Apache Kafka + Apache Mesos

Highly Scalable Streaming Microservices with Kafka Streams

Kai Waehner

Technology Evangelist

kontakt@kai-waehner.de

LinkedIn

@KaiWaehner

www.kai-waehner.de

Abstract

Microservices establish many benefits like agile, flexible development and deployment of business logic. However, a Microservice architecture also creates many new challenges like increased communication between distributed instances, the need for orchestration, new fail-over requirements, and resiliency design patterns.

This session discusses how to build a highly scalable, performant, mission-critical microservice infrastructure with Apache Kafka and Apache Mesos. Apache Kafka brokers are used as powerful, scalable, distributed message backbone. Kafka's Streams API allows to embed stream processing directly into any external microservice or business application; without the need for a dedicated streaming cluster. Apache Mesos can be used as scalable infrastructure for both, the Apache Kafka brokers and external applications using the Kafka Streams API, to leverage the benefits of a cloud native platforms like service discovery, health checks, or fail-over management.

A live demo shows how to develop real time applications for your core business with Kafka messaging brokers and Kafka Streams API and how to deploy / manage / scale them on a Mesos cluster using different deployment options.

Key takeaways for the audience
- Successful Microservice architectures require a highly scalable messaging infrastructure combined with a cloud-native platform which manages distributed microservices
- Apache Kafka offers a highly scalable, mission critical infrastructure for distributed messaging and integration
- Kafka's Streams API allows to embed stream processing into any external application or microservice
- Mesos allows management of both, Kafka brokers and external applications using Kafka Streams API, to leverage many built-in benefits like health checks, service discovery or fail-over control of microservices
- See a live demo which combines the Apache Kafka streaming platform and Apache Mesos

Agenda

1) Scalable Microservices
2) Apache Kafka and Confluent Platform
3) Kafka Streams
4) Scalable Microservices with Kafka and DC/OS
5) Live Demo

Agenda

1) Scalable Microservices

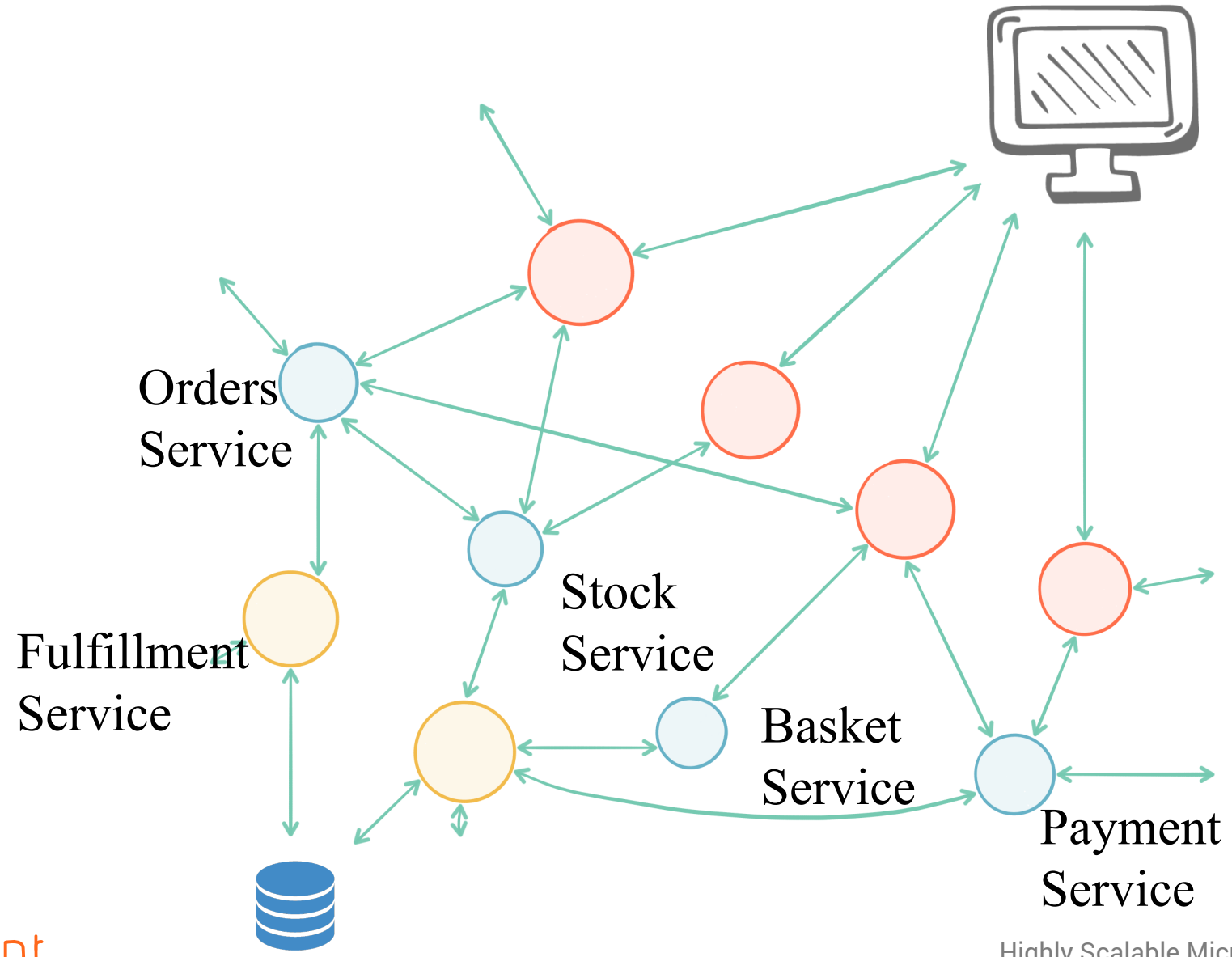
2) Apache Kafka and Confluent Platform

3) Kafka Streams

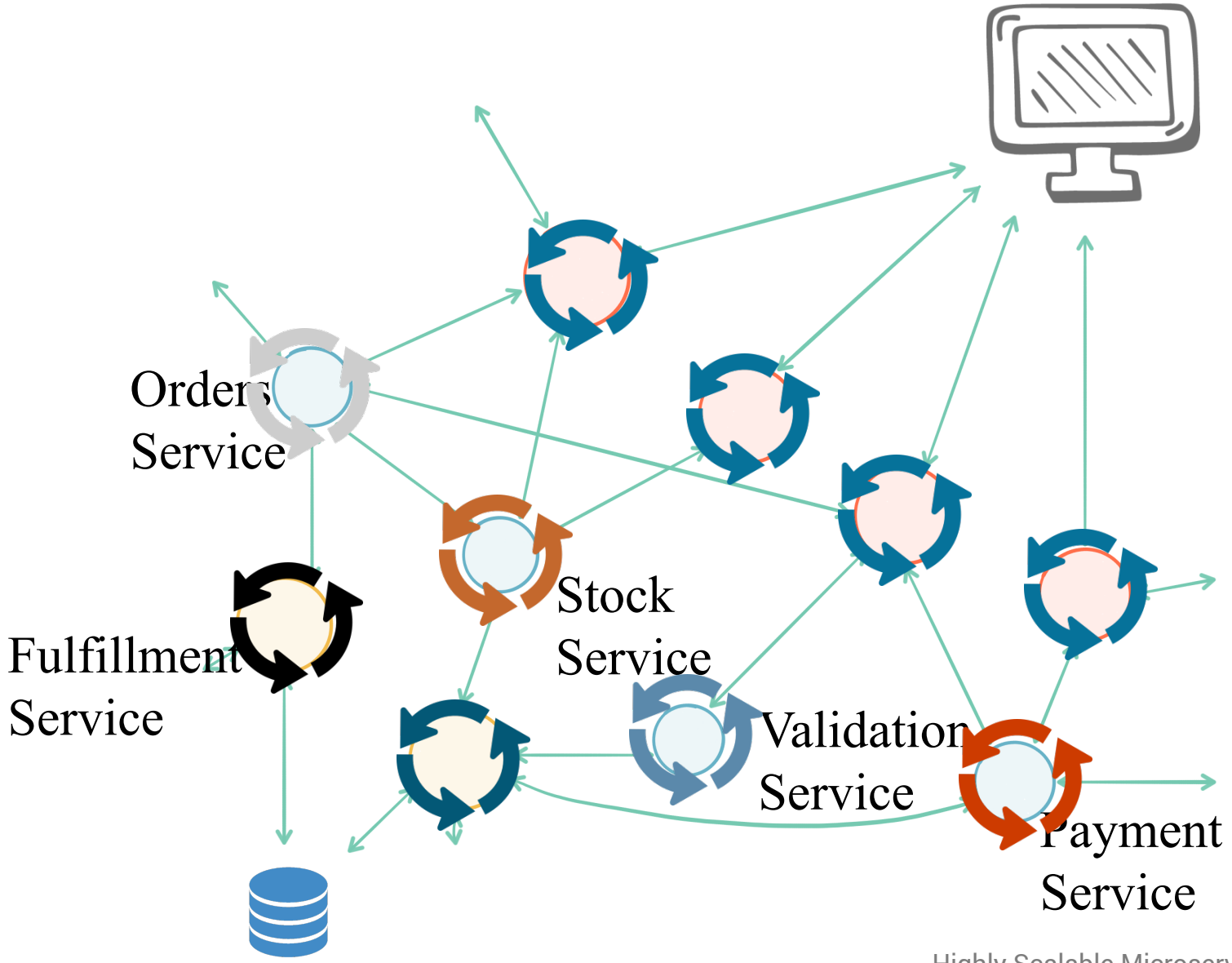
4) Scalable Microservices with Kafka and DC/OS

