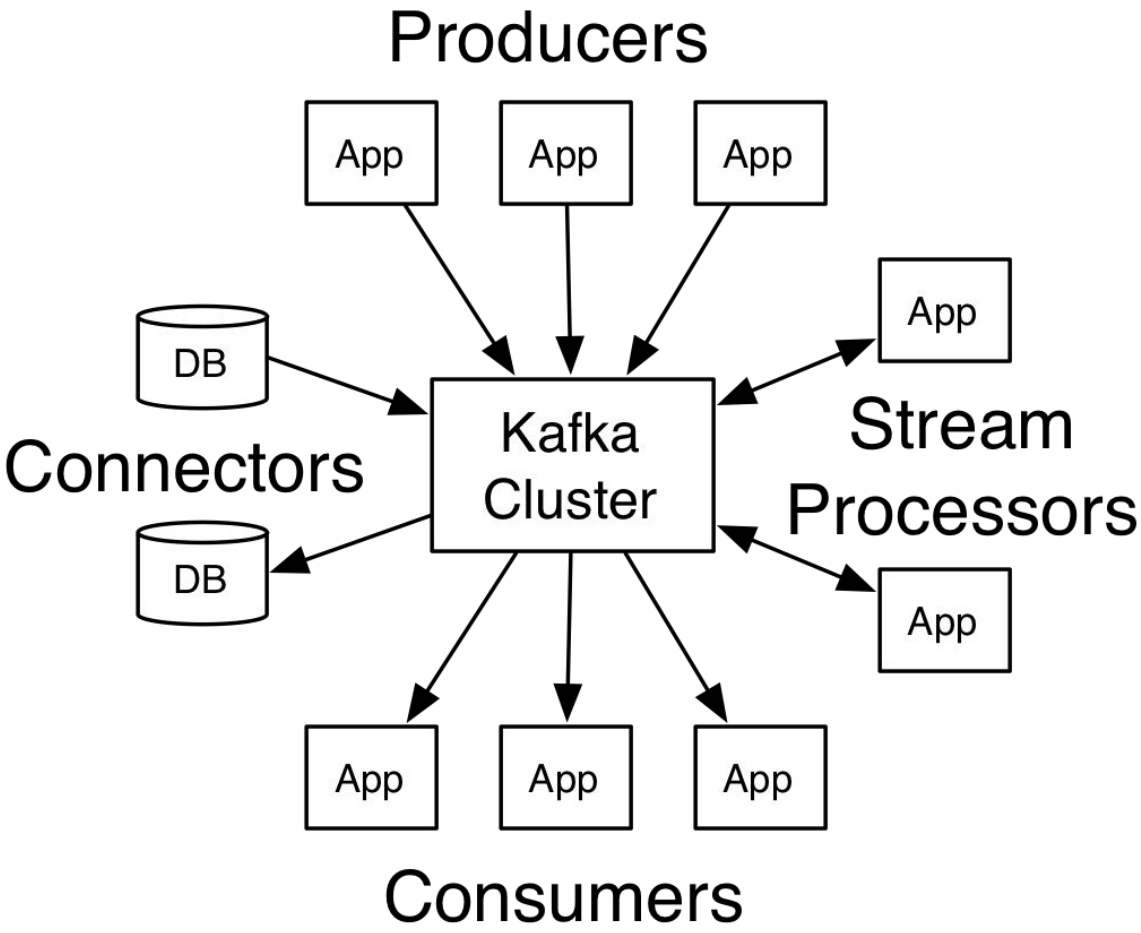
- You can consider kafka as a kind of distributed file system(Commit Log)
  - High performance
  - Low-latency
  - Replication & Propagation



