



# Apache Kafka + Apache Mesos

## Highly Scalable Streaming Microservices with Kafka Streams

---

**Kai Waehner**

Technology Evangelist

[kontakt@kai-waehner.de](mailto:kontakt@kai-waehner.de)

LinkedIn

[@KaiWaehner](https://www.linkedin.com/in/@KaiWaehner)

[www.confluent.io](http://www.confluent.io)

[www.kai-waehner.de](http://www.kai-waehner.de)



# Agenda

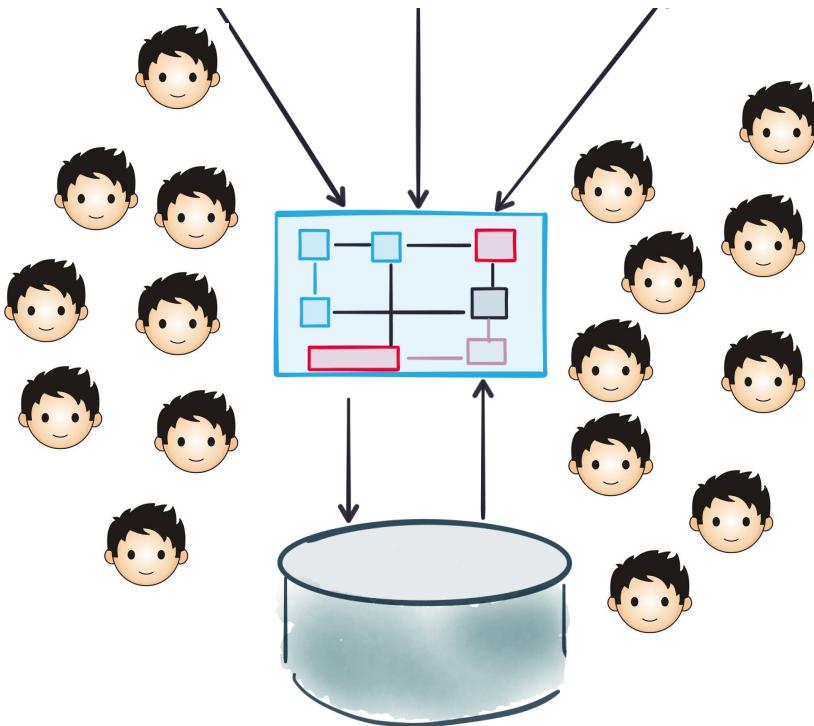
---

- 1) Scalable Microservices
- 2) Apache Kafka and Confluent Platform
- 3) Kafka Streams
- 4) Scalable Microservices with Kafka and DC/OS
- 5) Use Case - Scalable Flight Prediction

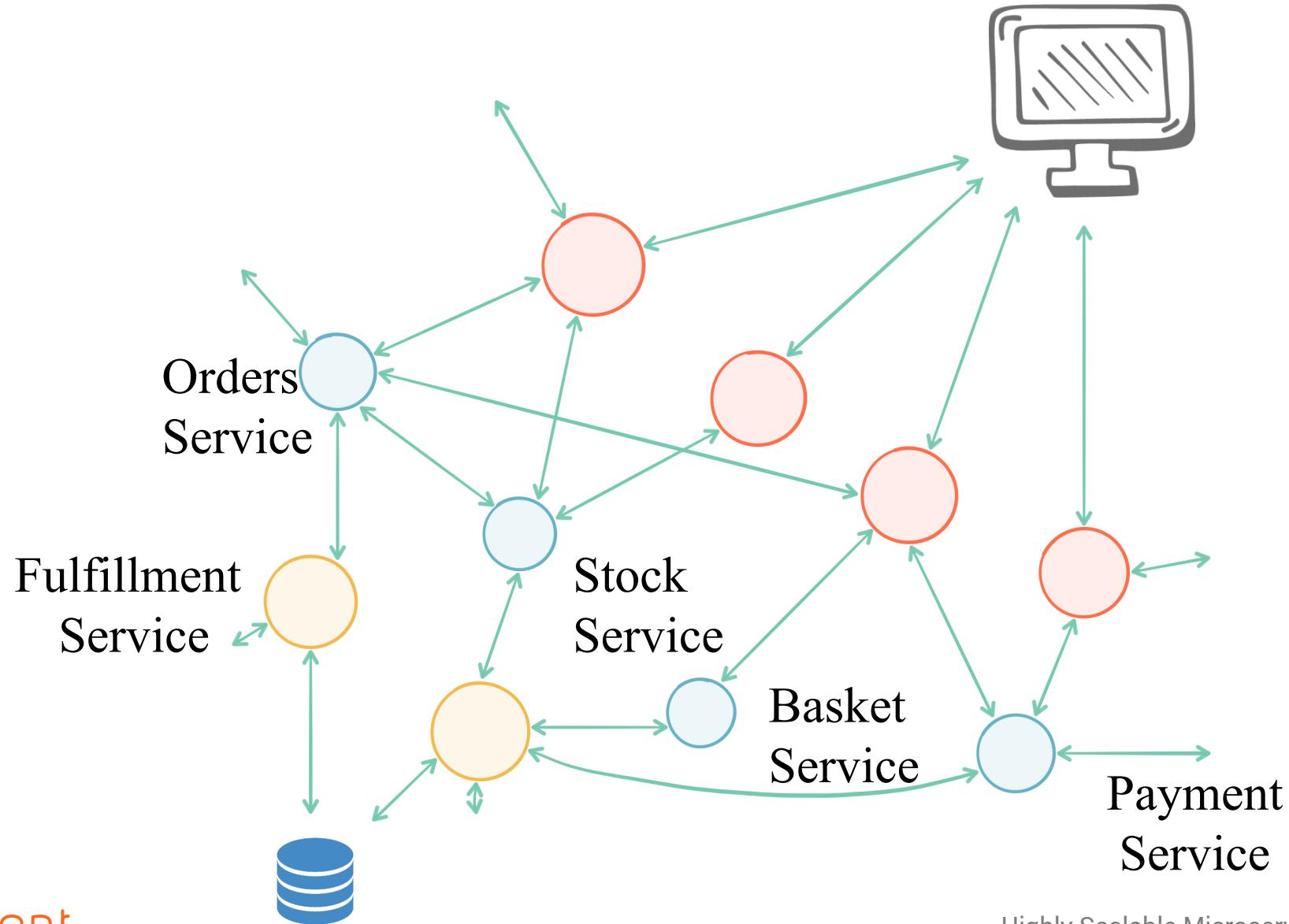
- 1) Scalable Microservices**
- 2) Apache Kafka and Confluent Platform
- 3) Kafka Streams
- 4) Scalable Microservices with Kafka and DC/OS
- 5) Use Case - Scalable Flight Prediction