5) Live Demo

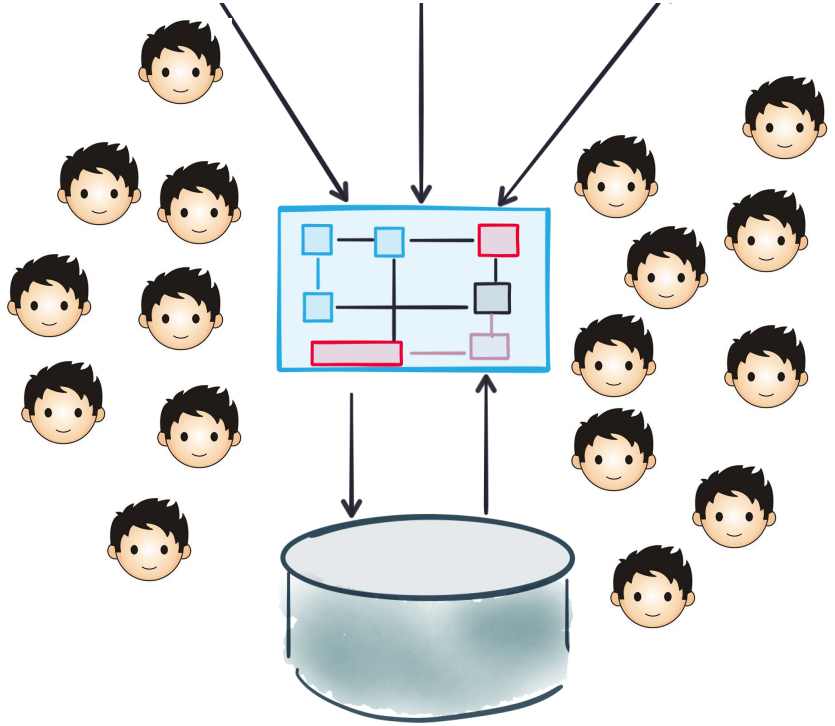
Microservices Architecture



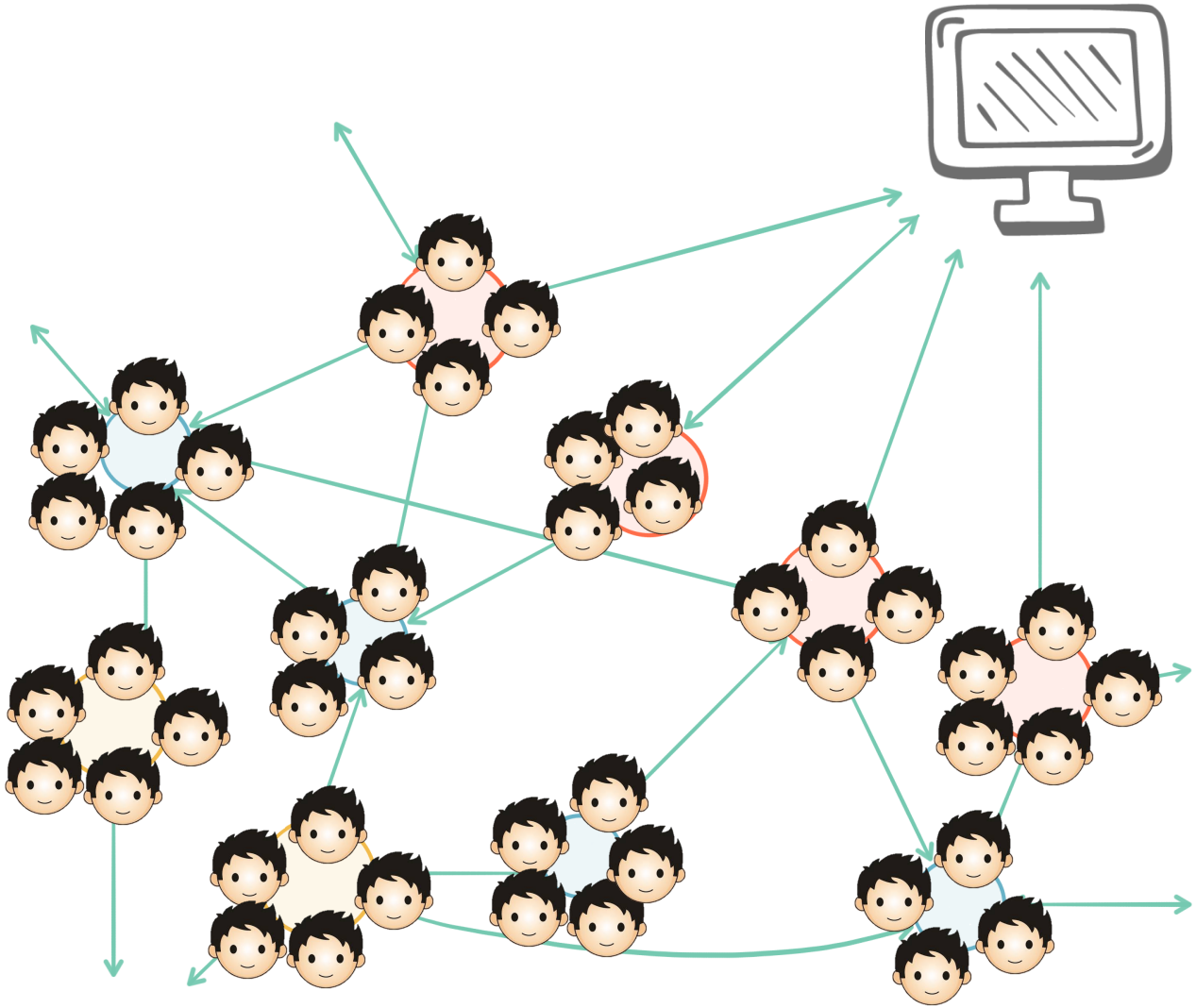
Independently Deployable



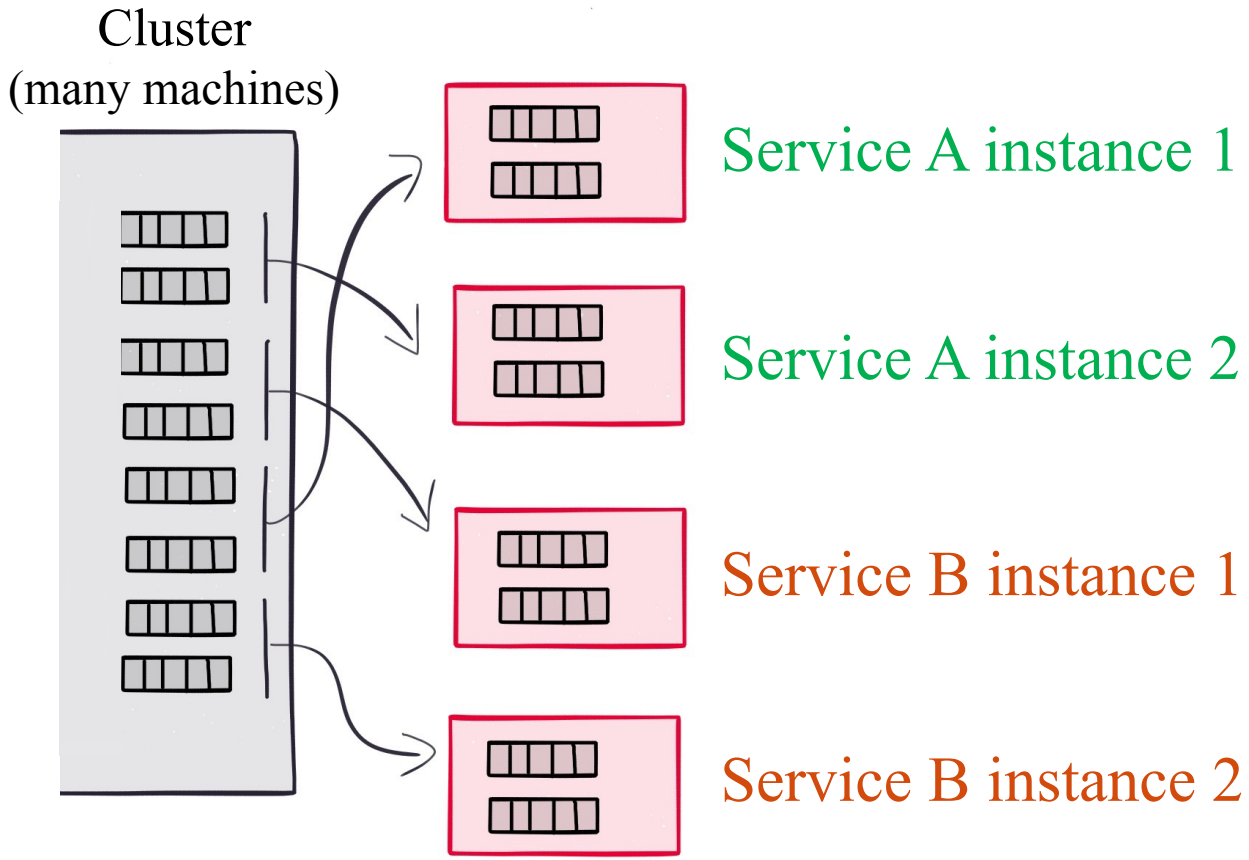
Allows us to scale



Scale in people terms



Scale in infrastructure terms



Scalable Microservices

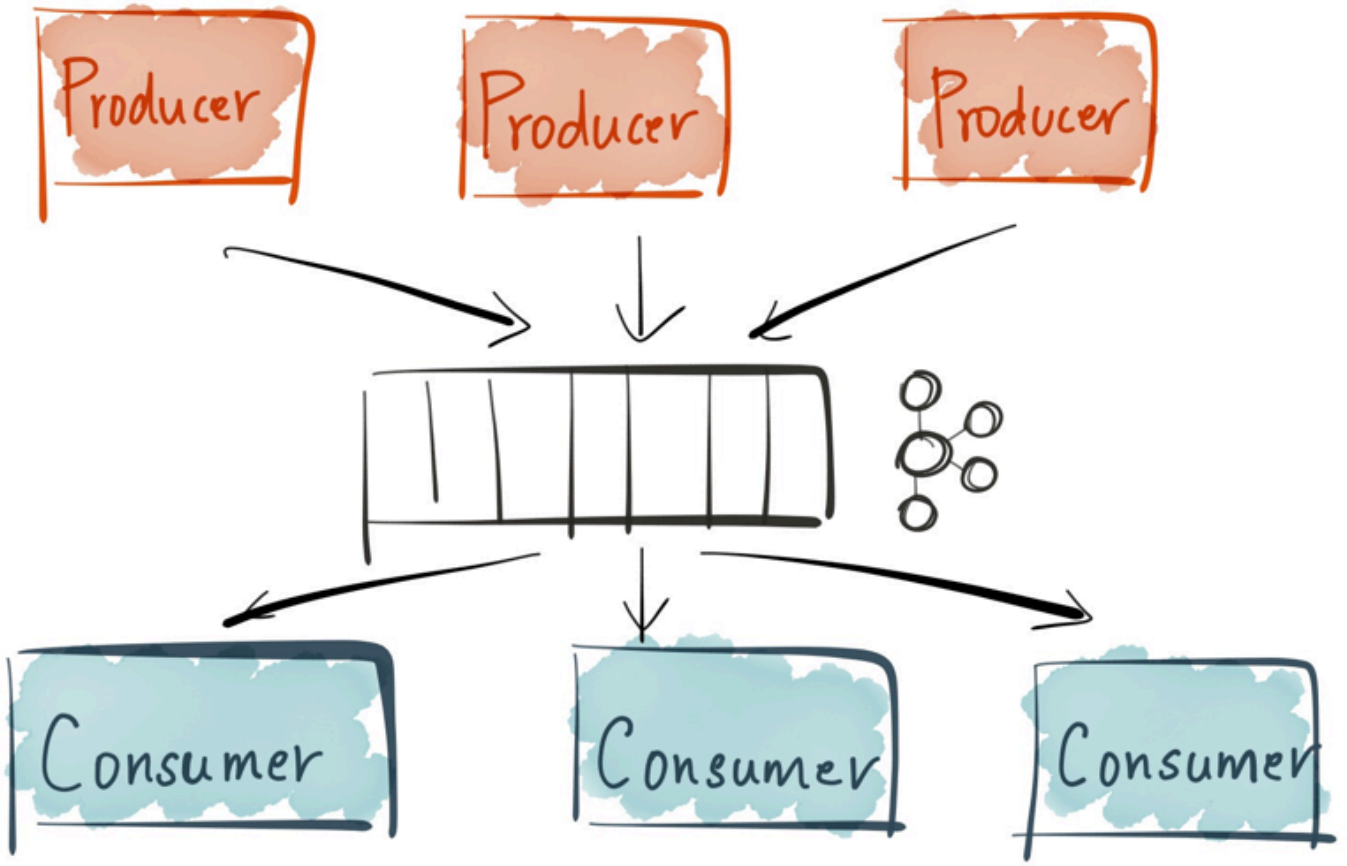
How do we get there?

- Loose Coupling
- Data Enabled
- Event Driven
- Operational Transparency

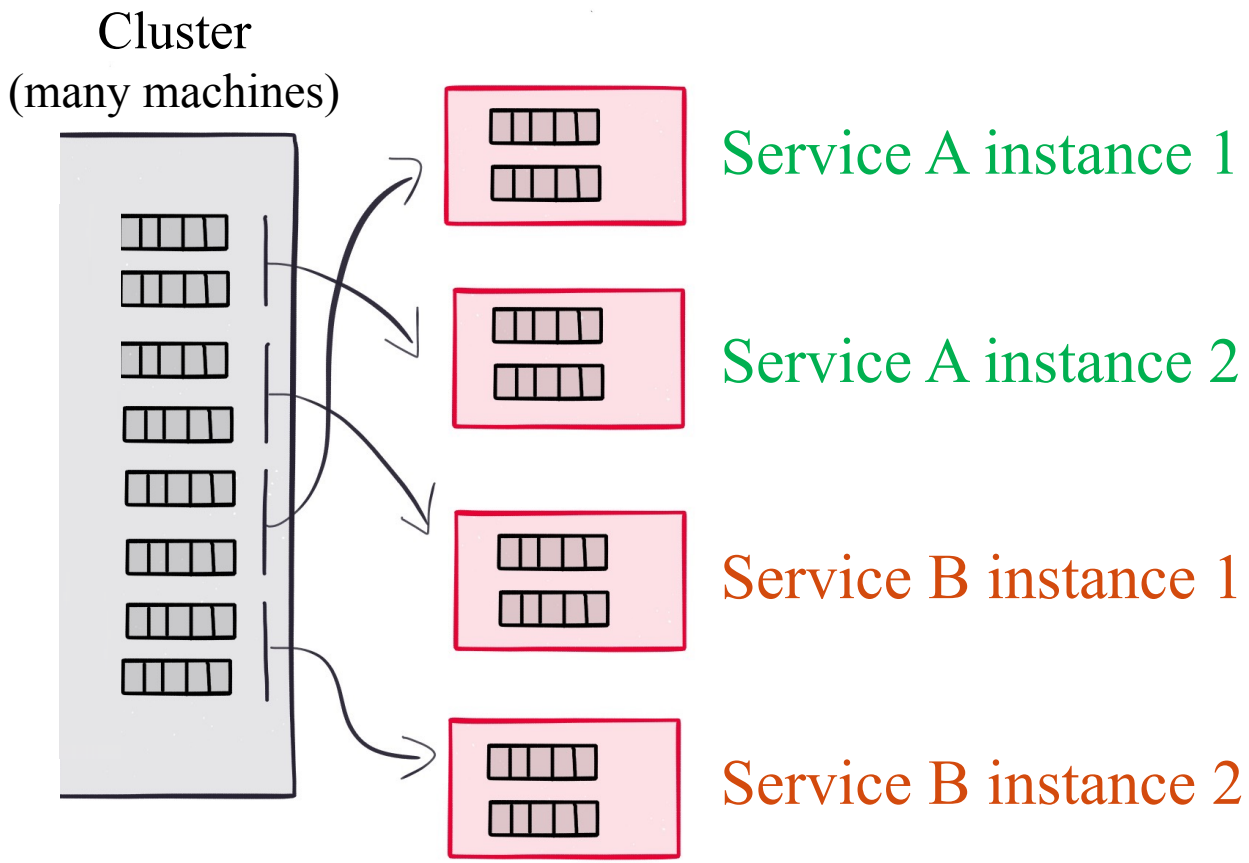
Agenda

1) Scalable Microservices
2) **Apache Kafka and Confluent Platform**
3) Kafka Streams
4) Scalable Microservices with Kafka and DC/OS
5) Live Demo