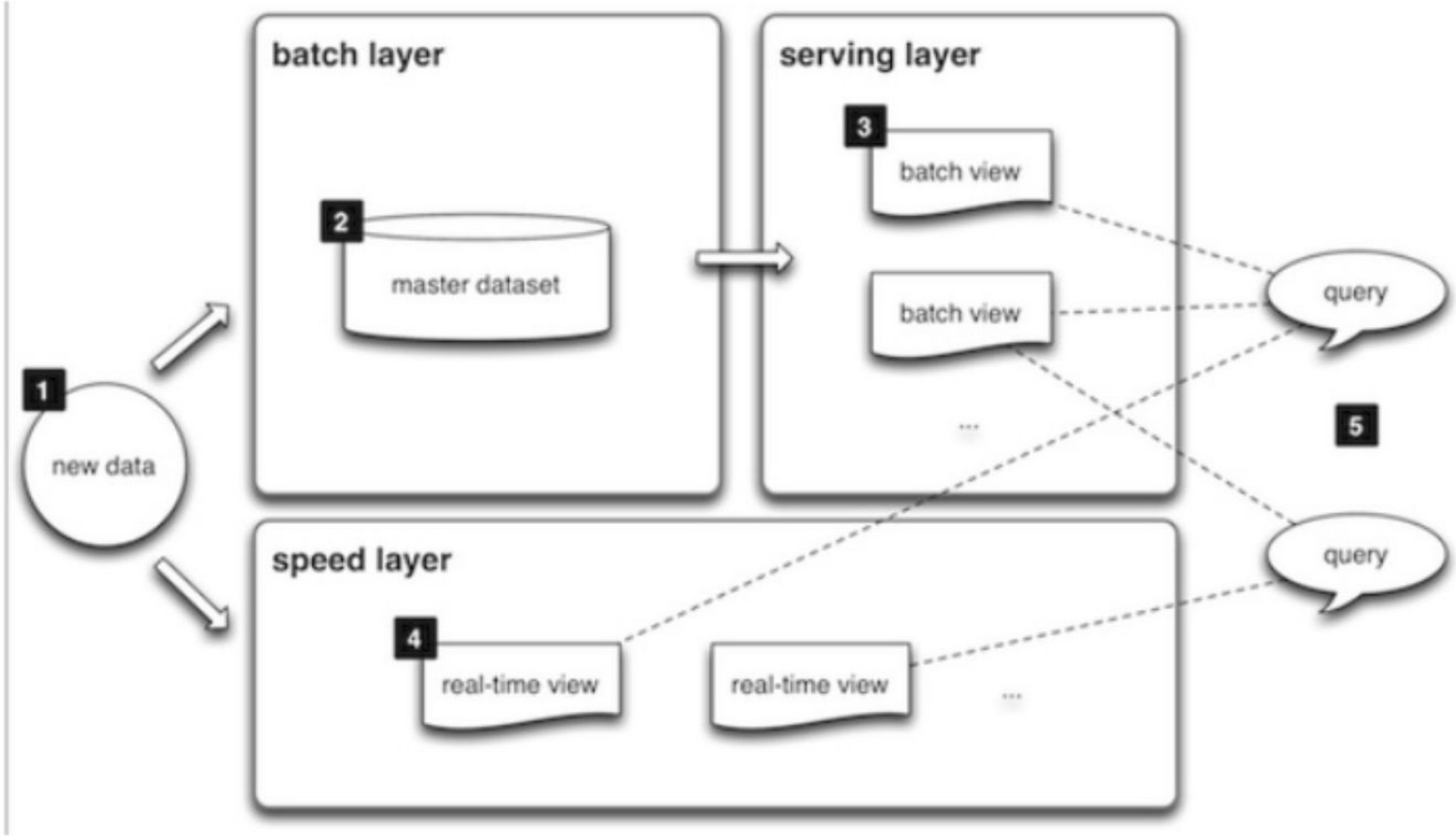
Delivery Semantics

- <u>At most once</u>—Messages may be lost but are never redelivered. (LOSS)
- <u>At least once</u>—Messages are never lost but may be redelivered. (DEPLICATED)
- <u>Exactly once</u>—this is what people actually want, each message is delivered once and only once. (Harder)