# Moving away from the Monolith

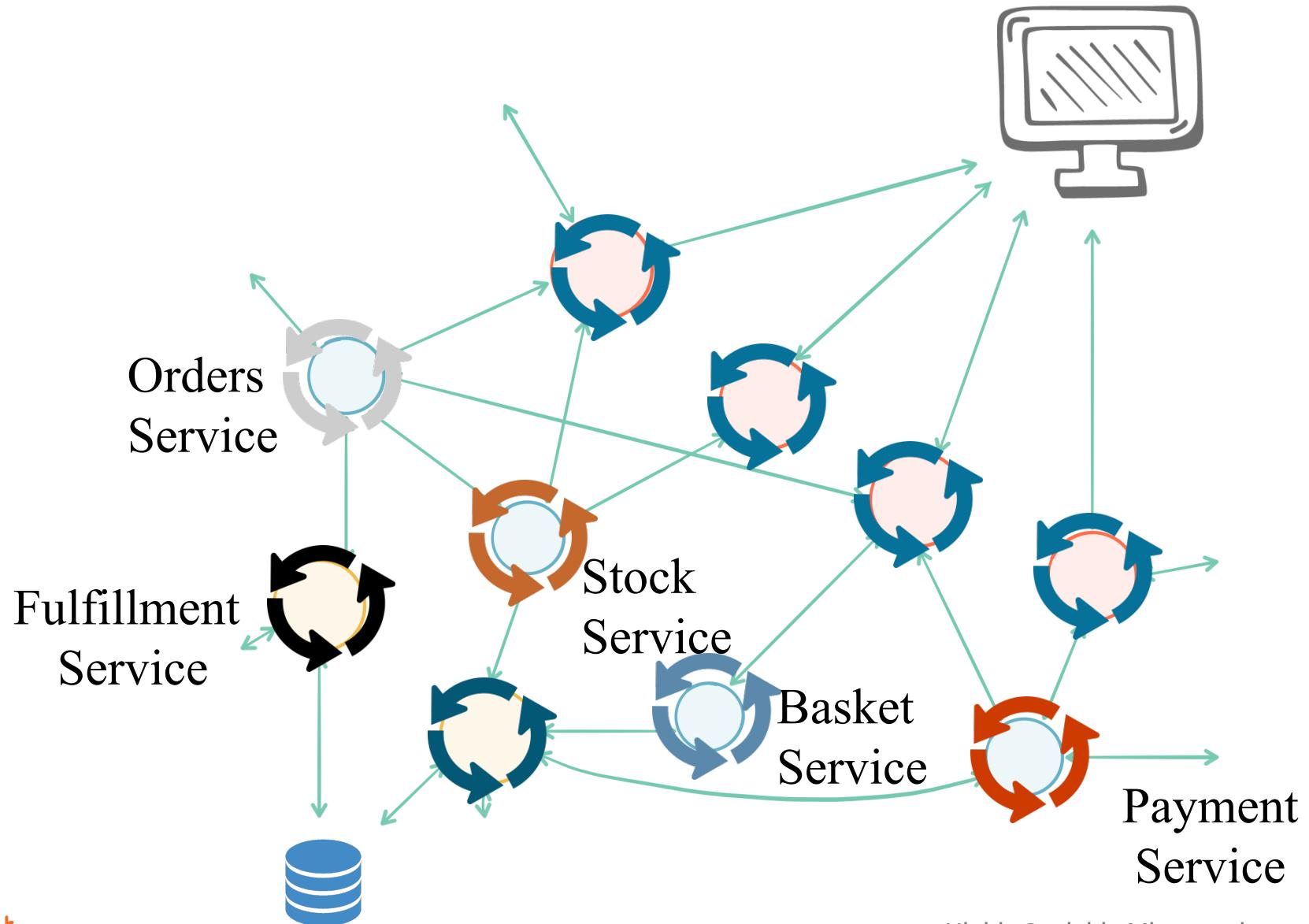
---



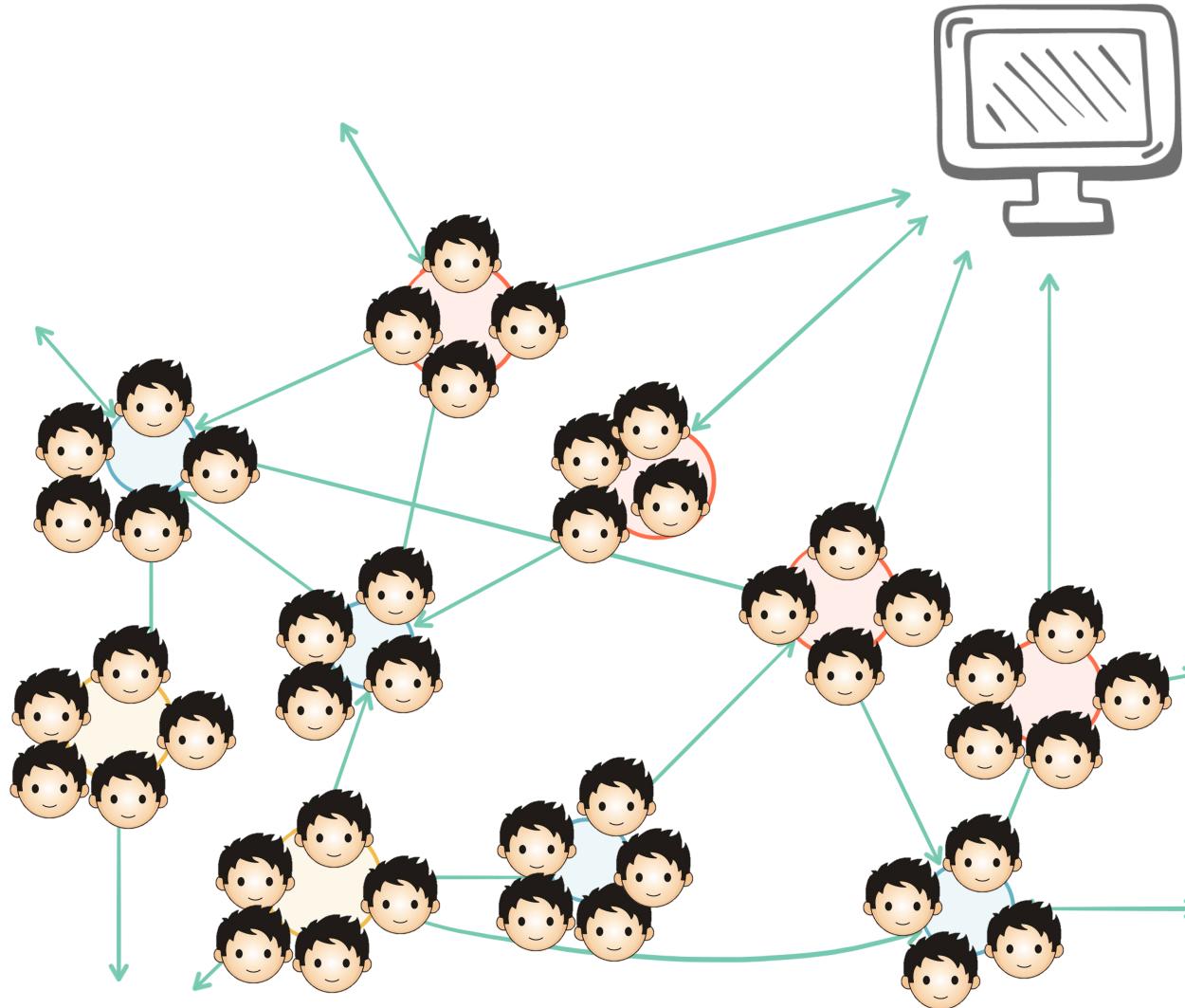
# Microservices



# Microservices are Independently Deployable

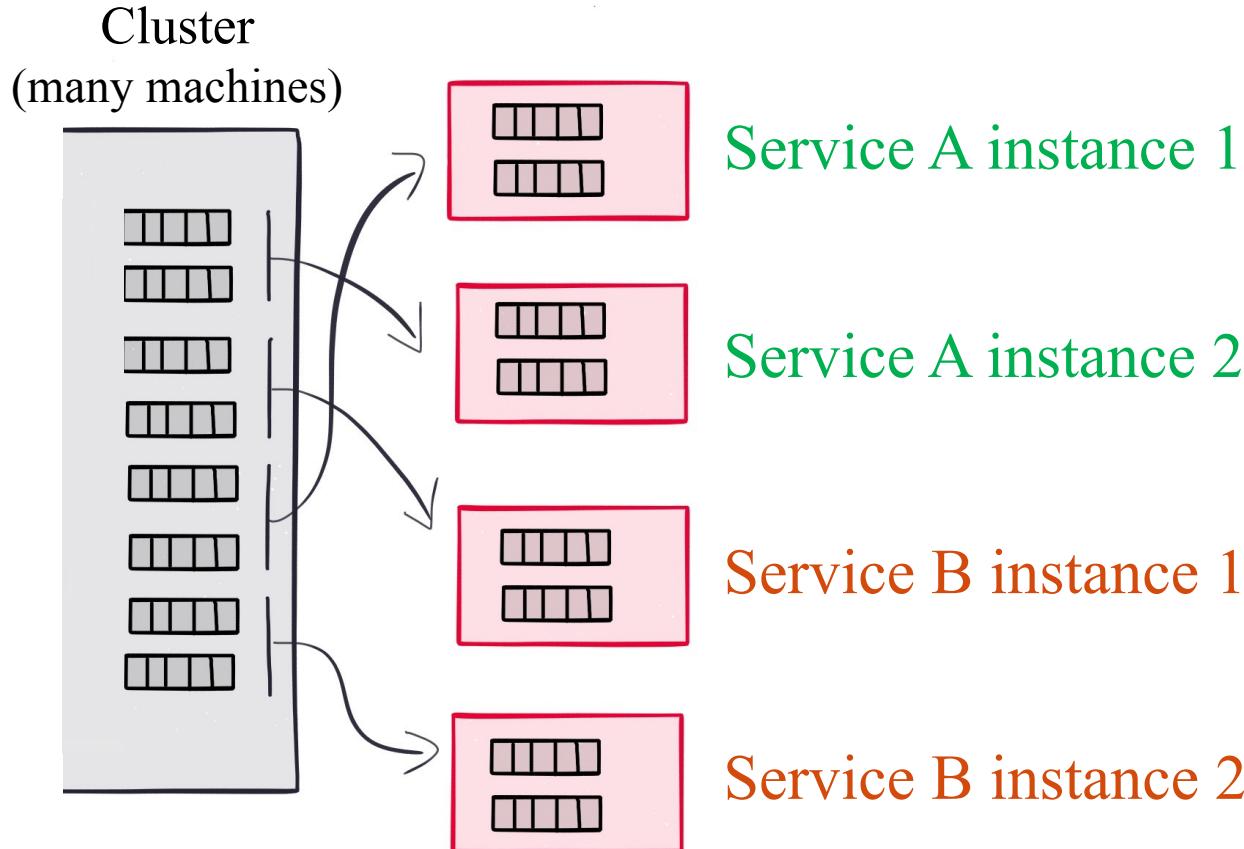


# Scale in People Terms



# Scale in Infrastructure Terms

---





## How do we get there?

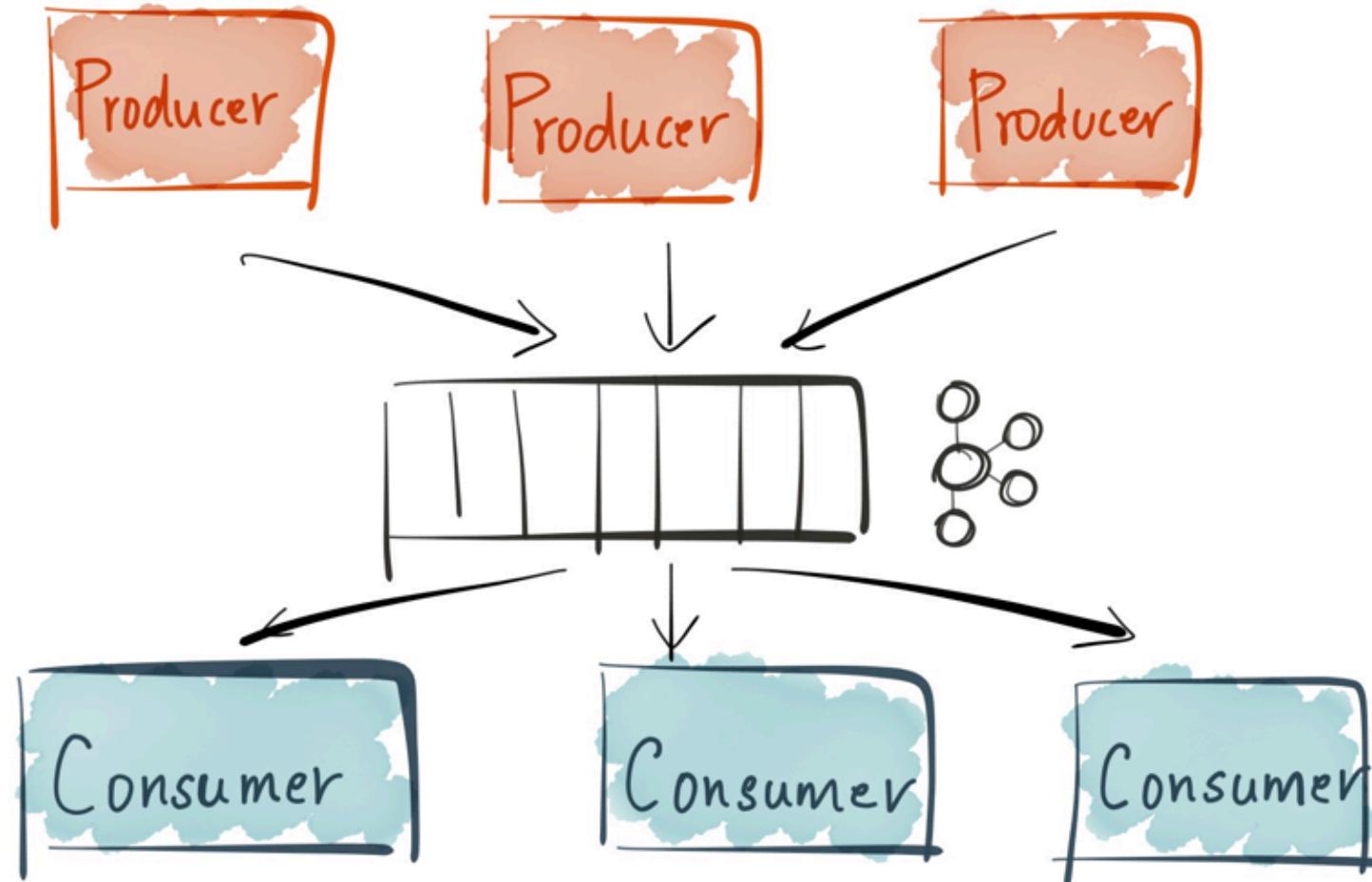
- Loose Coupling
- Data Enabled