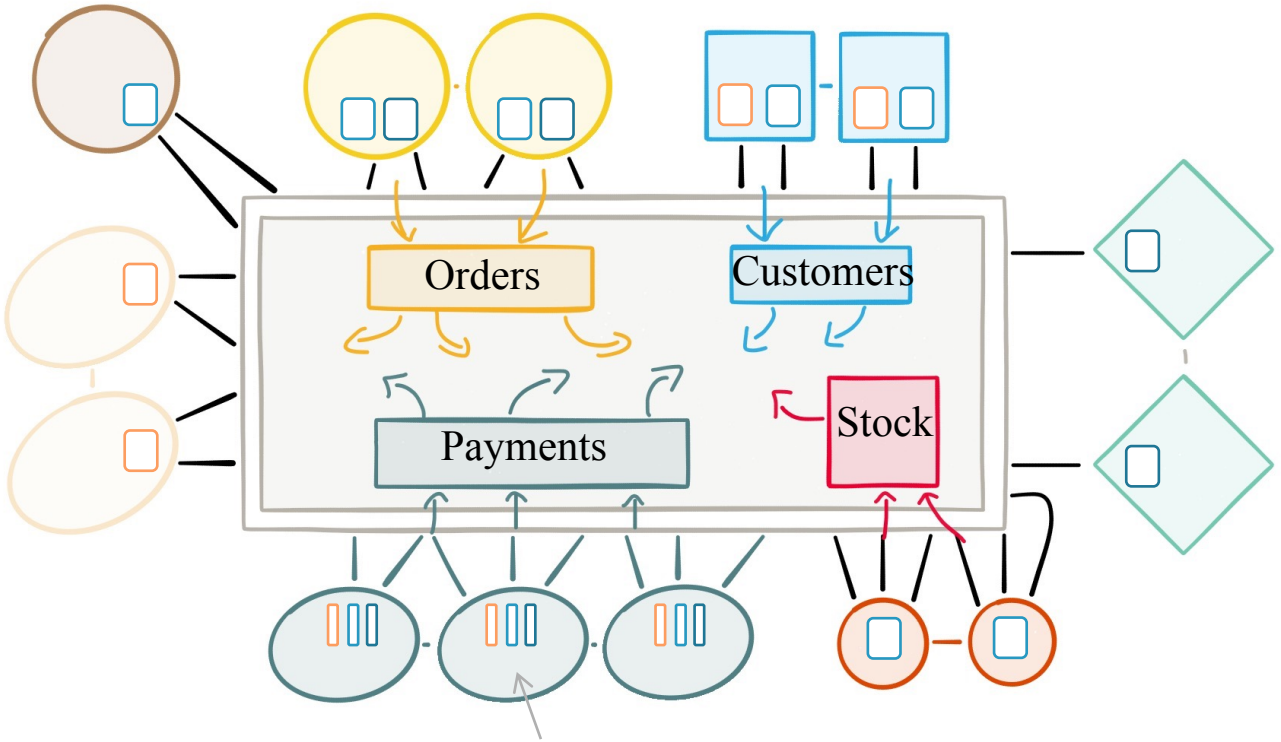
Apache Kafka – A Distributed, Fault-Tolerant, Scalable Commit Log



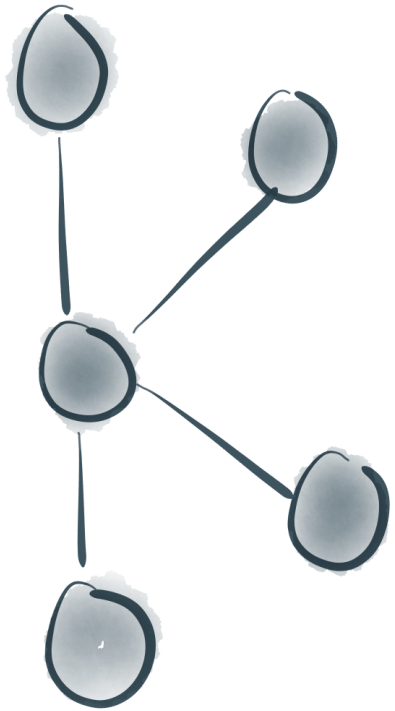
Scale in infrastructure terms



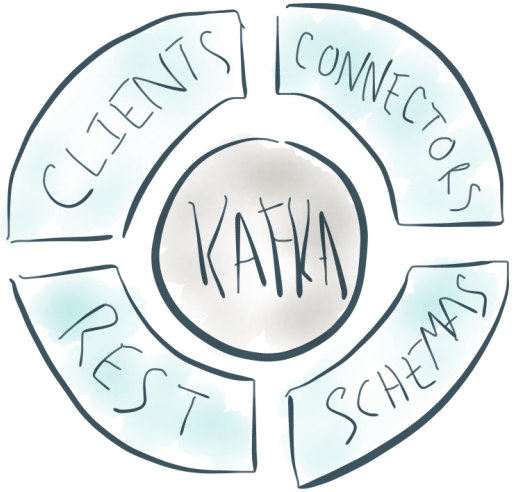
Single, Shared Source of Truth



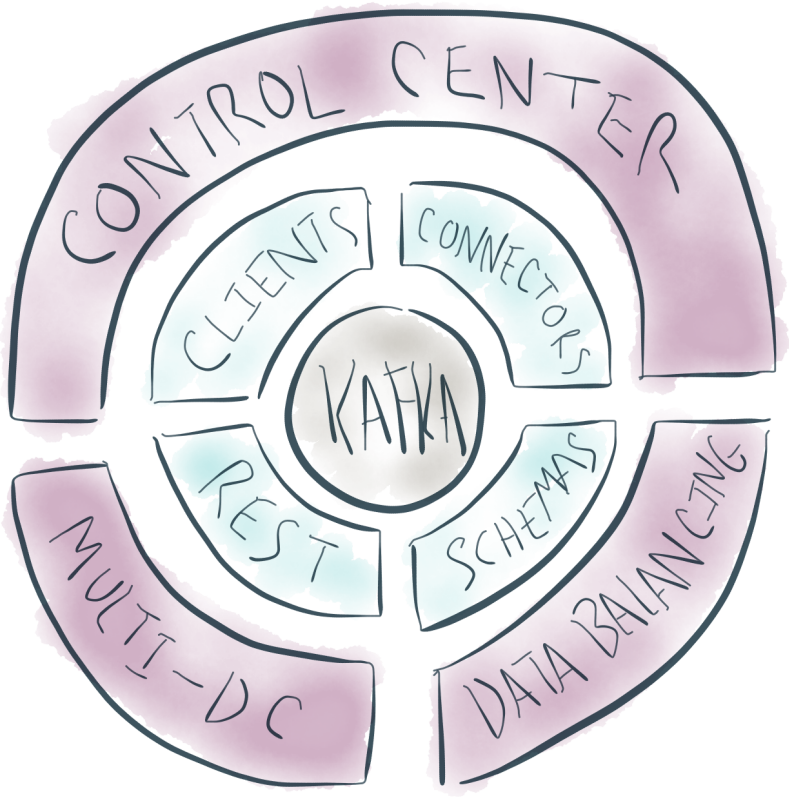
A MAJOR NEW ECOSYSTEM



CONFLUENT OPEN SOURCE



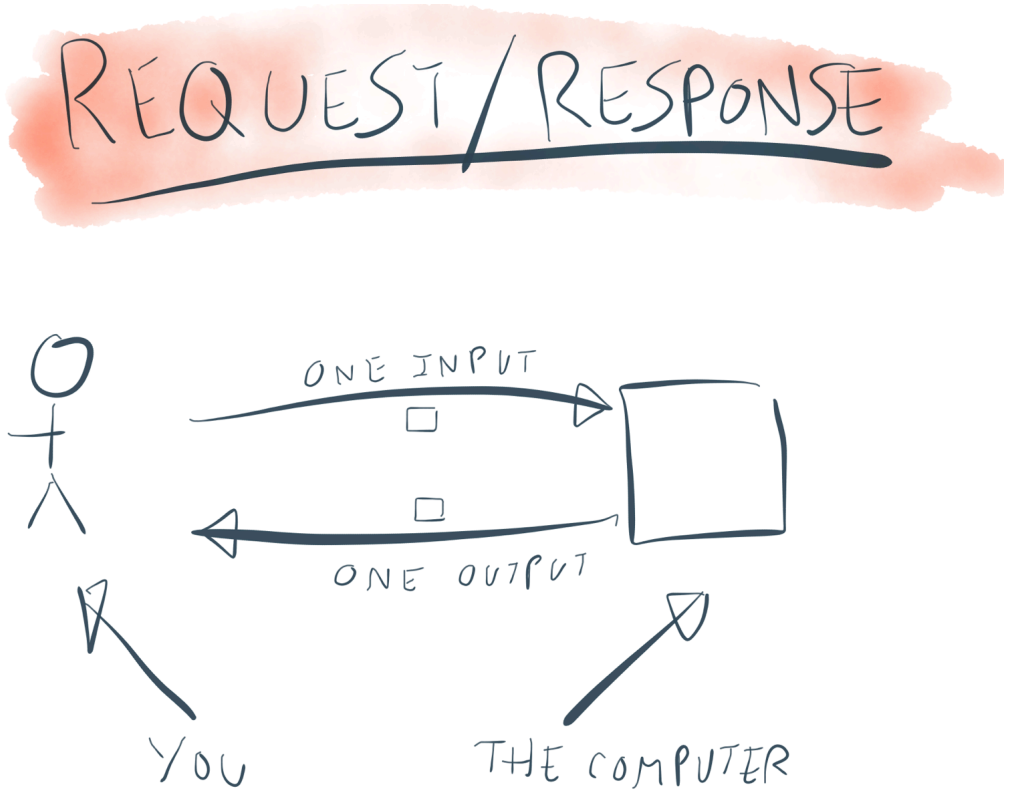
CONFLUENT ENTERPRISE

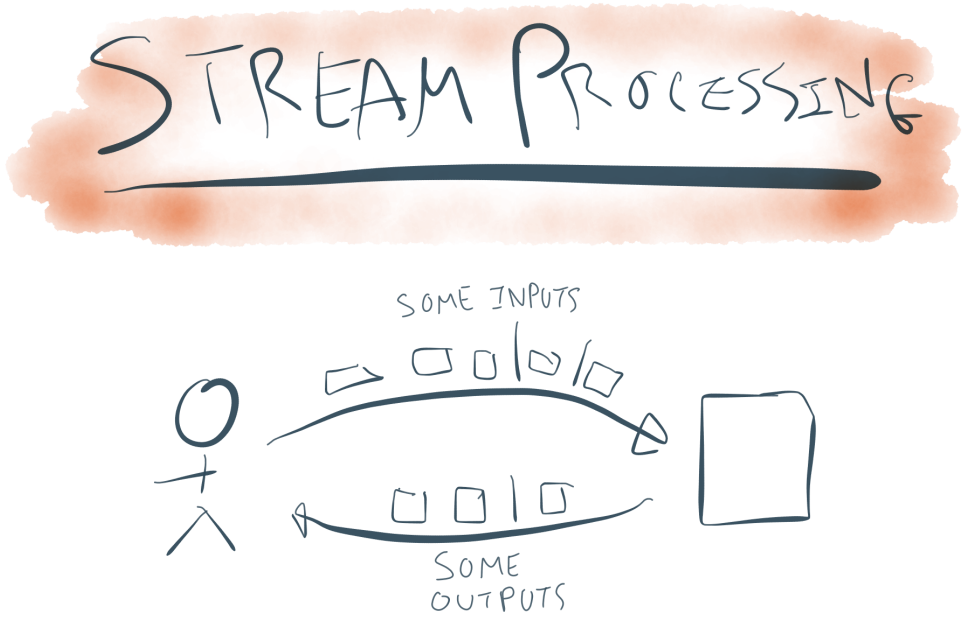


Agenda

1) Scalable Microservices
2) Apache Kafka and Confluent Platform
3) **Kafka Streams**
4) Scalable Microservices with Kafka and DC/OS
5) Live Demo