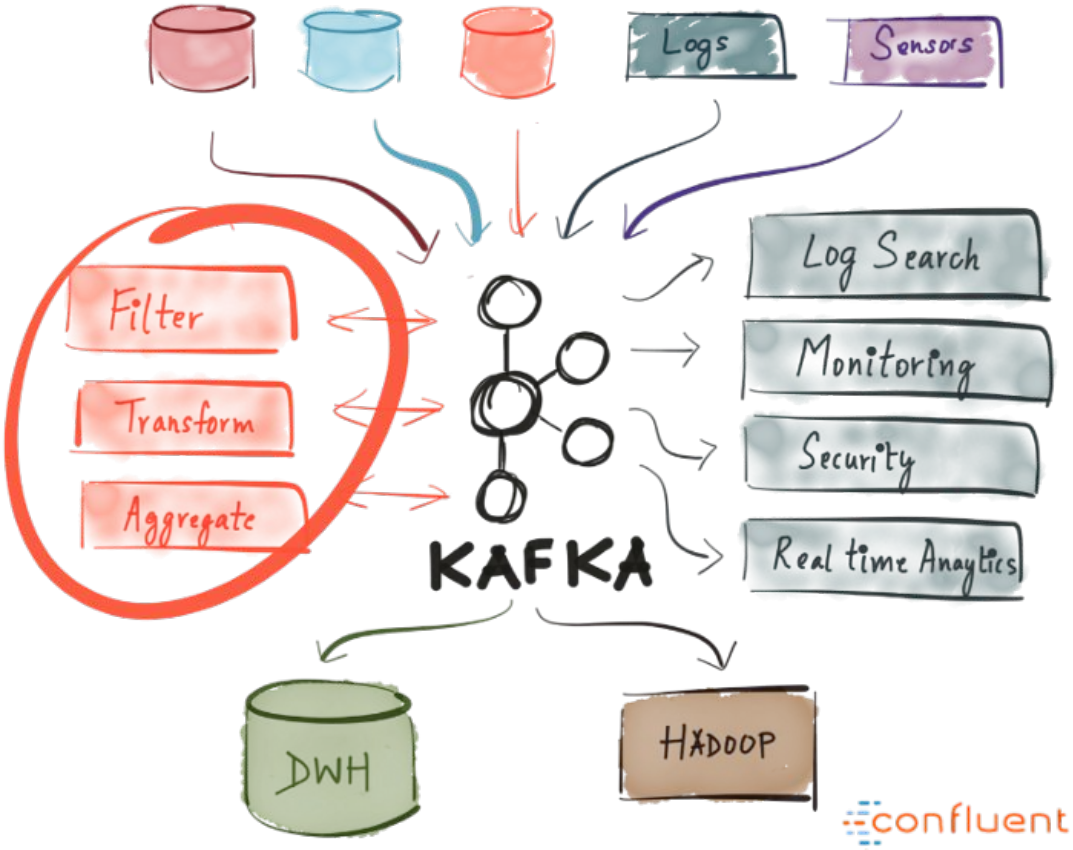
Kafka


Producers
App
App
App
App
DB
Connectors
Kafka
Cluster
Stream
Processors
DB
App
App
App
App
Consumers

Lambda / Kappa Architecture



Kafka: Other use Cases



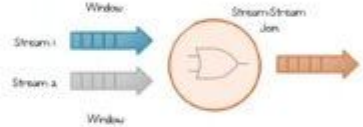
Kafka: What Platform can do!

Combine them:

1. Stateless

2. Data enriched

3. Gates

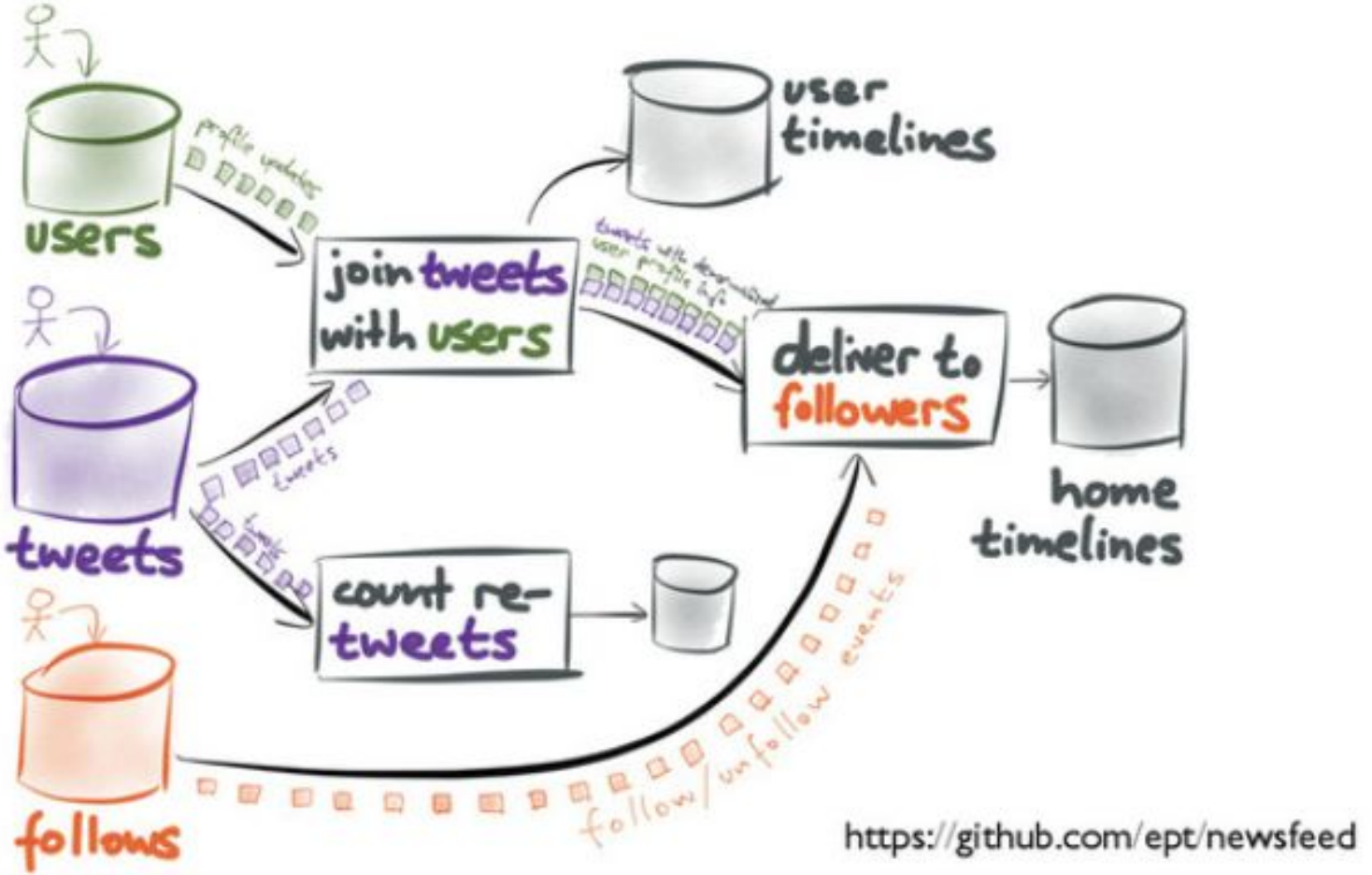


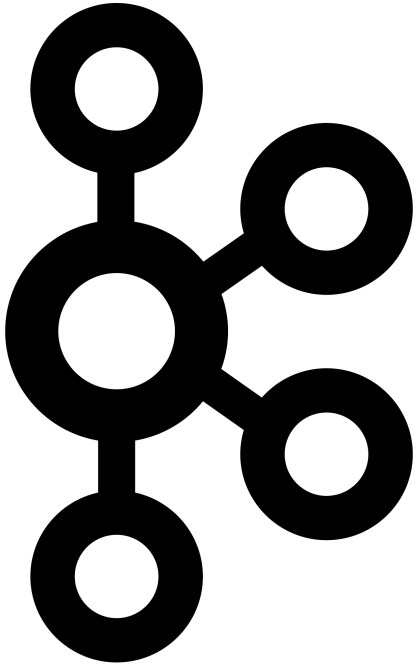
4. Stateful processors

5. Stream-aside

6. Sidecar

Streaming:: Brave new world!