- Event Driven
- Operational Transparency

# Agenda

---

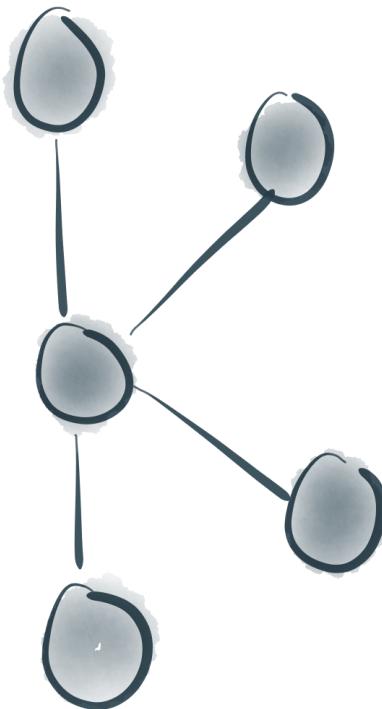
- 1) Scalable Microservices
- 2) Apache Kafka and Confluent Platform**
- 3) Kafka Streams
- 4) Scalable Microservices with Kafka and DC/OS
- 5) Use Case - Scalable Flight Prediction

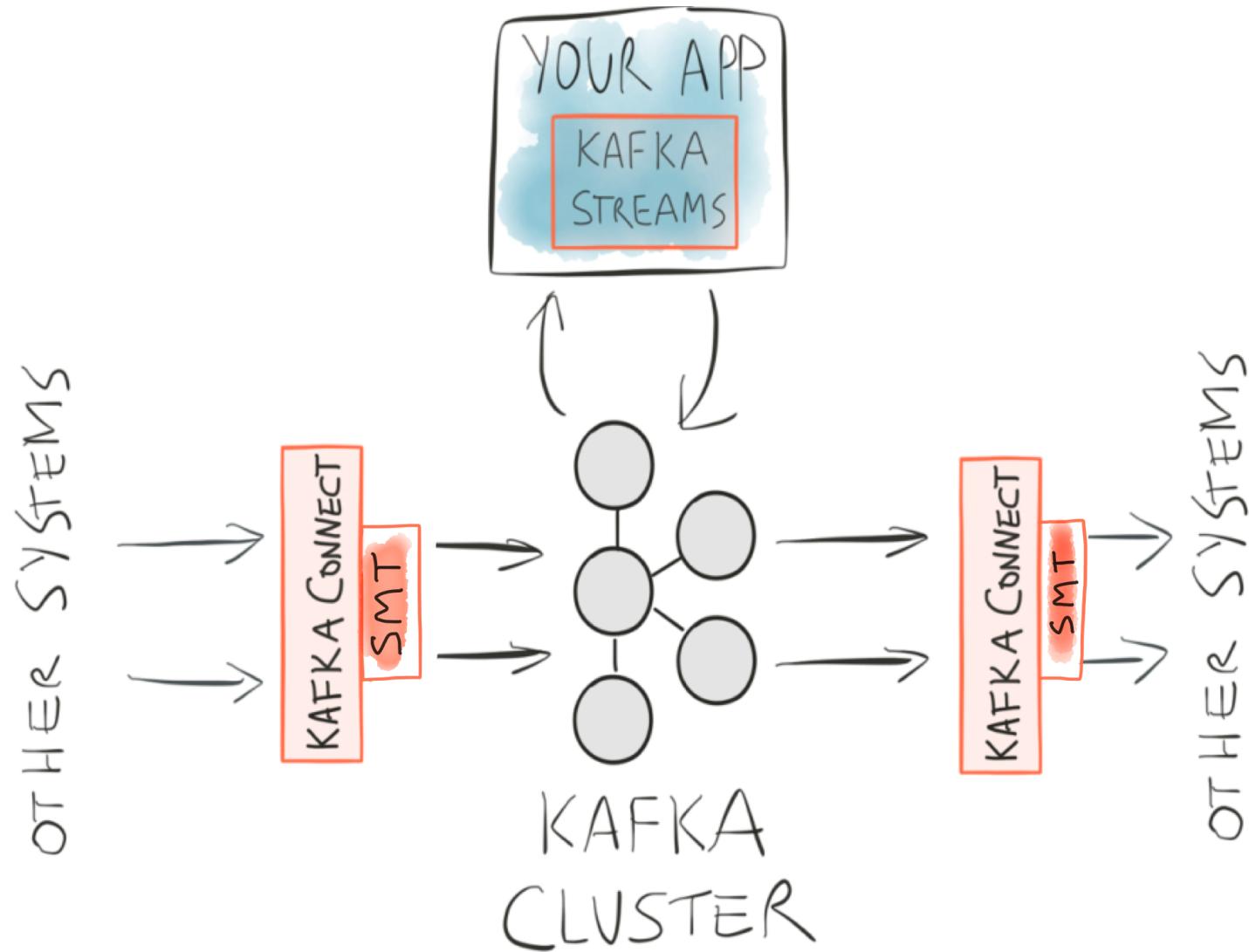
# Apache Kafka – A Distributed, Scalable, Fault-Tolerant Commit Log