Definition of Stream Processing



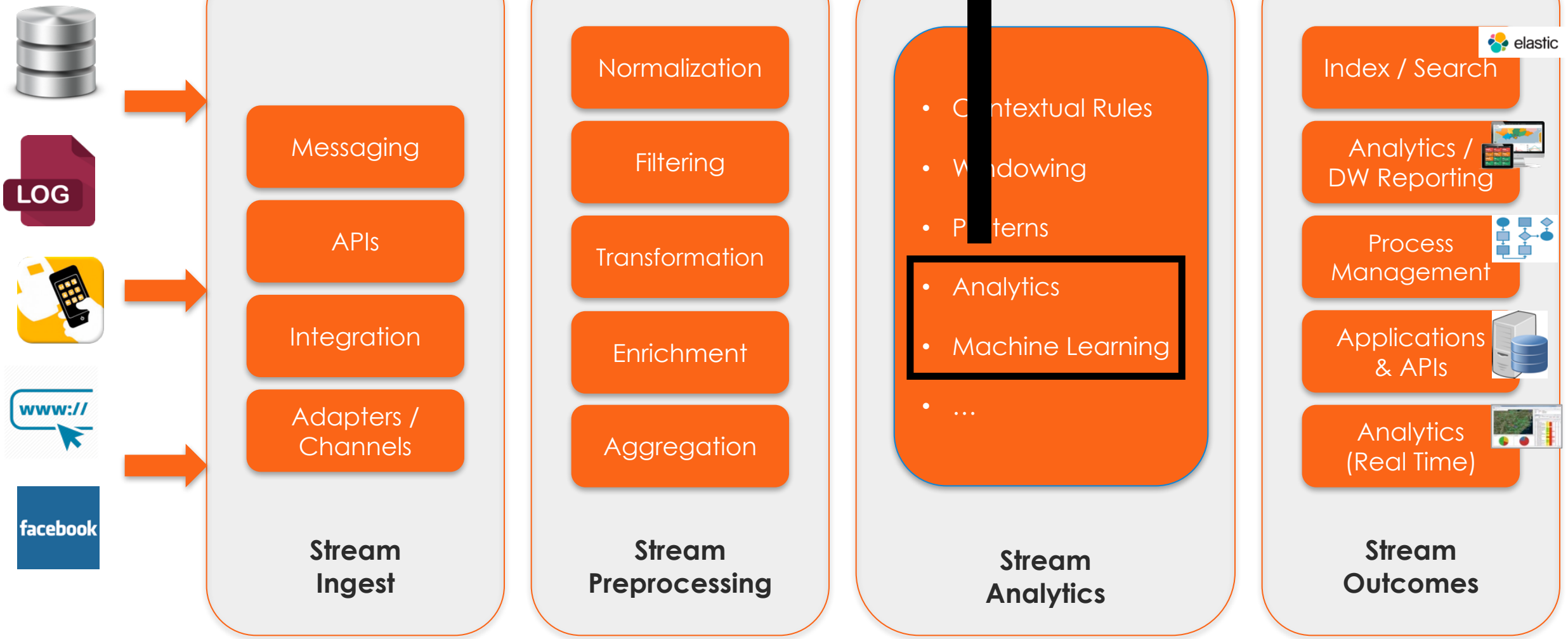


Data at Rest

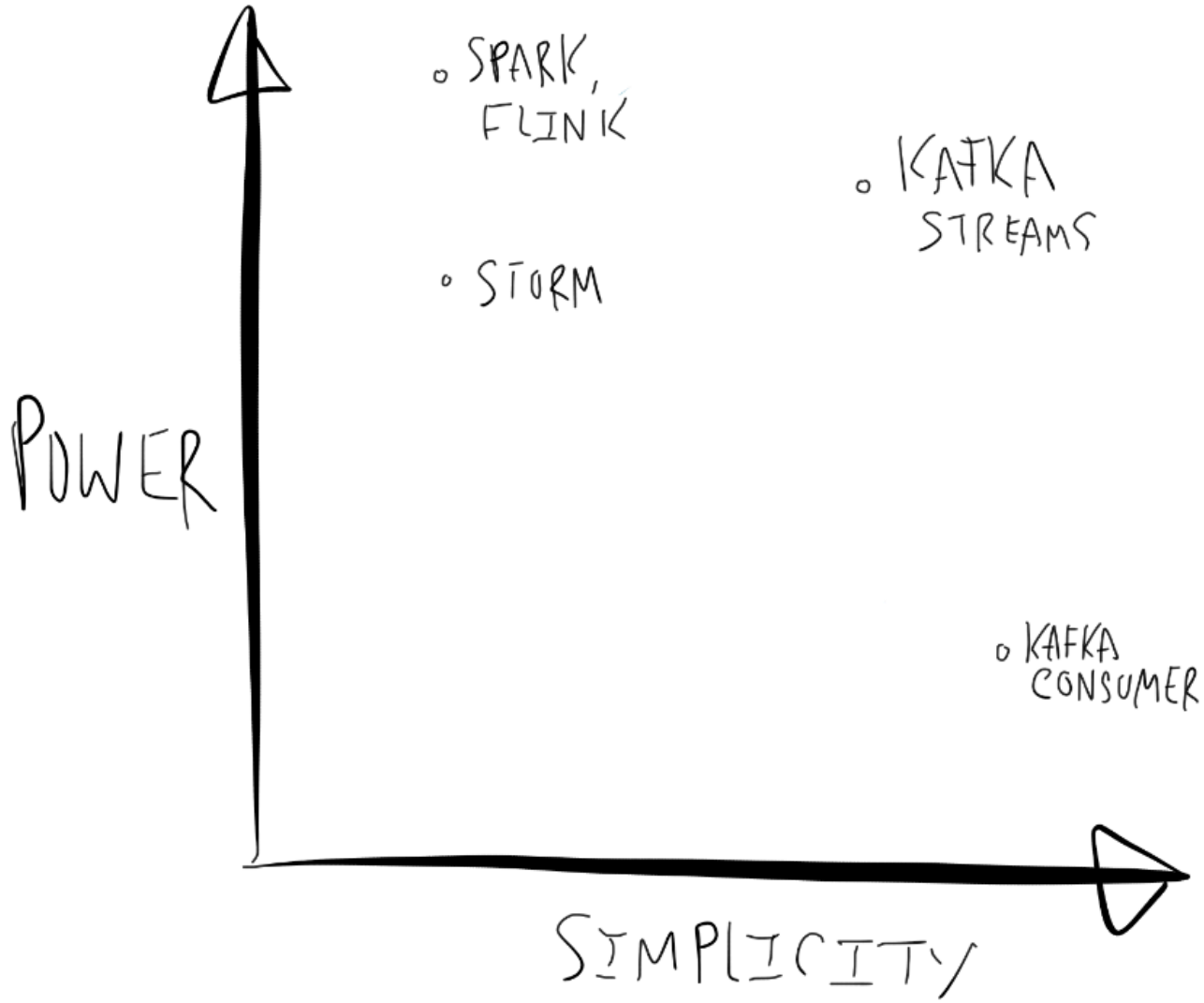
Data in Motion

Stream Processing Pipeline

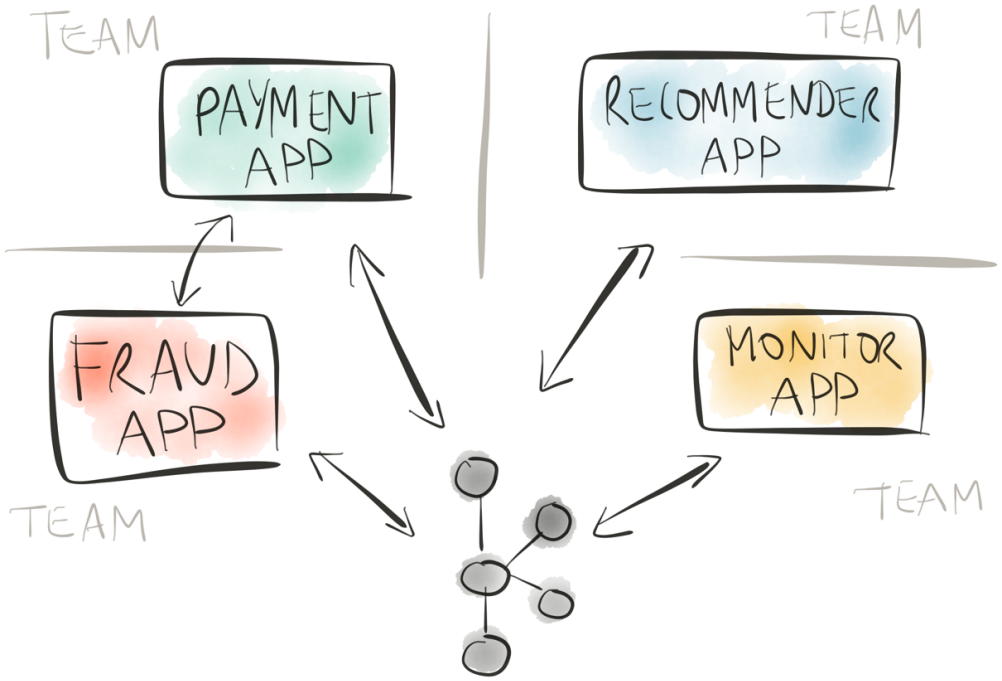
Applying an Analytic Model is just a piece of the puzzle!

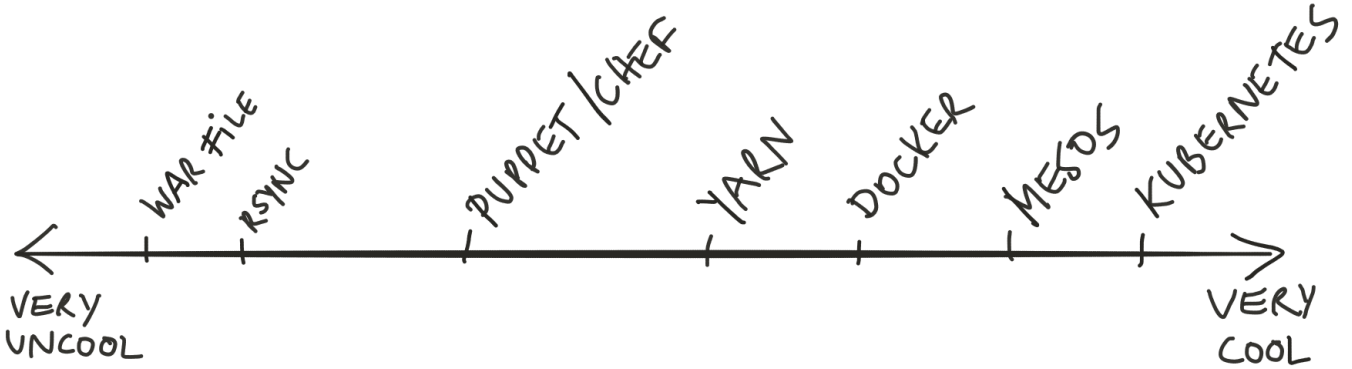


When to use Kafka Streams for Stream Processing?



When to use Kafka Streams for Stream Processing?





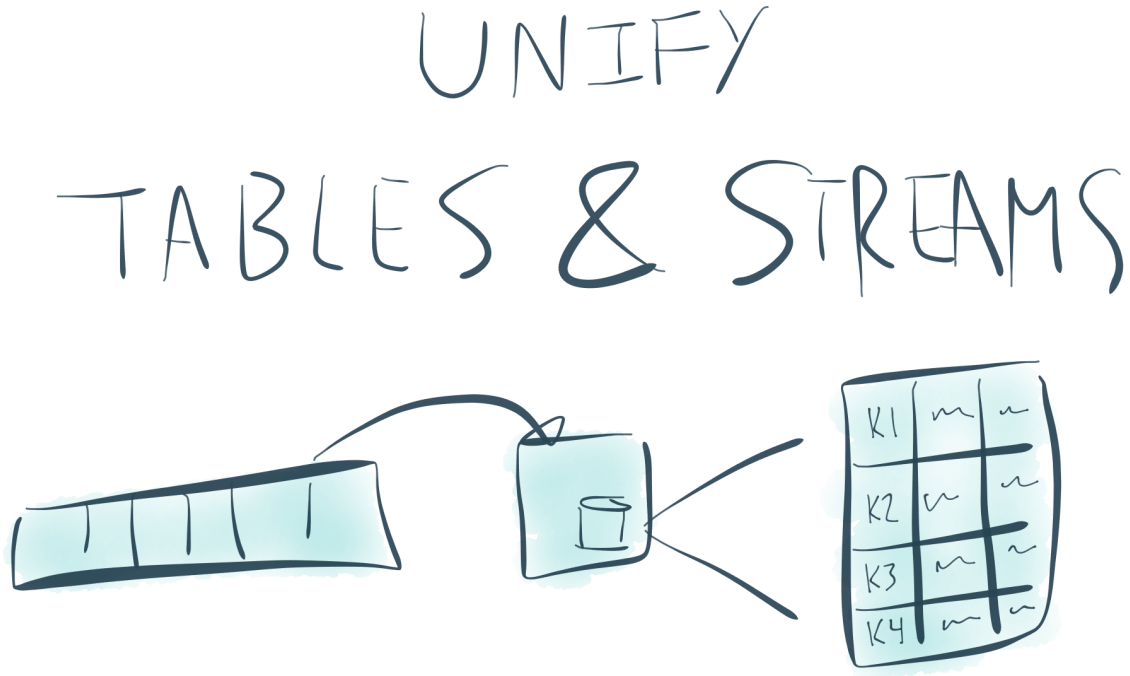
KAFKA STREAMS

- SIMPLE LIBRARY
- CONVENIENT DSL
- DATAFLOW STYLE WINDOWING
- REPROCESSING
- NO MICROBATCH
- LOCAL STATE

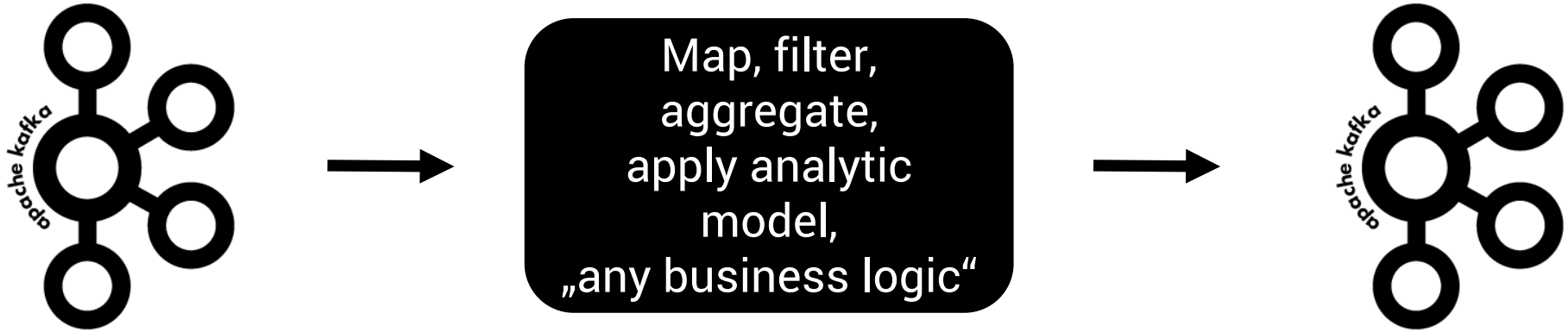
KEY OPERATIONS

- MAP
- FILTER
- AGGREGATE (COUNT, SUM, ETC)
- JOIN

Like Streams library or scala collections or reactive thingies BUT stateful, fault-tolerant, distributed


UNIFY
TABLES & STREAMS
K1
K2
K3
K4

Kafka Streams (shipped with Apache Kafka)



Input Stream
(Kafka Topic)

Stream Processing Microservice
(Kafka Streams)

Output Stream
(Kafka Topic)

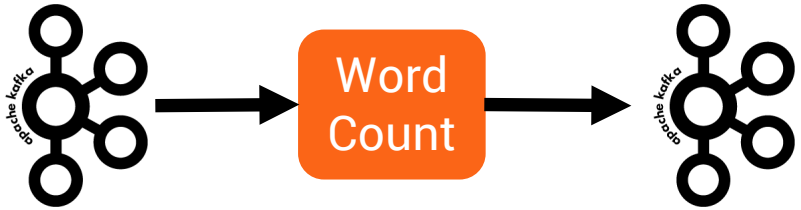
Kafka Cluster

Deployed Anywhere

Java App, Docker,
Kubernetes, Mesos,
“you-name-it”