Running Kafka with Docker & Kafka API Using Java / Spring

File: **kafka-docker-compose.yaml**

```yaml
version: '2.1'

services:
  zoo1:
    image: zookeeper:3.4.9
    hostname: zoo1
    ports:
      - "2181:2181"
    environment:
        ZOO_MY_ID: 1
        ZOO_PORT: 2181
        ZOO_SERVERS: server.1=zoo1:2888:3888
    volumes:
      - ./zk-single-kafka-single/zoo1/data:/data
      - ./zk-single-kafka-single/zoo1/datalog:/datalog

  kafka1:
    image: confluentinc/cp-kafka:5.3.0
    hostname: kafka1
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka1:19092,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 1
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    volumes:
      - ./zk-single-kafka-single/kafka1/data:/var/lib/kafka/data
    depends_on:
      - zoo1
```

$ docker-compose up

docker-compose up

File Edit View Search Terminal Help

```
kafka1_1  |     send.buffer.bytes = 131072
kafka1_1  |     ssl.cipher.suites = null
kafka1_1  |     ssl.enabled.protocols = [TLSv1.2, TLSv1.1, TLSv1]
kafka1_1  |     ssl.endpoint.identification.algorithm = https
kafka1_1  |     ssl.key.password = null
kafka1_1  |     ssl.keymanager.algorithm = SunX509
kafka1_1  |     ssl.keystore.location = null
kafka1_1  |     ssl.keystore.password = null
kafka1_1  |     ssl.keystore.type = JKS
kafka1_1  |     ssl.protocol = TLS
kafka1_1  |     ssl.provider = null
kafka1_1  |     ssl.secure.random.implementation = null
kafka1_1  |     ssl.trustmanager.algorithm = PKIX
kafka1_1  |     ssl.truststore.location = null
kafka1_1  |     ssl.truststore.password = null
kafka1_1  |     ssl.truststore.type = JKS
kafka1_1  |     transaction.timeout.ms = 60000
kafka1_1  |     transactional.id = null
kafka1_1  |     value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer
kafka1_1  |  (org.apache.kafka.clients.producer.ProducerConfig)
kafka1_1  | [2019-11-16 01:01:53,224] INFO [Producer clientId=producer-1] Closing the Kafka producer with timeoutMillis = 0 ms. (org.apache.kafka.clients.producer.KafkaProducer)
kafka1_1  | [2019-11-16 01:01:53,224] ERROR Could not submit metrics to Kafka topic __confluent.support.metrics: Failed to construct kafka producer (io.confluent.support.metrics.BaseMetricsReporter)
kafka1_1  | [2019-11-16 01:01:53,319] INFO [Log partition=__confluent.support.metrics-0, dir=/var/lib/kafka/data] Loading producer state till offset 0 with message format version 2 (kafka.log.Log)
kafka1_1  | [2019-11-16 01:01:53,332] INFO [Log partition=__confluent.support.metrics-0, dir=/var/lib/kafka/data] Completed load of log with 1 segments, log start offset 0 and log end offset 0 in 82 ms (kafka.log.Log)
kafka1_1  | [2019-11-16 01:01:53,336] INFO Created log for partition __confluent.support.metrics-0 in /var/lib/kafka/data with properties {compression.type -> producer, message.downconversion.enable -> true, min.insync.replicas -> 1, segment.jitter.ms -> 0, cleanup.policy -> [delete], flush.ms -> 9223372036854775807, segment.bytes -> 1073741824, retention.ms -> 31536000000, flush.messages -> 9223372036854775807, message.format.version -> 2.3-IV1, file.delete.delay.ms -> 60000, max.compaction.lag.ms -> 9223372036854775807, max.message.bytes -> 1000012, min.compaction.lag.ms -> 0, message.timestamp.type -> CreateTime, preallocate -> false, min.cleanable.dirty.ratio -> 0.5, index.interval.bytes -> 4096, unclean.leader.election.enable -> false, retention.bytes -> -1, delete.retention.ms -> 86400000, segment.ms -> 604800000, message.timestamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760}. (kafka.log.LogManager)
kafka1_1  | [2019-11-16 01:01:53,337] INFO [Partition __confluent.support.metrics-0 broker=1] No checkpointed highwatermark is found for partition __confluent.support.metrics-0 (kafka.cluster.Partition)
kafka1_1  | [2019-11-16 01:01:53,339] INFO Replica loaded for partition __confluent.support.metrics-0 with initial high watermark 0 (kafka.cluster.Replica)
kafka1_1  | [2019-11-16 01:01:53,344] INFO [Partition __confluent.support.metrics-0 broker=1] __confluent.support.metrics-0 starts at Leader Epoch 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)
kafka1_1  | [2019-11-16 01:01:54,353] INFO Successfully submitted metrics to Confluent via secure endpoint (io.confluent.support.metrics.submitters.ConfluentSubmitter)
kafka1_1  | [2019-11-16 01:01:57,726] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
```