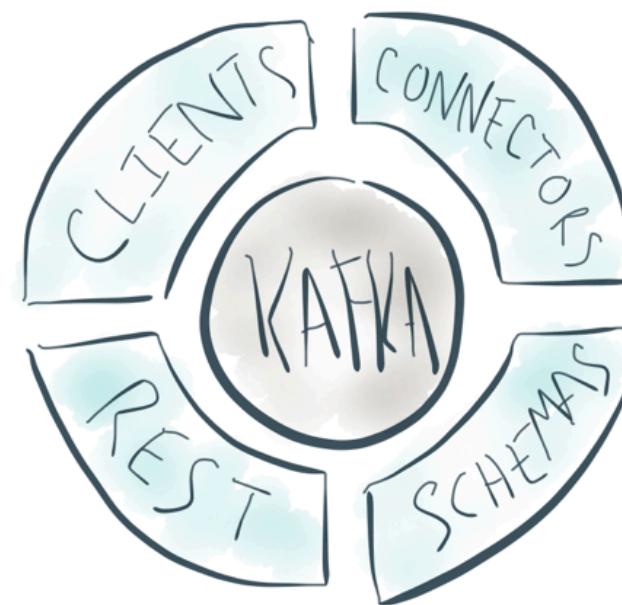
---

A MAJOR NEW ECOSYSTEM

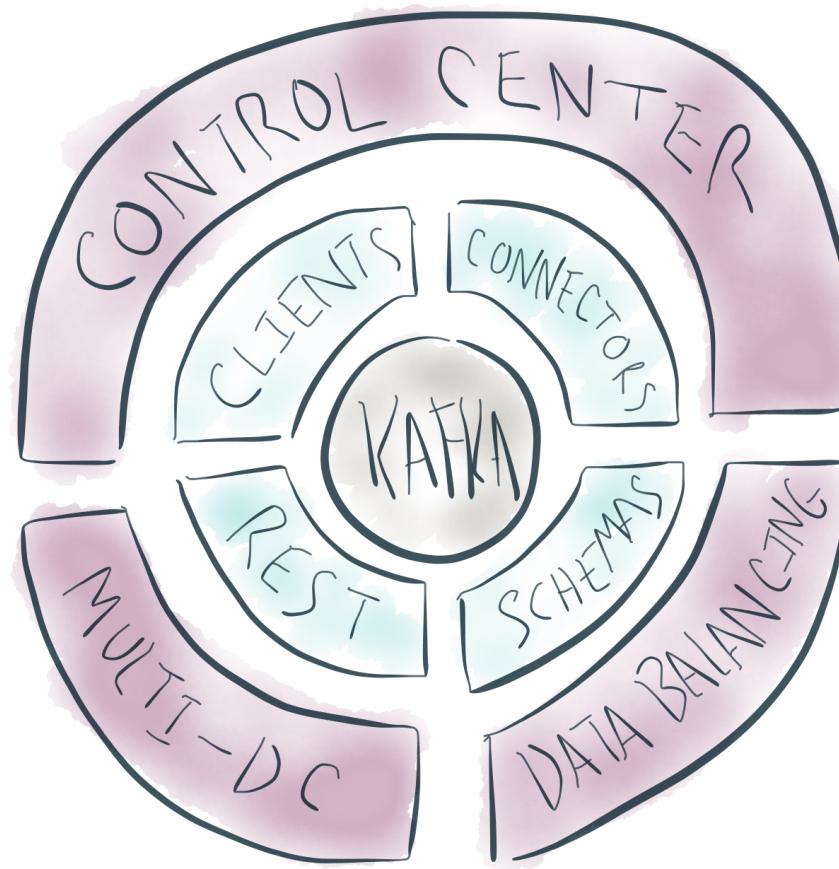




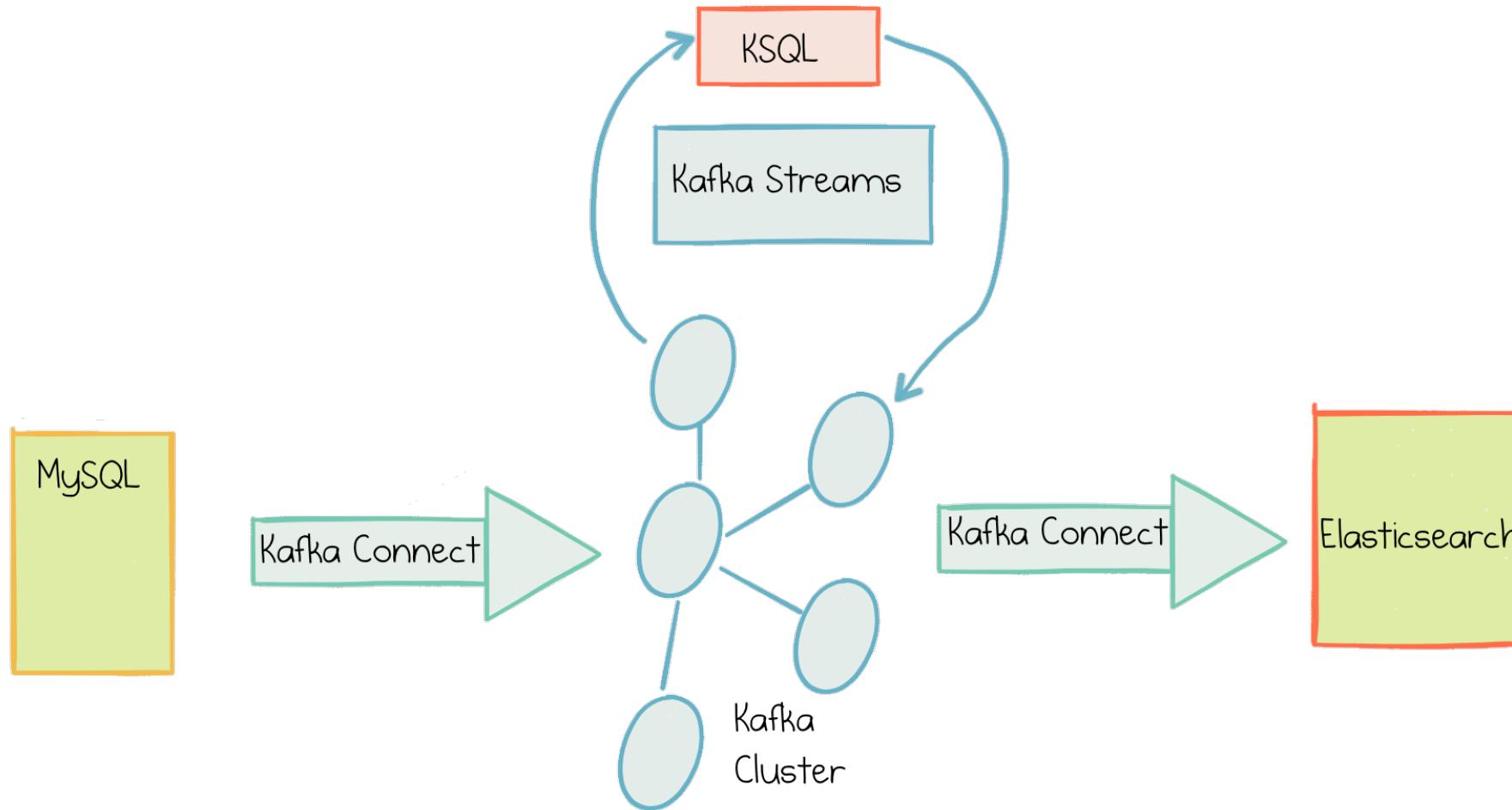
# CONFLUENT OPEN SOURCE



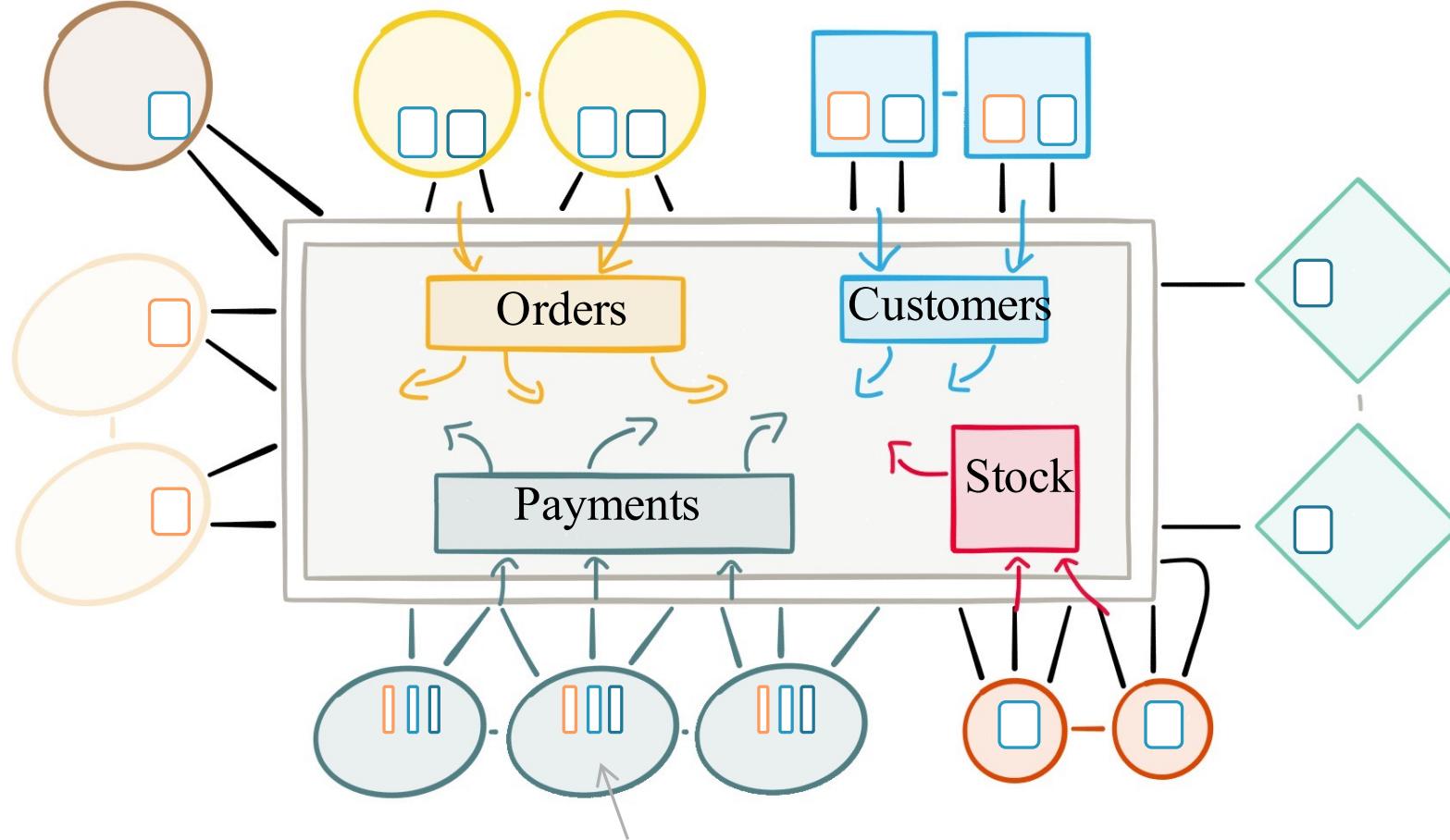
# CONFLUENT ENTERPRISE



# Apache Kafka – A Streaming Platform



# Kafka as Single, Shared