Kafka Cluster

A complete streaming microservice, ready for production at large-scale



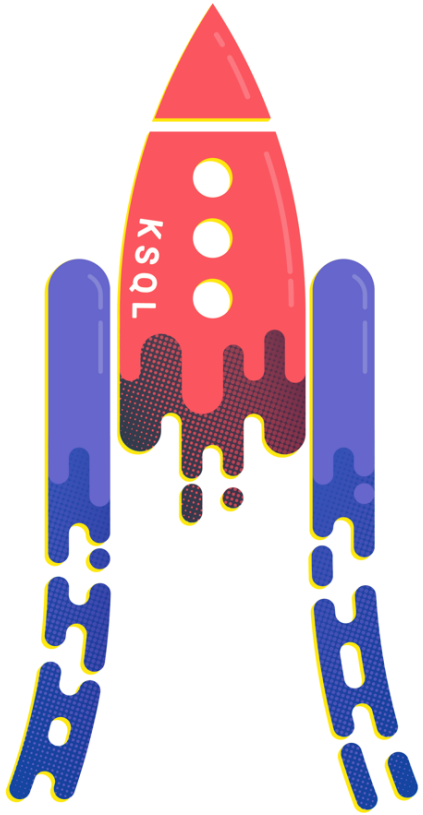

```
1  public static void main(final String[] args) throws Exception {
2    Properties config = new Properties();
3    config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-example");
4    config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092");
5    config.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
6    config.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
7
8    KStreamBuilder builder = new KStreamBuilder();
9    KStream<String, String> textLines = builder.stream("TextLinesTopic");
10   KStream<String, Long> wordCounts = textLines
11     .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
12     .groupBy((key, word) -> word)
13     .count("Counts")
14     .toStream();
15   wordCounts.to(Serdes.String(), Serdes.Long(), "WordsWithCountsTopic");
16
17   KafkaStreams streams = new KafkaStreams(builder, config);
18   streams.start();
19 }
```

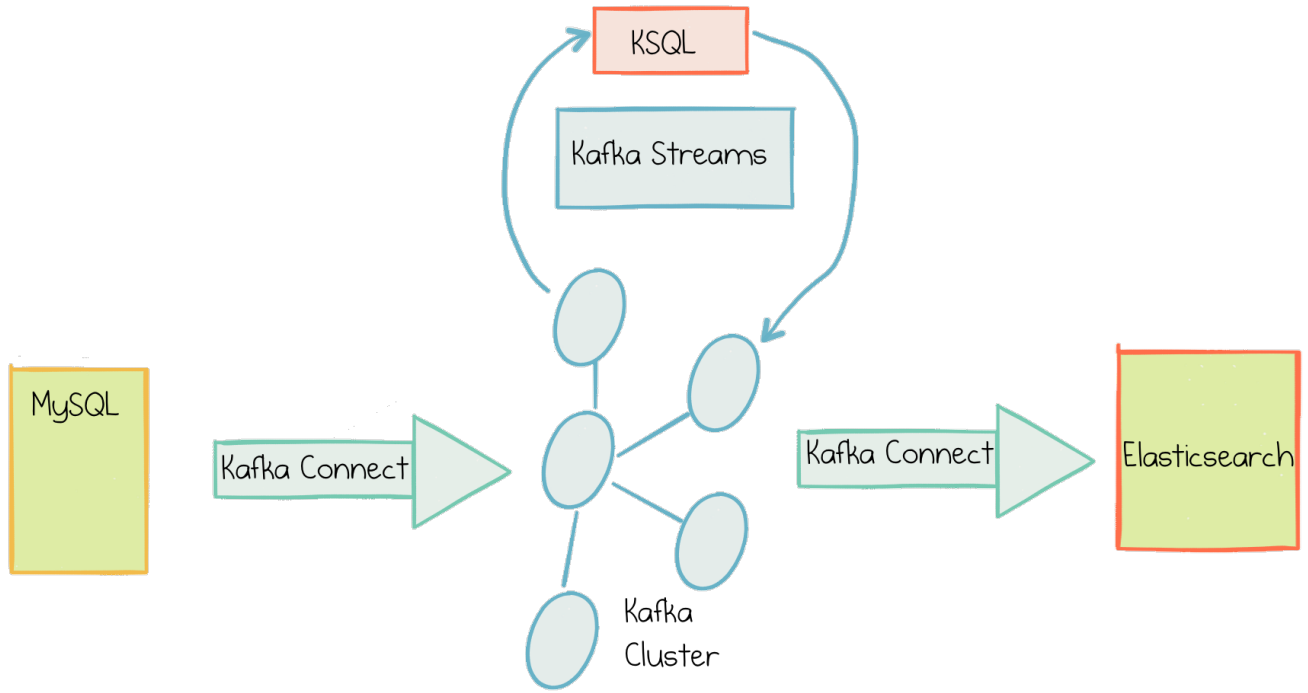
App configuration

Define processing (here: WordCount)

Start processing

KSQL – A Streaming SQL Engine for Apache Kafka



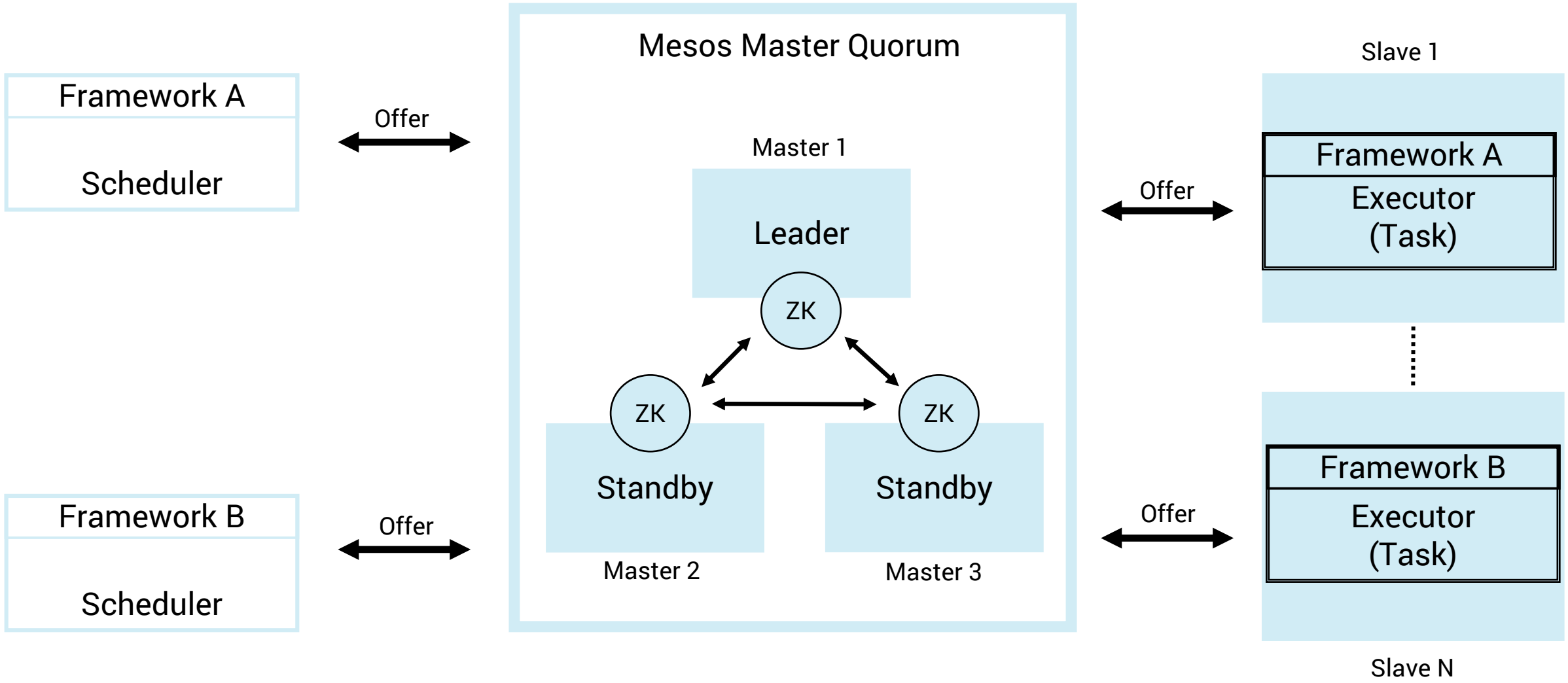



```
SELECT STREAM
  CEIL(timestamp TO HOUR) AS timeWindow, productId, COUNT(*) AS hourlyOrders, SUM(units) AS units
FROM Orders GROUP BY CEIL(timestamp TO HOUR), productId;

 timeWindow | productId | hourlyOrders | units
------------+-----------+--------------+-------
   08:00:00 |        10 |            2 |     5
   08:00:00 |        20 |            1 |     8
   09:00:00 |        10 |            4 |    22
   09:00:00 |        40 |            1 |    45
        ... |       ... |          ... |   ...
```

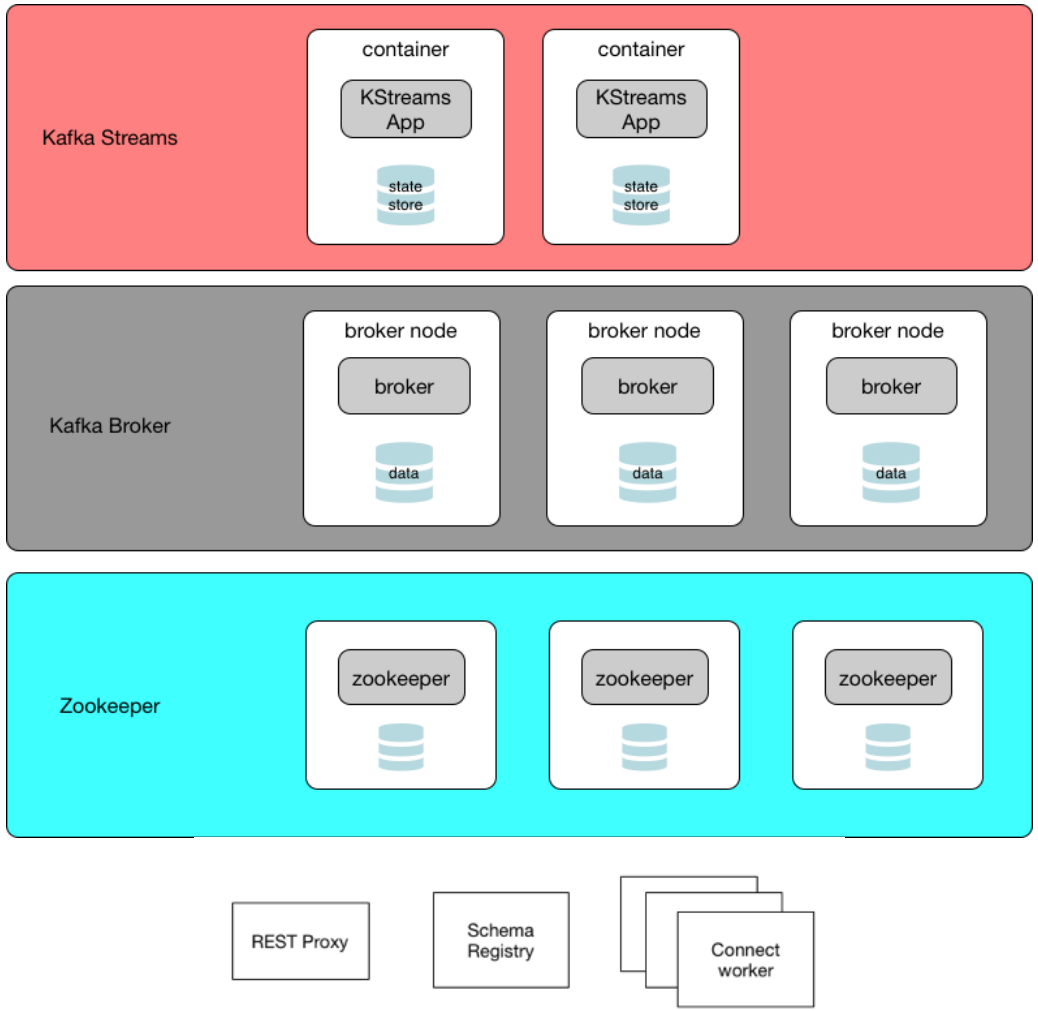
Agenda

1) Scalable Microservices
2) Apache Kafka and Confluent Platform
3) Kafka Streams
4) **Scalable Microservices with Kafka and DC/OS**
5) Live Demo

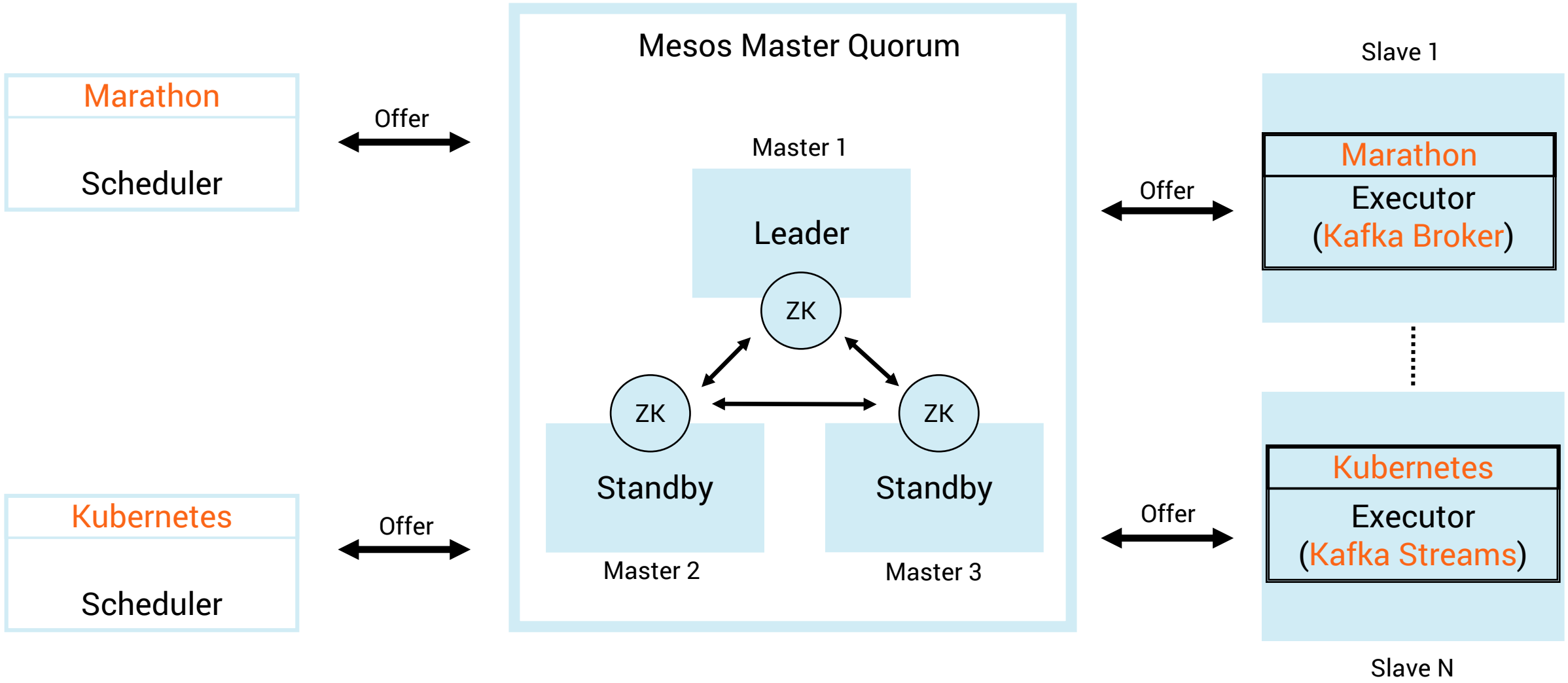
Mesos Architecture



Components of a Kafka Cluster



Mesos Architecture



Why DC/OS for the Kafka Ecosystem?