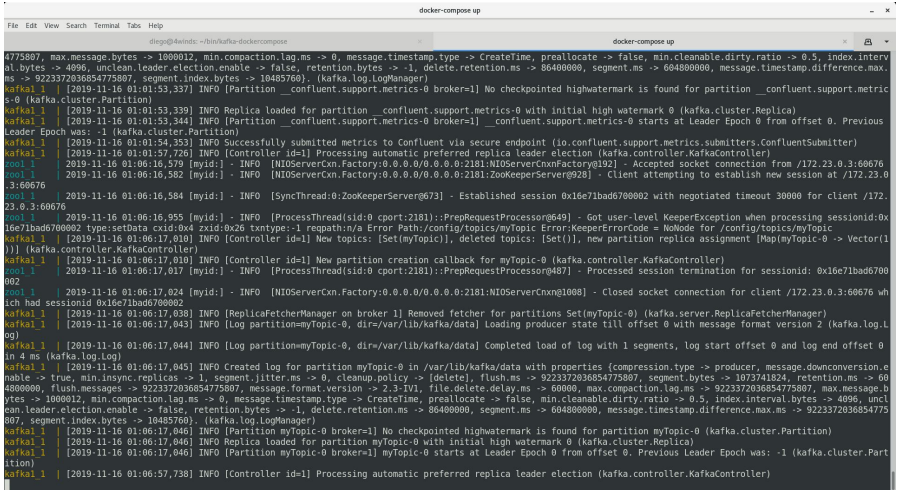
Creating Topics & Sending Messages

diego@4winds: ~/bin/kafka-dockercompose

File Edit View Search Terminal Tabs Help

diego@4winds: ~/bin/kafka-dockercompose | docker-compose up

```
diego@4winds  ~/bin/kafka-dockercompose  docker-compose exec kafka1  \                                  17:06:09  6.64G  1.00
kafka-topics --create --topic myTopic --partitions 1 --replication-factor 1 --if-not-exists --zookeeper zoo1:2181
Created topic myTopic.
diego@4winds  ~/bin/kafka-dockercompose  docker-compose exec kafka1  \                                  17:06:17  6.65G  1.03
  bash -c "seq 10 | kafka-console-producer --request-required-acks 1 --broker-list localhost:9092 --topic myTopic && echo 'Produced 10 messages.'"
>>>>>>>>>>>Produced 10 messages.
diego@4winds  ~/bin/kafka-dockercompose                                                                  17:06:43  6.64G  0.99
```



Java & Kafka - Using Spring Kafka

```
diego@4winds  ~/bin/kafka-dockercompose  bat build.gradle                                  17:09:07  6.61G  1.49
File: build.gradle

plugins {
    id 'java'
    id 'application'
         id 'org.springframework.boot' version '2.1.8.RELEASE'
         id 'io.spring.dependency-management' version '1.0.8.RELEASE'
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

mainClassName = 'kafka.SpringKafkaSpringApp'

repositories {
    mavenCentral()
    maven { url 'https://oss.sonatype.org/content/groups/public/' }
}

dependencies {
       implementation 'org.springframework.boot:spring-boot-starter'
       implementation 'org.springframework.kafka:spring-kafka'

  testCompile([
      'junit:junit:4.12'
  ])
}

run {
  systemProperties System.getProperties()
}
```