Source of Truth for Microservices



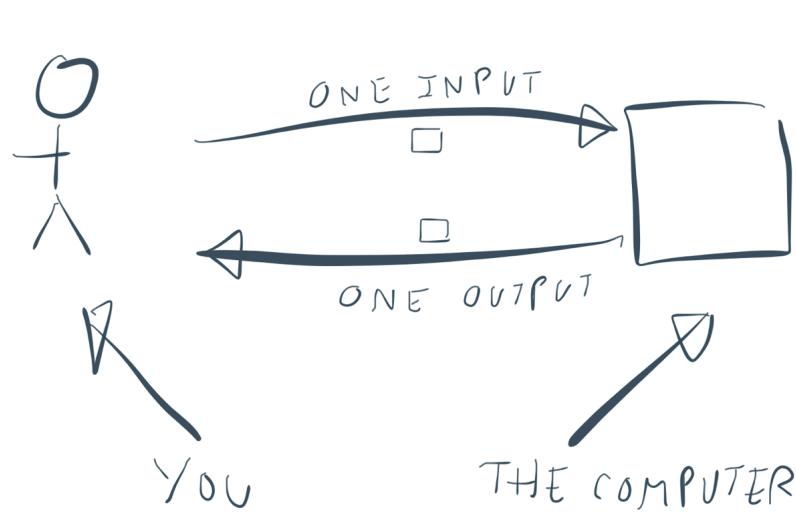
# Agenda

---

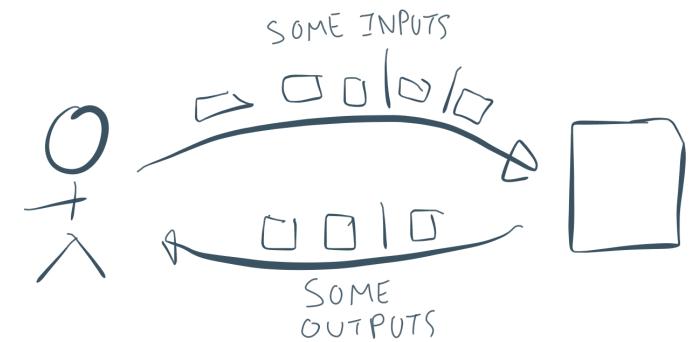
- 1) Scalable Microservices
- 2) Apache Kafka and Confluent Platform
- 3) Kafka Streams**
- 4) Scalable Microservices with Kafka and DC/OS
- 5) Use Case - Scalable Flight Prediction