- Automated provisioning and upgrading of Kafka components
  - Broker, REST Proxy, Schema Registry, Connect …
  - Kafka applications (Java / Go / .NET / Python Clients, Kafka Streams, KSQL)
  - Monitoring (Confluent Control Center, etc.)

- Unified management and monitoring
  - Easy interactive installation
  - Multiple Kafka Cluster on one infrastructure + multi-tenancy
  - Combination with other Big Data components (Spark, Cassandra, etc.) on one infrastructure
  - Integration with syslog-compatible logging services for diagnostics and troubleshooting

- Elastic scaling, fault tolerance and self-healing
  - Stateful and stateless services
  - Service discovery and routing (using the corresponding Mesos framework, i.e. Marathon or Kubernetes)
  - Kafka VIP Connection (one “static” bootstrap server url)
  - Storage volumes for enhanced data durability, known as Mesos Dynamic Reservations and Persistent Volumes

Agenda

1) Scalable Microservices
2) Apache Kafka and Confluent Platform
3) Kafka Streams
4) Scalable Microservices with Kafka and DC/OS
5) **Live Demo**

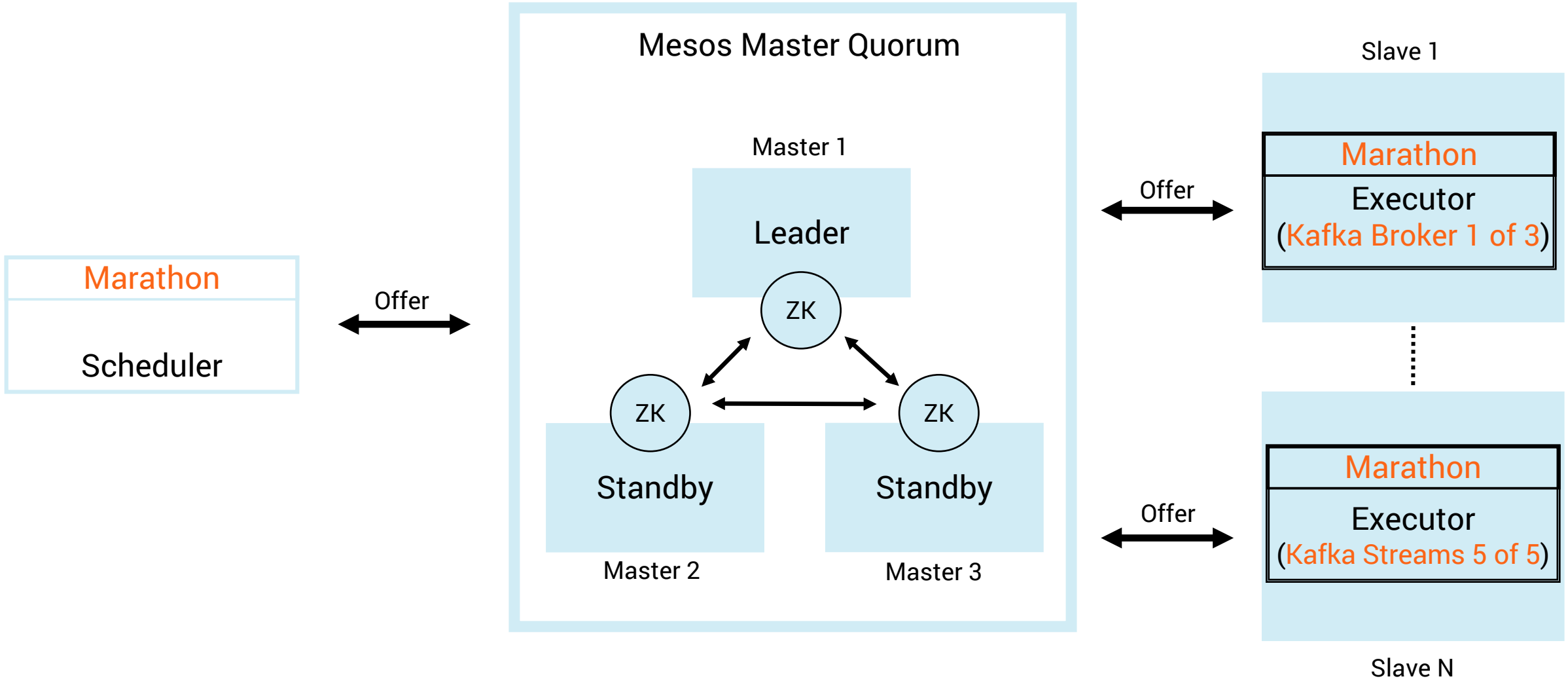
Live Demo

Use Case:
Airline Flight Delay Prediction

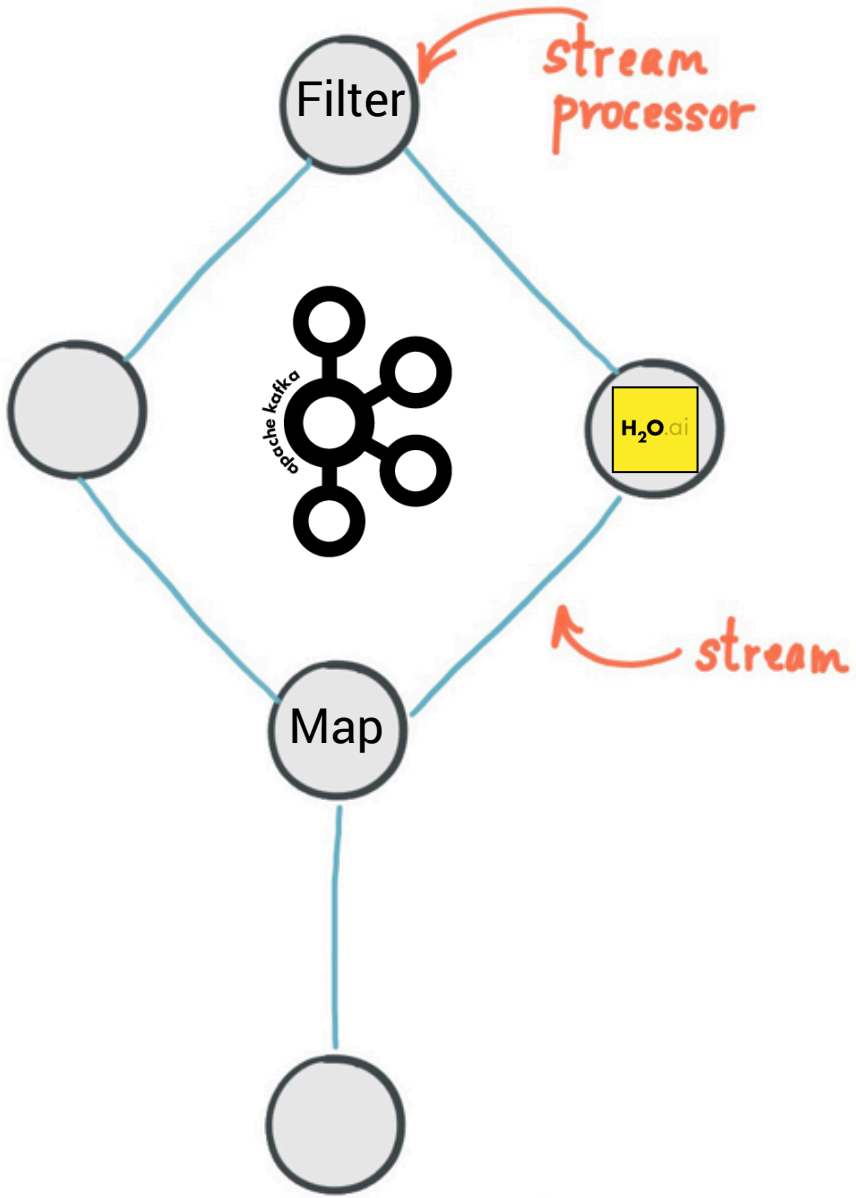
Machine Learning Algorithm:
Gradient Boosting (GBM)
using Decision Trees

Technologies:
DC/OS
Kafka Broker
Kafka Streams
H2O.ai

Architecture – Live Demo



H2O.ai Model + Kafka Streams



1) Create H2O ML Model

```
// Create H2O object (see gbm_pojo_test.java)
hex.genmodel.GenModel rawModel;
rawModel = (hex.genmodel.GenModel) Class.forName(modelClassName).newInstance();
EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);
```

2) Configure Kafka Streams Application

```
// Configure Kafka Streams Application
final String bootstrapServers = args.length > 0 ? args[0] : "localhost:9092";
final Properties streamsConfiguration = new Properties();
// Give the Streams application a unique name. The name must be unique
// in the Kafka cluster
// against which the application is run.
streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "machine-learning-example");
// Where to find Kafka broker(s).
streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
```

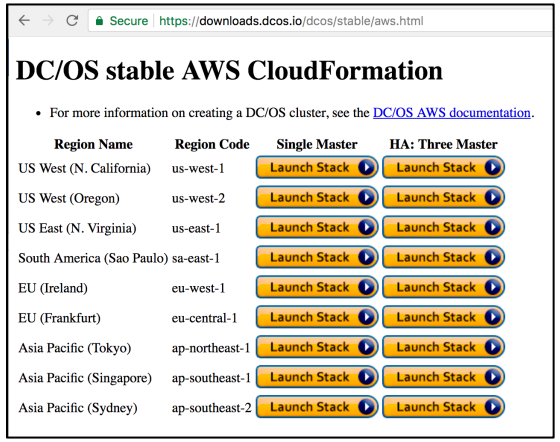
3) Apply H2O ML Model to Streaming Data

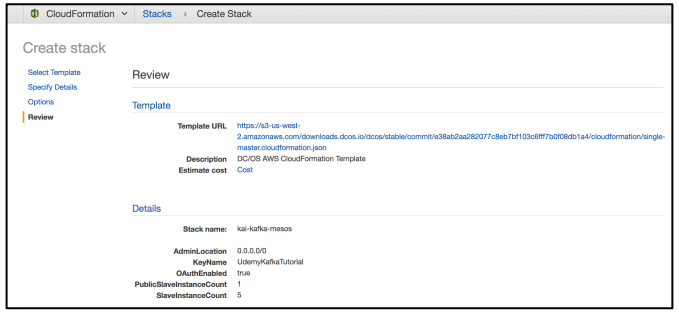
```
airlineInputLines.foreach(new ForeachAction<String, String>() {
    public void apply(String key, String value) {

        // Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,FlightNum,TailNum,Actu
        // value:
        // 1987,10,14,3,741,730,912,849,PS,1451,NA,91,79,NA,23,11,SAN,SFO,447,NA,NA,0,NA,0,NA,NA,NA,NA,NA,YES,YES

        if (value != null && !value.equals("")) {
            System.out.println("#####################");
            System.out.println("Flight Input:" + value);

            String[] valuesArray = value.split(",");

            RowData row = new RowData();
            row.put("Year", valuesArray[0]);
            row.put("Month", valuesArray[1]);
            row.put("DayofMonth", valuesArray[2]);
            row.put("DayOfWeek", valuesArray[3]);
            row.put("CRSDepTime", valuesArray[5]);
            row.put("UniqueCarrier", valuesArray[8]);
            row.put("Origin", valuesArray[16]);
            row.put("Dest", valuesArray[17]);
            BinomialModelPrediction p = null;
            try {
                p = model.predictBinomial(row);
            } catch (PredictException e) {
                e.printStackTrace();
            }

            System.out.println("Label (aka prediction) is flight departure delayed: " + p.label);
            System.out.print("Class probabilities: ");
            for (int i = 0; i < p.classProbabilities.length; i++) {
                if (i > 0) {
                    System.out.print(",");
                }
                System.out.print(p.classProbabilities[i]);
            }
```