Java & Kafka - Using Spring Kafka

src/main/resources/application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id=myGroup
```

File: **SpringKafkaApp.java**

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.core.KafkaTemplate;

@SpringBootApplication
public class SpringKafkaApp implements CommandLineRunner {

  private final KafkaTemplate<String, String> kafkaTemplate;

  @Autowired
  public SpringKafkaSpringApp(KafkaTemplate<String, String> kafkaTemplate) {
    this.kafkaTemplate = kafkaTemplate;
  }

  @KafkaListener(topics = "myTopic")
  public void processMessage(String content) {
    System.out.println(content);
  }

  @Override
  public void run(String... args) {
    System.out.println("Spring app Running... " + kafkaTemplate);
    kafkaTemplate.send("myTopic", "hey how is it going? ");
  }

  public static void main(String[] args) {
    SpringApplication.run(SpringKafkaSpringApp.class, args);
  }

}
```

Java & Kafka - Using Spring Kafka: $ ./gradlew bootRun

./gradlew bootRun

File Edit View Search Terminal Tabs Help

./gradlew bootRun | docker-compose up | diego@4winds: ~/github/diegopacheco/java-pocs/pocs/spring-kafka-fun

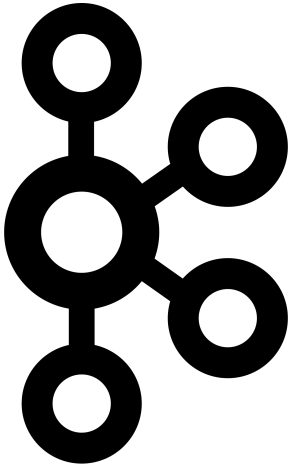
```
        ssl.keystore.password = null
        ssl.keystore.type = JKS
        ssl.protocol = TLS
        ssl.provider = null
        ssl.secure.random.implementation = null
        ssl.trustmanager.algorithm = PKIX
        ssl.truststore.location = null
        ssl.truststore.password = null
        ssl.truststore.type = JKS
        transaction.timeout.ms = 60000
        transactional.id = null
        value.serializer = class org.apache.kafka.common.serialization.StringSerializer

2019-11-15 17:29:30.768  INFO 19428 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : [Consumer clientId=consumer-2, groupId=myGroup] Discovered group coordin
ator 127.0.0.1:9092 (id: 2147483646 rack: null)
2019-11-15 17:29:30.789  INFO 19428 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator  : [Consumer clientId=consumer-2, groupId=myGroup] Revoking previously assi
gned partitions []
2019-11-15 17:29:30.794  INFO 19428 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer    : partitions revoked: []
2019-11-15 17:29:30.795  INFO 19428 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : [Consumer clientId=consumer-2, groupId=myGroup] (Re-)joining group
2019-11-15 17:29:30.832  INFO 19428 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka version : 2.0.1
2019-11-15 17:29:30.833  INFO 19428 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId : fa14705e51bd2ce5
2019-11-15 17:29:30.862  INFO 19428 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : Cluster ID: Kiv-UNglRGSfFa3Cjw0ROg
2019-11-15 17:29:33.916  INFO 19428 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : [Consumer clientId=consumer-2, groupId=myGroup] Successfully joined grou
p with generation 3
2019-11-15 17:29:33.920  INFO 19428 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator  : [Consumer clientId=consumer-2, groupId=myGroup] Setting newly assigned p
artitions [myTopic-0]
2019-11-15 17:29:33.940  INFO 19428 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer    : partitions assigned: [myTopic-0]
hey how is it going?
1
2
3
4
5
6
7
8
9
10
<=========----> 75% EXECUTING [30s]
> :bootRun
```

Exercises

You can do this exercises with Java or Scala.
You can use additional frameworks / libs to access Kafka.

1. <u>Build a ES Calculator system</u> which does basic math operations like (+,-,*,/) and for each operation expose a RAW(JSON) event in kafka of the calculation.
2. <u>Build a E-Commerce system</u> which does basic operations like book sales in one service, and sales report(top sales, top salesman) in another service. Code 1 service in java, the other in Scala. Use ES in kafka with RAW JSON event to make 2 services "talk".
3. <u>Build a Analytic system</u> Based on your previous exercise #2, project the amount on revenue for the next year (create a new service *revenue-forecast* read the sales data from Kafka. Produce a Revenue event in Kafka as well.



ES & Kafka

Diego Pacheco