# Stream Processing

REQUEST / RESPONSE



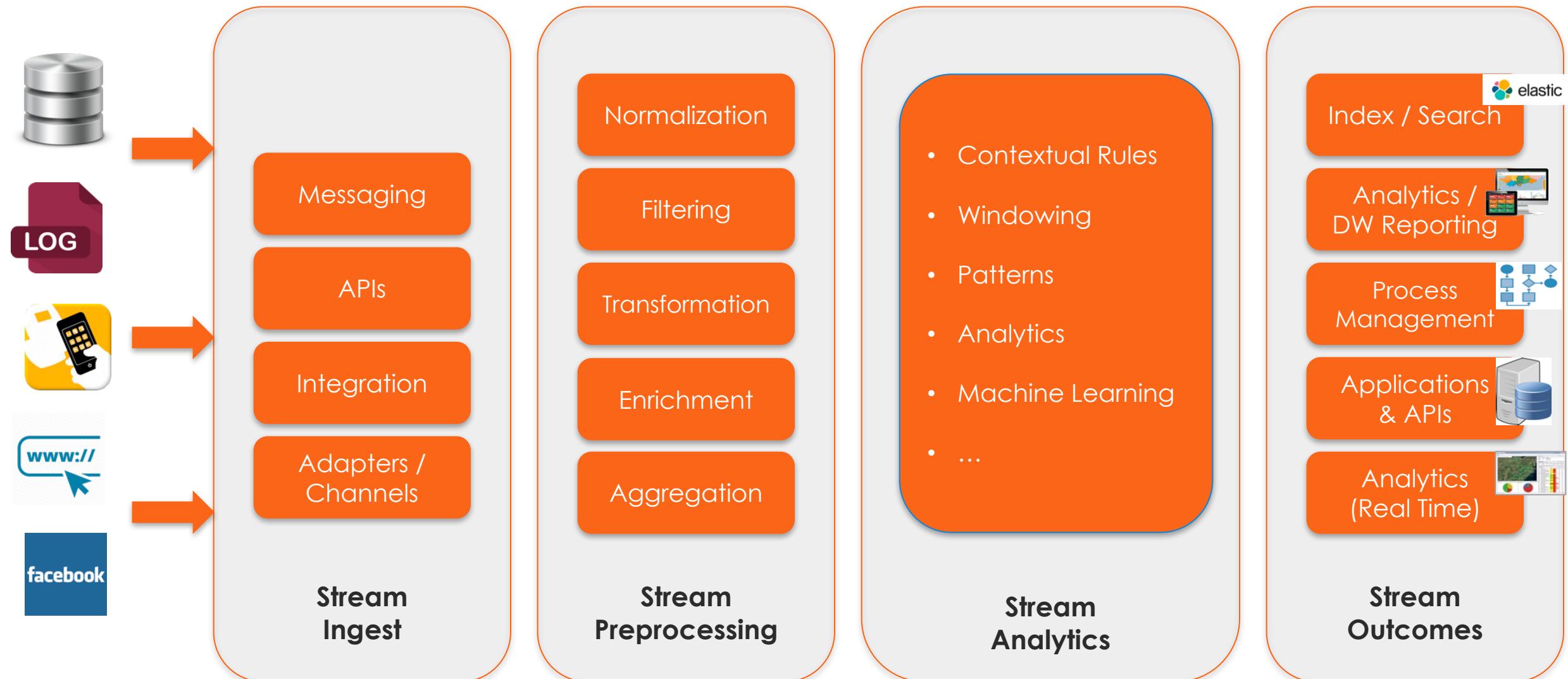
STREAM PROCESSING



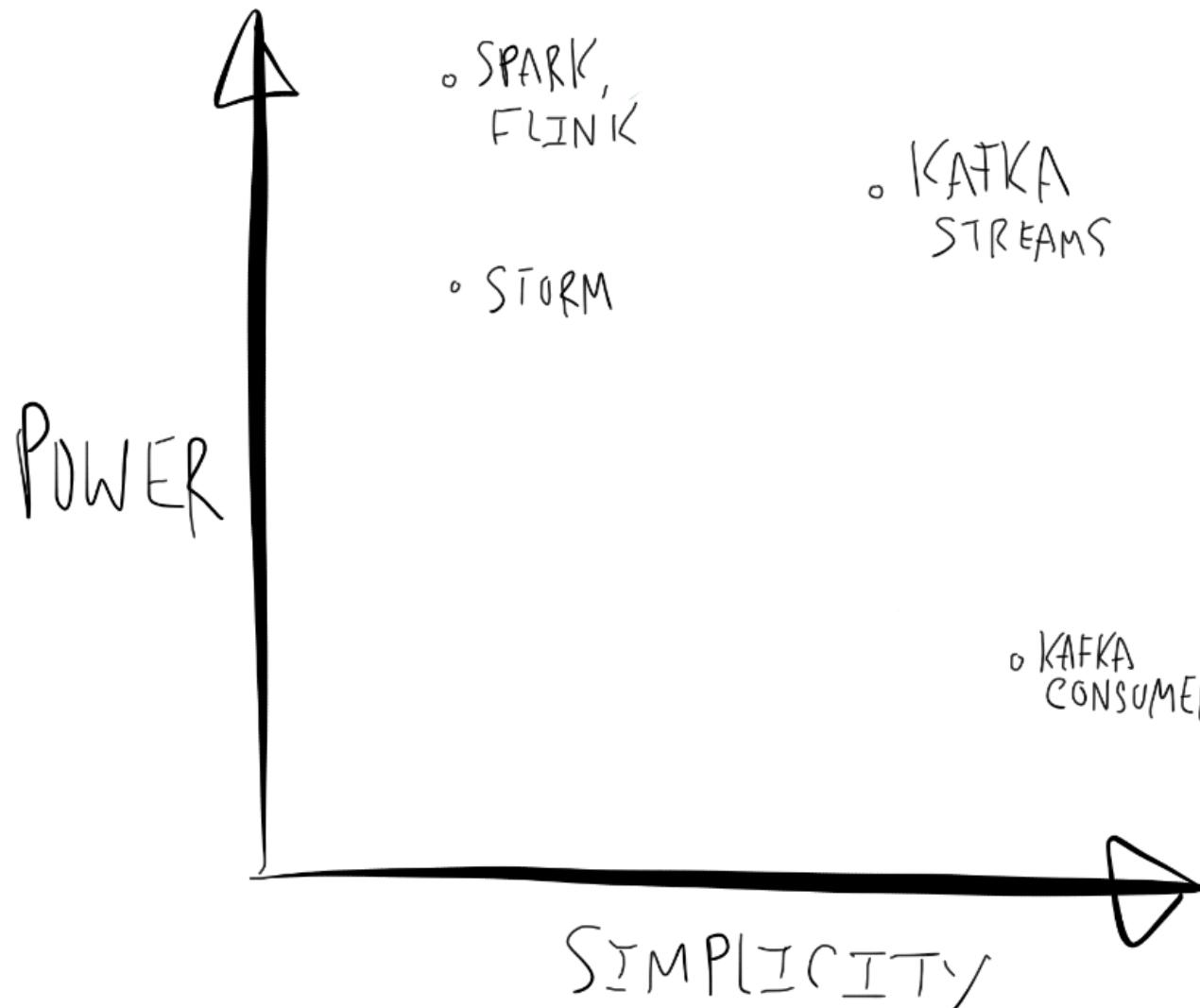
## Data at Rest

## Data in Motion

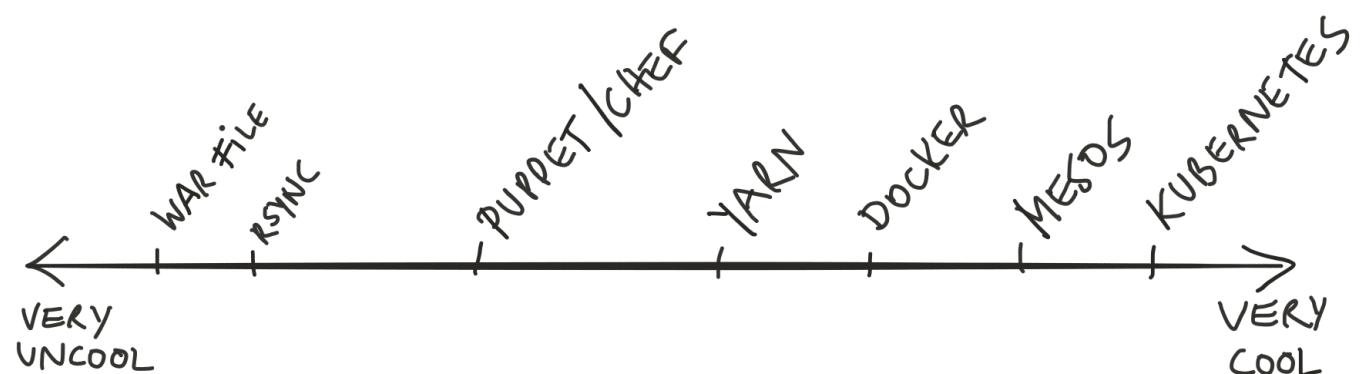
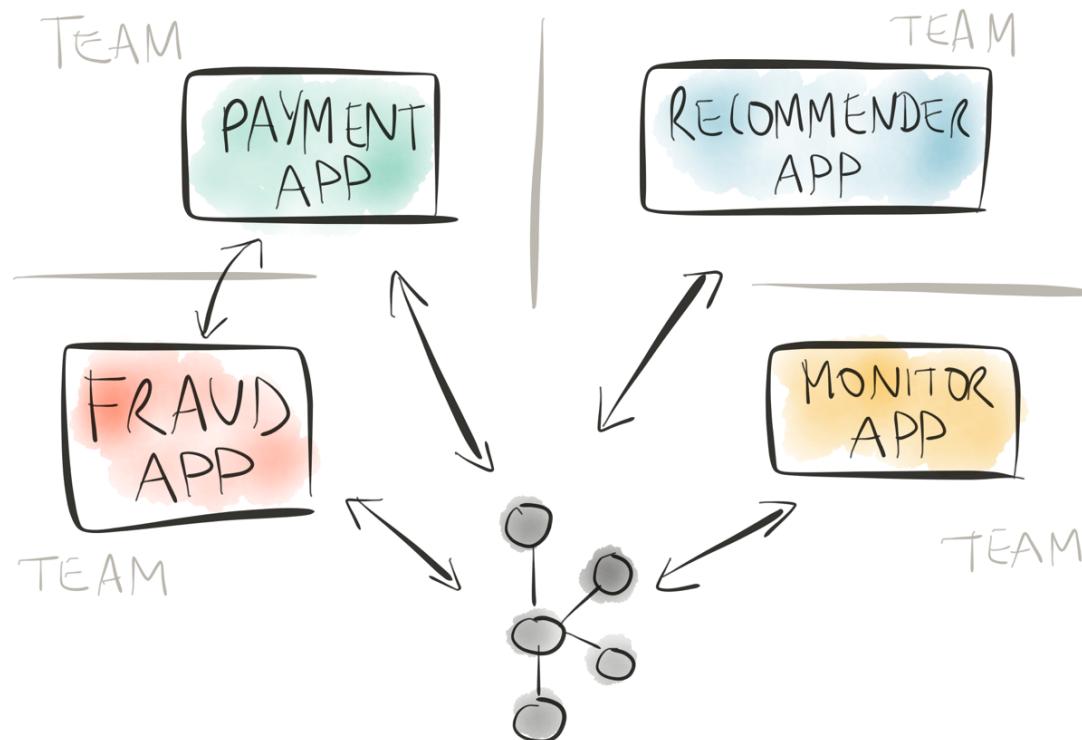
# Stream Processing Pipeline