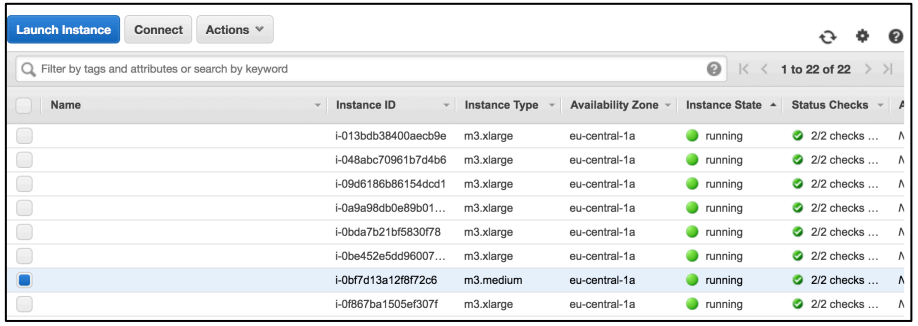
4) Start Kafka Streams App

```
KStream<String, Object> transformedMessage = airlineInputLines.mapValues(value -> "KAI");

transformedMessage.to("AirlineOutputTopic");

final KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);

streams.start();
```

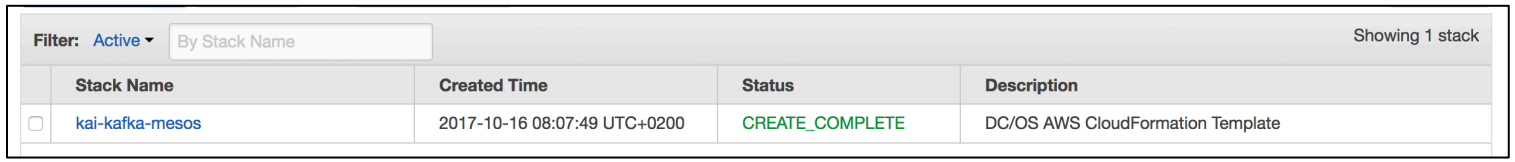
DC/OS on AWS





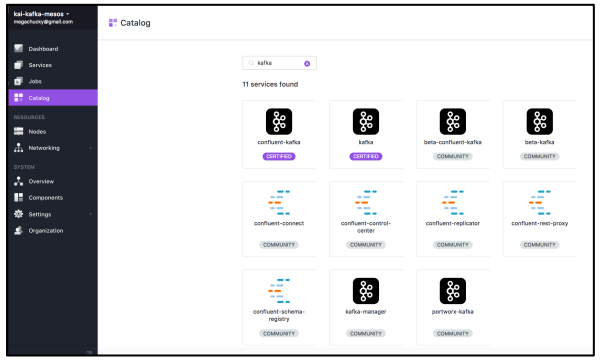


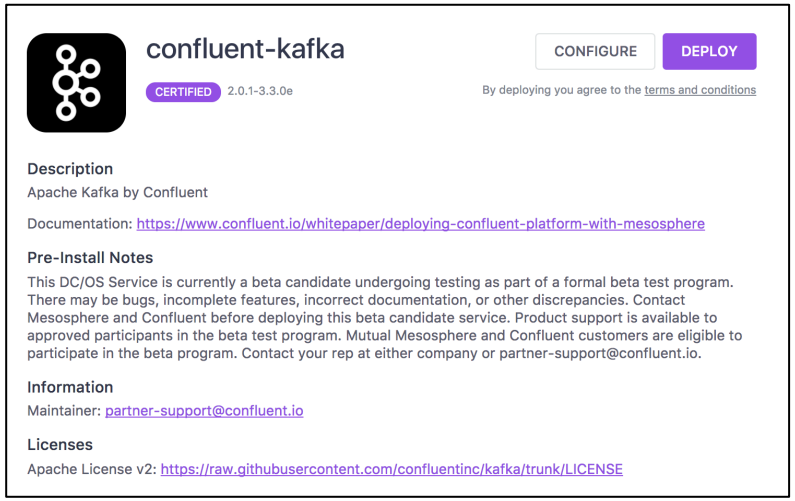
Logical ID	Physical ID	Type	Status
AdminSecurityGroup	sg-4ddf0c27	AWS::EC2::SecurityGroup	CREATE_COMPLETE
DHCPOptions	dopt-3d785355	AWS::EC2::DHCPOptions	CREATE_COMPLETE
ElasticLoadBalancer	kai-kafka-ElasticL-1CC7BQF1NQU83	AWS::ElasticLoadBalancing::...	CREATE_COMPLETE
ExhibitorS3Bucket	kai-kafka-mesos-exhibitors3bucket-fvl6h731h5i8	AWS::S3::Bucket	CREATE_COMPLETE
GatewayToInternet	kai-k-Gatew-N67EBATROY0G	AWS::EC2::VPCGatewayAtta...	CREATE_COMPLETE
InboundNetworkAclEntry	kai-k-Inbou-15NK4D5HF8YIL	AWS::EC2::NetworkAclEntry	CREATE_COMPLETE
InternalMasterLoadBal...	kai-kafka-Internal-1MH6OA6BQ9G5D	AWS::ElasticLoadBalancing::...	CREATE_COMPLETE
InternetGateway	igw-e2f4288a	AWS::EC2::InternetGateway	CREATE_COMPLETE
LbSecurityGroup	sg-4cdf0c26	AWS::EC2::SecurityGroup	CREATE_COMPLETE
MasterInstanceProfile	kai-kafka-mesos-MasterInstanceProfile-J0UX3T5WNR0K	AWS::IAM::InstanceProfile	CREATE_COMPLETE
MasterLaunchConfig	kai-kafka-mesos-MasterLaunchConfig-1MB0MEPZ4DD7O	AWS::AutoScaling::LaunchCo...	CREATE_COMPLETE
MasterRole	kai-kafka-mesos-MasterRole-1UCTGMIDFEWKC	AWS::IAM::Role	CREATE_COMPLETE
MasterSecurityGroup	sg-4ede0d24	AWS::EC2::SecurityGroup	CREATE_COMPLETE
MasterServerGroup	kai-kafka-mesos-MasterServerGroup-4O40KL2JNEPL	AWS::AutoScaling::AutoScali...	CREATE_COMPLETE
MasterToMasterIngress	MasterToMasterIngress	AWS::EC2::SecurityGroupIngr...	CREATE_COMPLETE
MasterToPublicSlaveIn...	MasterToPublicSlaveIngress	AWS::EC2::SecurityGroupIngr...	CREATE_COMPLETE

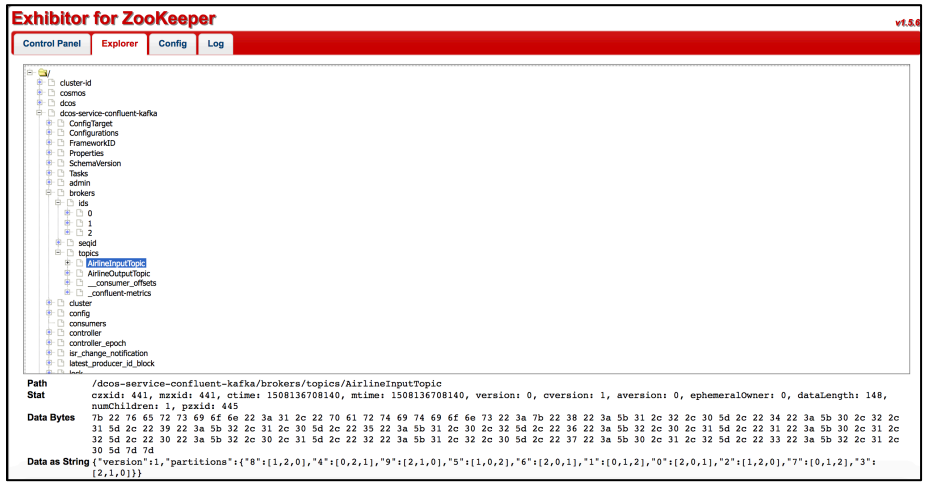


https://downloads.dcos.io/dcos/stable/aws.html

Kafka Brokers on DC/OS

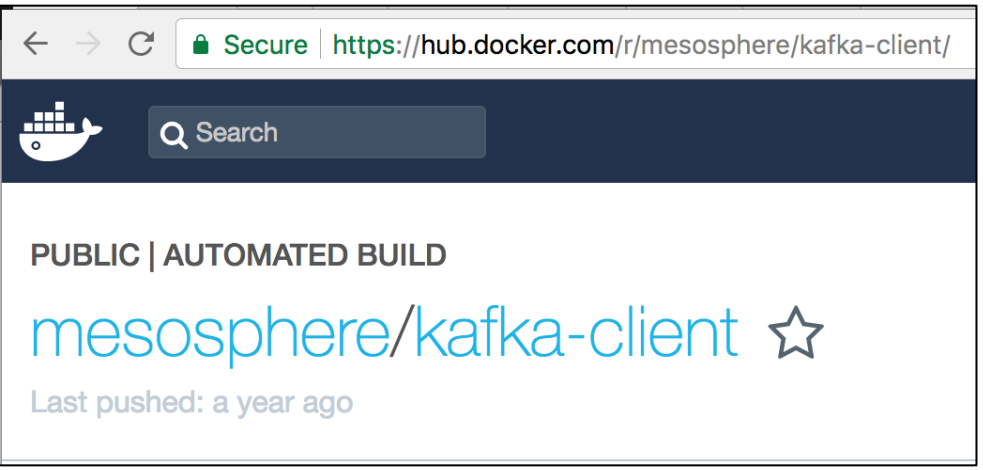


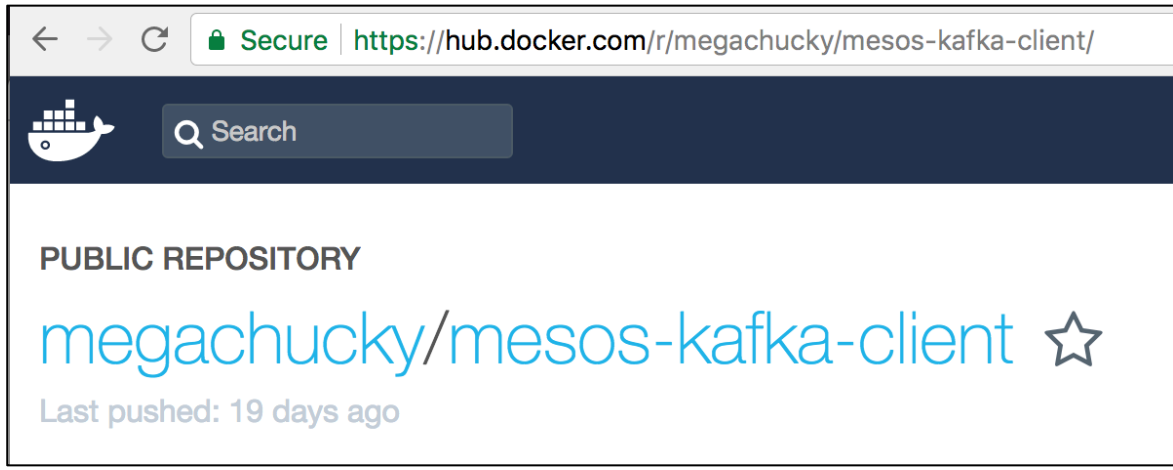





```
[~: dcos kafka --name confluent-kafka endpoints broker
{
  "address": [
    "10.0.2.155:1025",
    "10.0.2.118:1025",
    "10.0.2.211:1025"
  ],
  "dns": [
    "kafka-0-broker.confluent-kafka.autoip.dcos.thisdcos.directory:1025",
    "kafka-1-broker.confluent-kafka.autoip.dcos.thisdcos.directory:1025",
    "kafka-2-broker.confluent-kafka.autoip.dcos.thisdcos.directory:1025"
  ],
  "vip": "broker.confluent-kafka.l4lb.thisdcos.directory:9092"
}
```

Kafka Client (compatible with Kafka Streams) on DC/OS





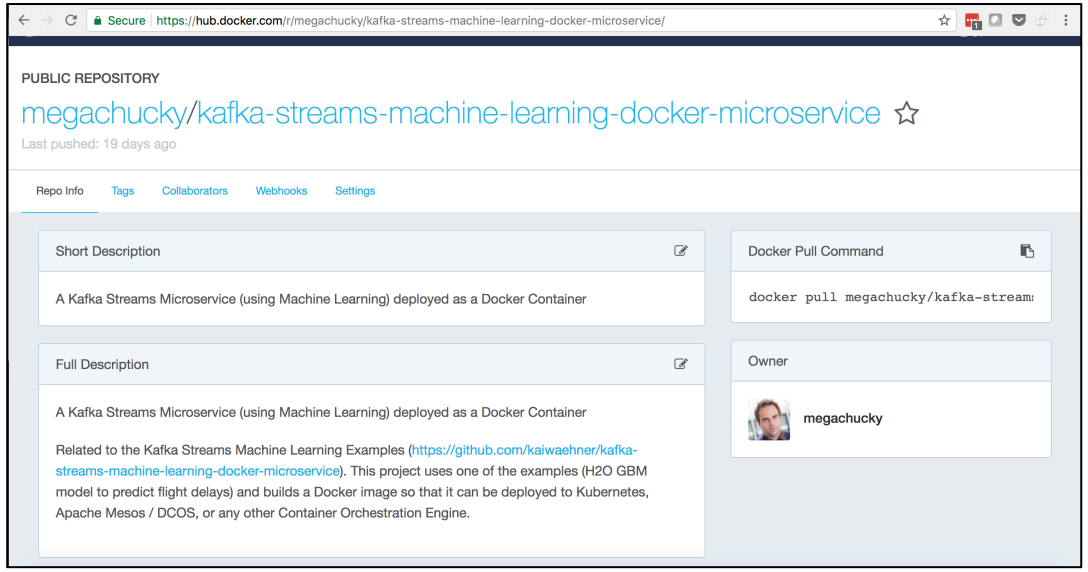
From Apache Kafka 0.9 to 0.11 (Kafka Streams messages require timestamps)