# When to use Kafka Streams for Stream Processing?



# Lightweight, Independent, Scalable Streaming Microservices



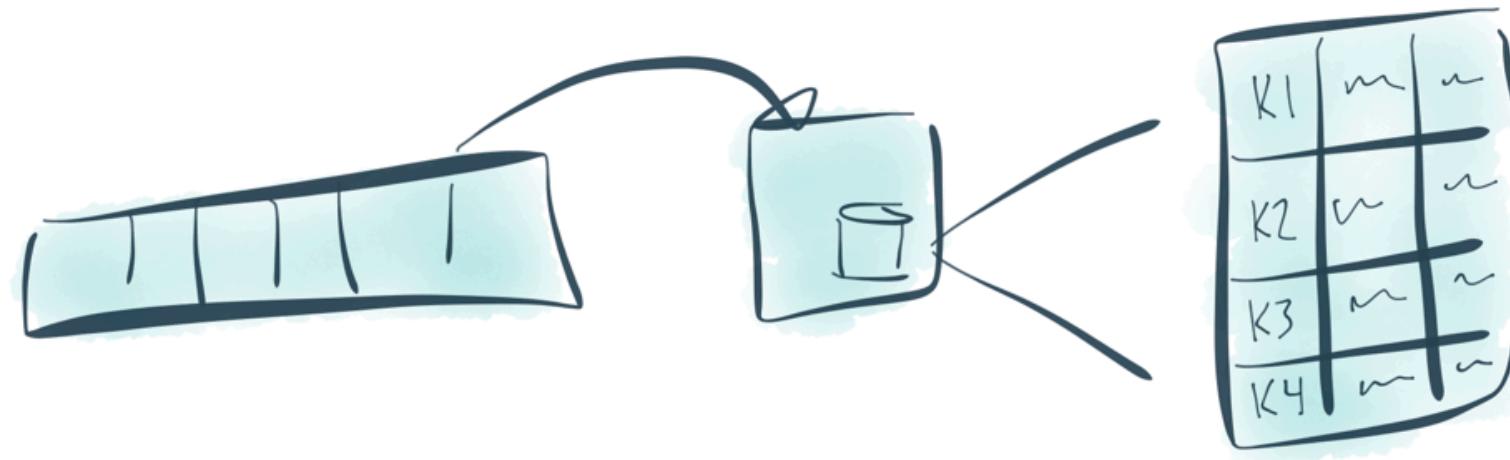
# KAFKA STREAMS

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

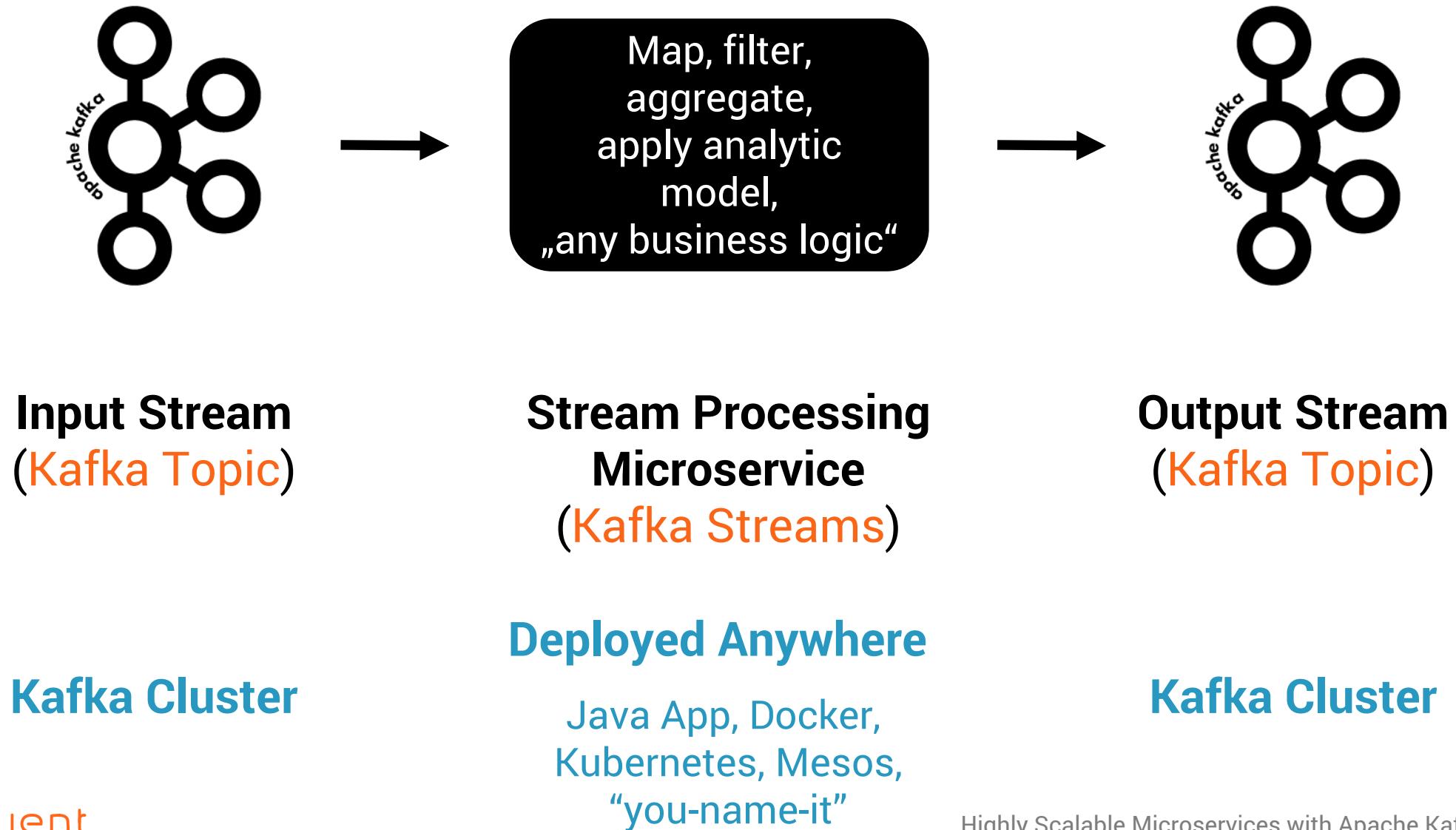
# KEY OPERATIONS

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

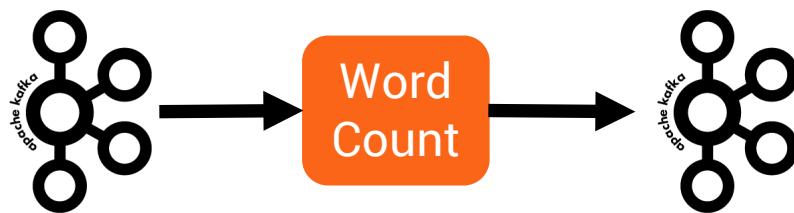
# UNIFY TABLES & STREAMS



# Kafka Streams (shipped with Apache Kafka)



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



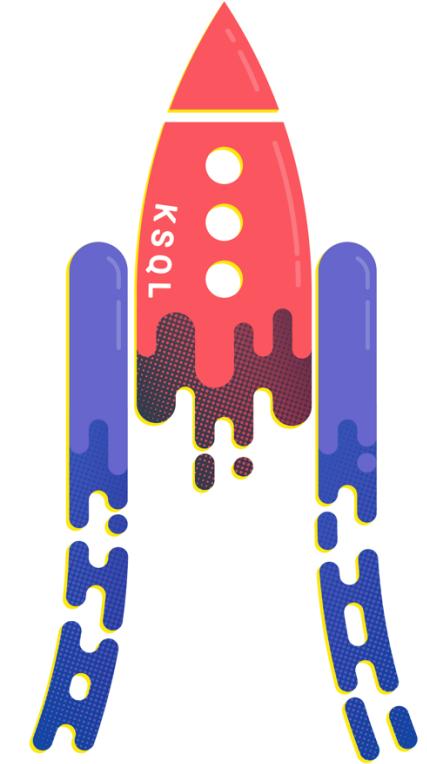
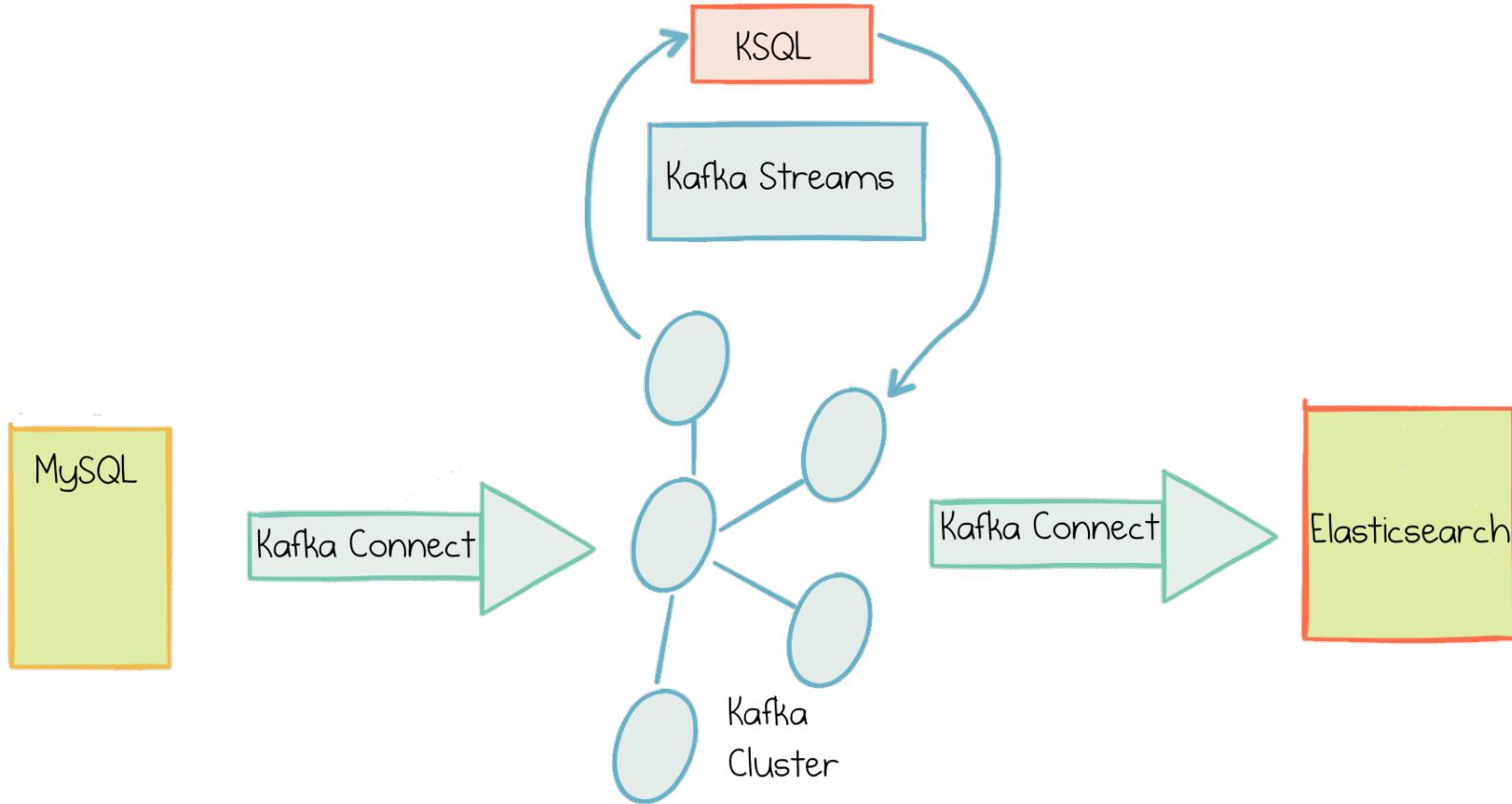
```
1 public static void main(final String[] args) throws Exception {  
2     Properties config = new Properties();  
3     config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-example");  
4     config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092");  
5     config.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, 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

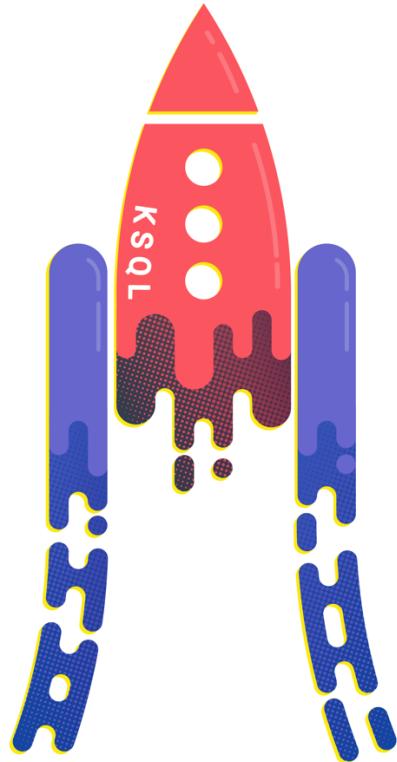
Define processing  
(here: WordCount)

Start processing

# KSQL – An Open Source Streaming SQL Engine for Apache Kafka



# Highly Scalable KSQL Streaming Microservices