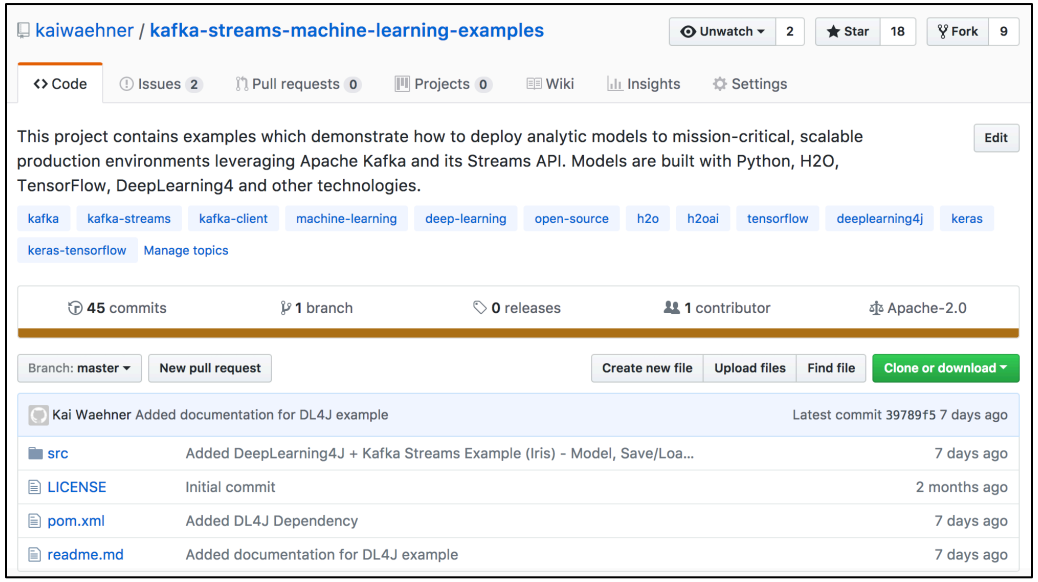
dcos node ssh --master-proxy –leader
docker run -it megachucky/mesos-kafka-client

Dockerfile:
FROM java:openjdk-8-jreMAINTAINER Kai Waehner
curl http://apache.mirrors.spacedump.net/kafka/0.11.0.1/kafka_2.11-0.11.0.1.tgz | tar xvz --strip-components=1
WORKDIR /bin

https://hub.docker.com/r/mesosphere/kafka-client/
https://hub.docker.com/r/megachucky/mesos-kafka-client/

Kafka Streams Microservice



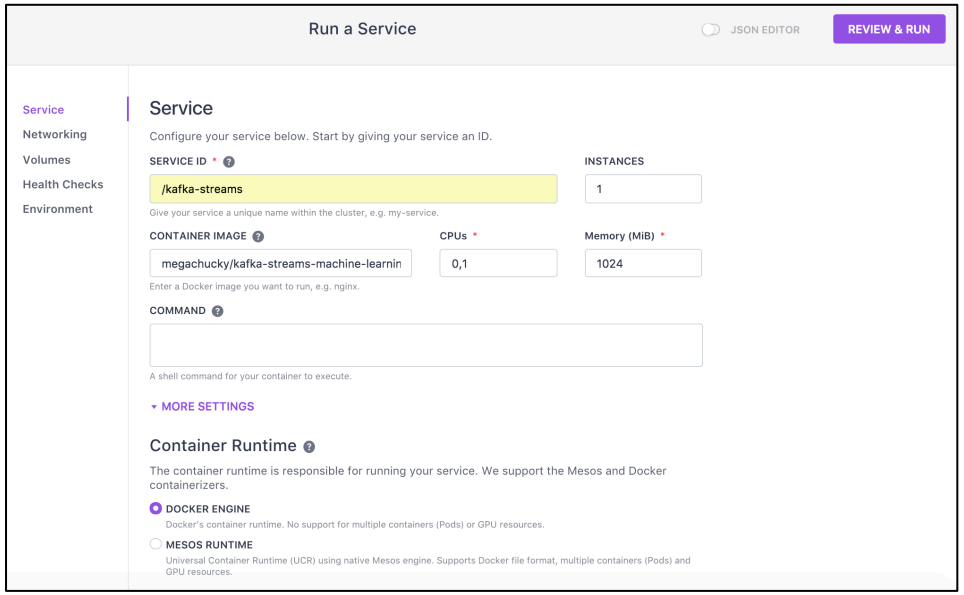


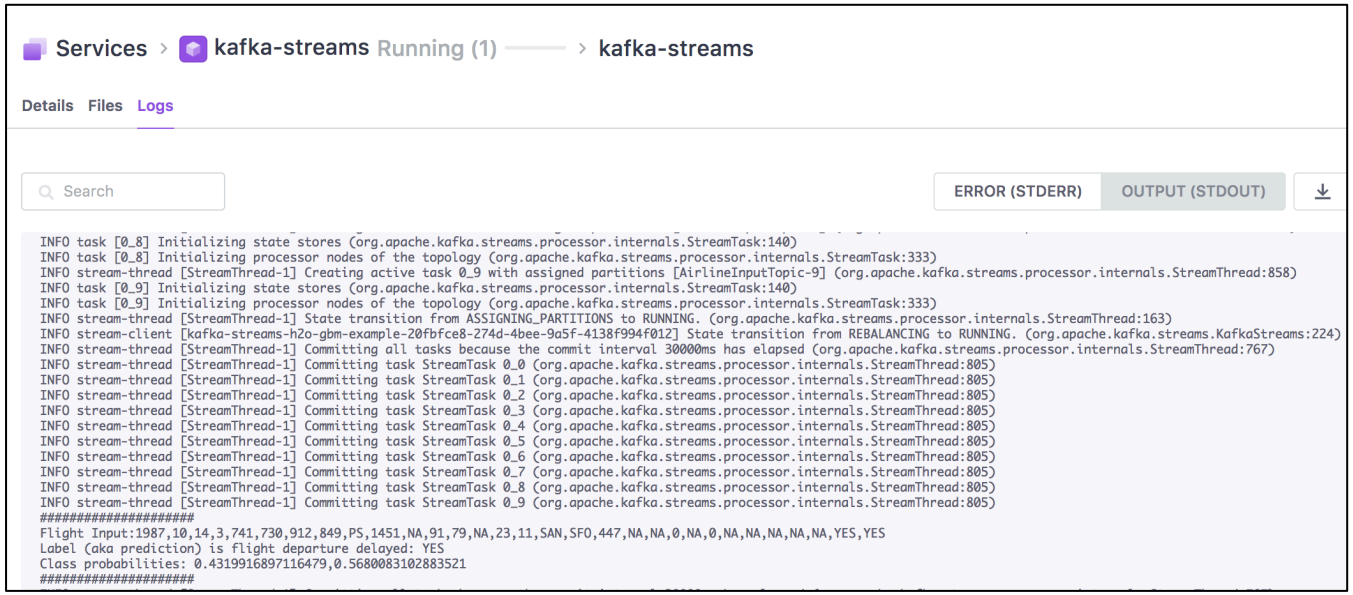
Dockerfile:
```
FROM java:8
ADD /opt/kafka-streams-h2o-docker-microservice-1.0-SNAPSHOT-jar-with-dependencies.jar /opt/
ENTRYPOINT ["java", "-jar", "/opt/kafka-streams-h2o-docker-microservice-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

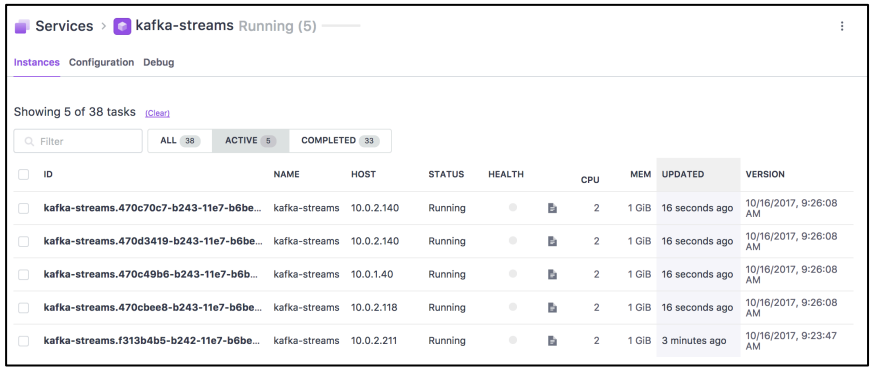
dcos kafka --name confluent-kafka topic create AirlineInputTopic --partitions 10 --replication 3
dcos kafka --name confluent-kafka topic create AirlineOutputTopic --partitions 10 --replication 3

https://hub.docker.com/r/megachucky/kafka-streams-machine-learning-docker-microservice/
https://github.com/kaiwaehner/kafka-streams-machine-learning-docker-microservice

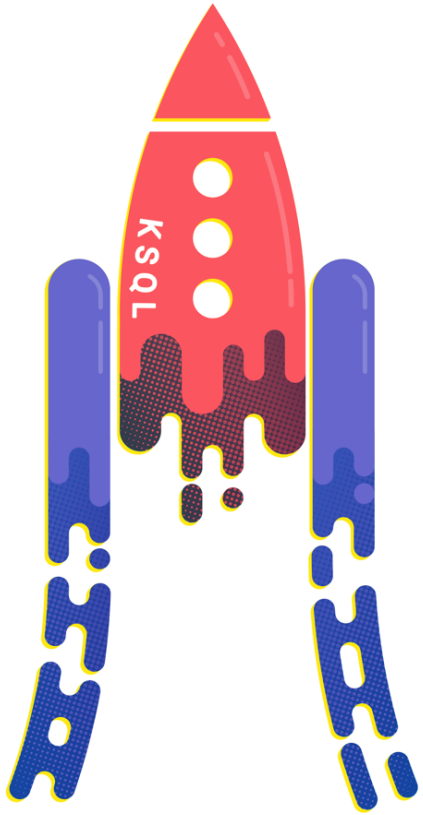
Kafka Streams Microservice on DC/OS

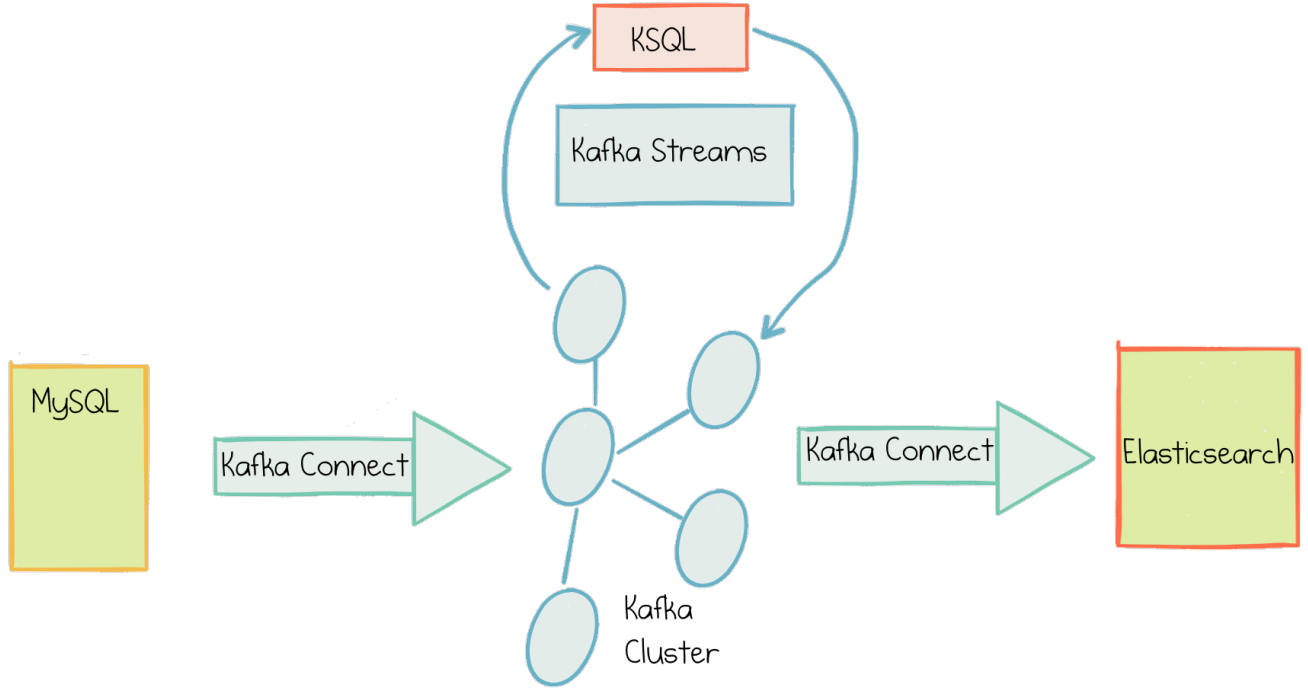






KSQL on DC/OS – Why not?






```
SELECT STREAM
  CEIL(timestamp TO HOUR) AS timeWindow, productId, COUNT(*) AS hourlyOrders, SUM(units) AS units
FROM Orders GROUP BY CEIL(timestamp TO HOUR), productId;

 timeWindow | productId | hourlyOrders | units
------------+-----------+--------------+-------
   08:00:00 |        10 |            2 |     5
   08:00:00 |        20 |            1 |     8
   09:00:00 |        10 |            4 |    22
   09:00:00 |        40 |            1 |    45
        ... |       ... |          ... |   ...
```

Key Take-Aways

- Apache Kafka Ecosystem on DC/OS for Highly Scalable, Fault-Tolerant Microservices
- DC/OS offers many Kafka Features out-of-the-box (one-click-provisioning, VIP connection, …)
- Kafka Streams Microservices run and scale on DC/OS via Marathon or Kubernetes

Questions? Feedback? Please contact me!

Kai Waehner

Technology Evangelist

kontakt@kai-waehner.de
@KaiWaehner
www.kai-waehner.de
LinkedIn