```
CREATE STREAM vip_actions AS  
SELECT userid, page, action FROM clickstream c  
LEFT JOIN users u ON c.userid = u.user_id  
WHERE u.level = 'Platinum';
```

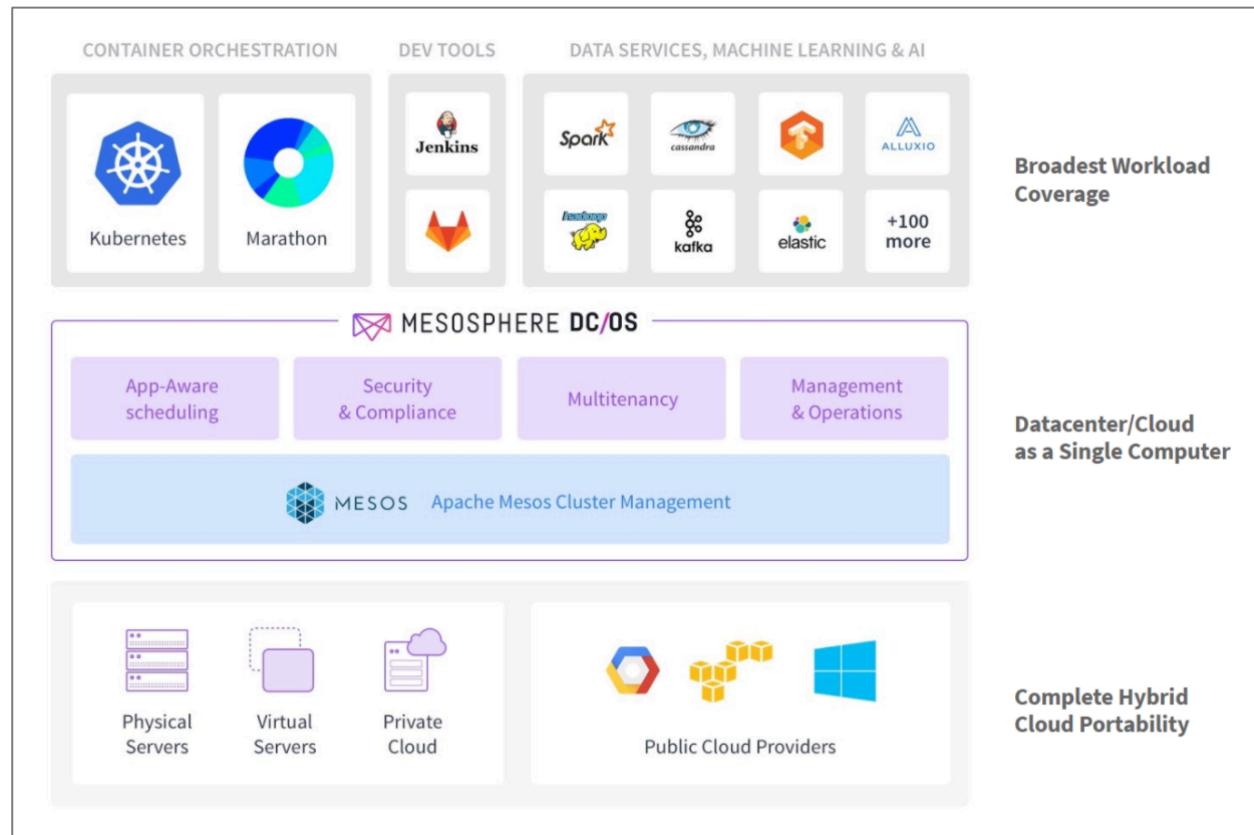
```
CREATE TABLE possible_fraud AS  
SELECT card_number, count(*)  
FROM authorization_attempts  
WINDOW TUMBLING (SIZE 5 MINUTES)  
GROUP BY card_number  
HAVING count(*) > 3;
```

# Agenda

---

- 1) Scalable Microservices
- 2) Apache Kafka and Confluent Platform
- 3) Kafka Streams
- 4) Scalable Microservices with Kafka and DC/OS**
- 5) Use Case - Scalable Flight Prediction

# Apache Mesos and DC/OS



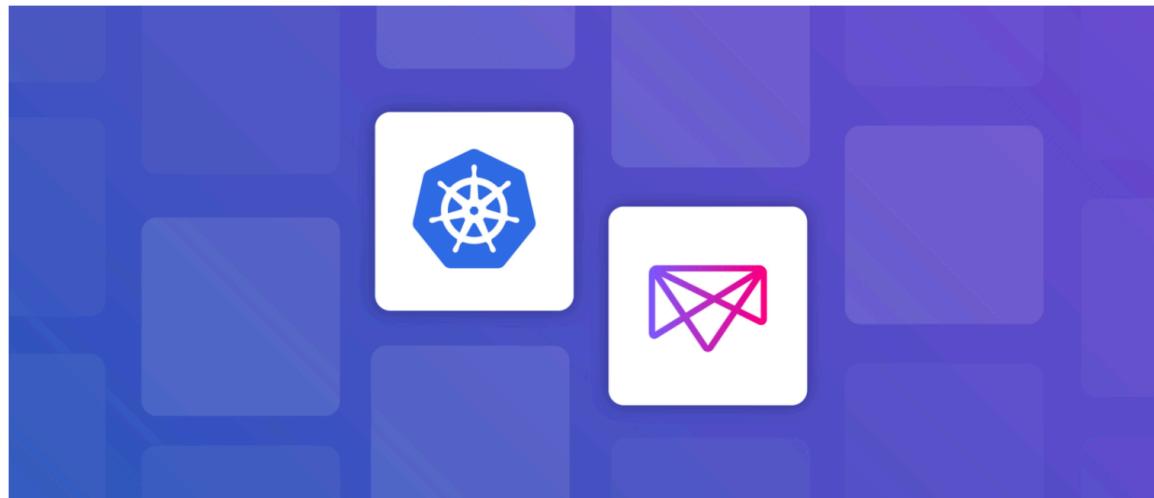
„... bring together all of the tools needed to operate data-intensive modern applications at scale such as container orchestration, distributed databases, message queues, data streaming and processing, machine learning, monitoring and management capabilities, etc.“

# Support for Kubernetes added in late 2017...

COMPANY, PARTNERS

## Announcing: Kubernetes on DC/OS

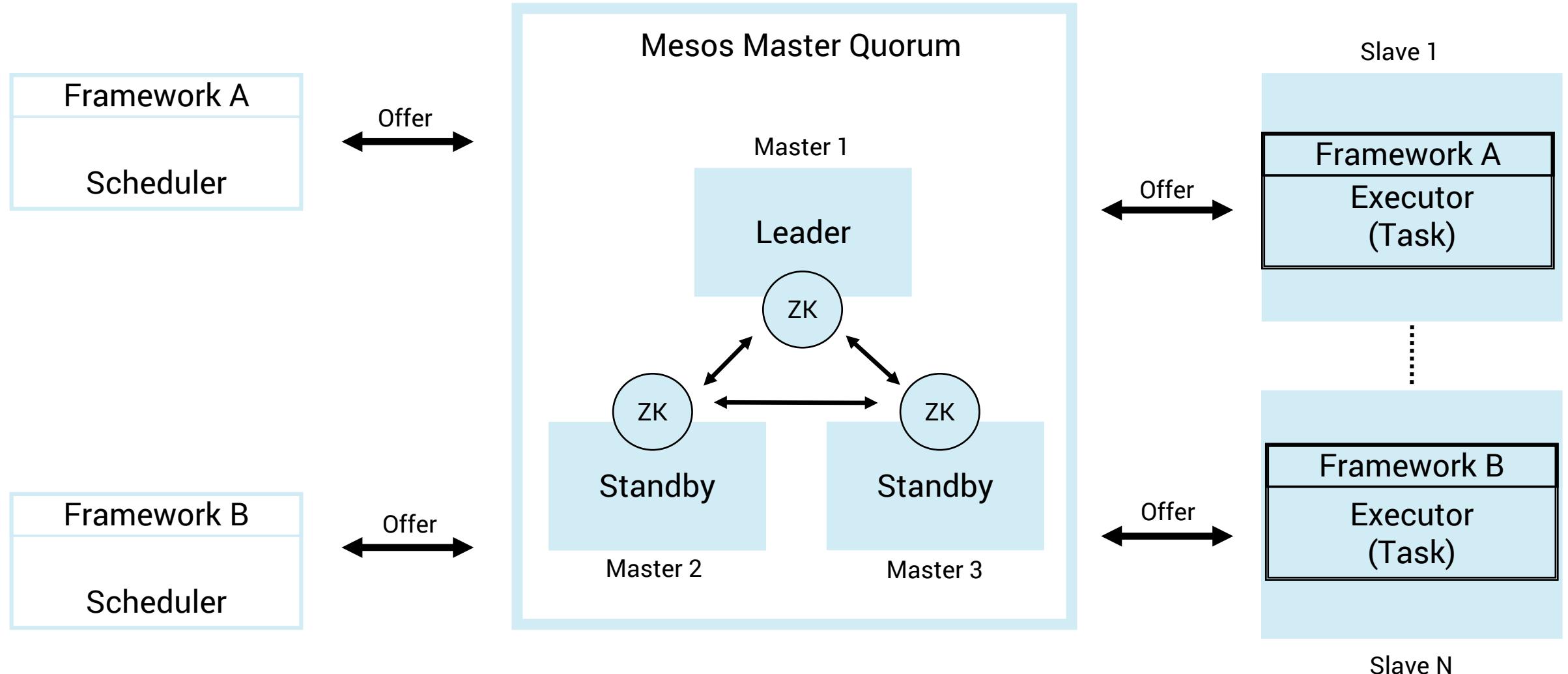
*Announcing the beta availability of Kubernetes on DC/OS. Learn why this is the first step towards making DC/OS the best place to run K8s.*



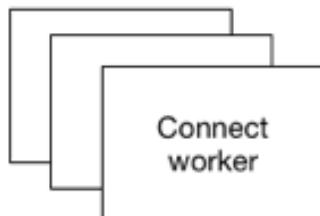
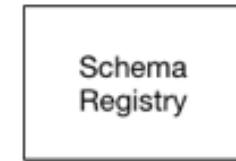
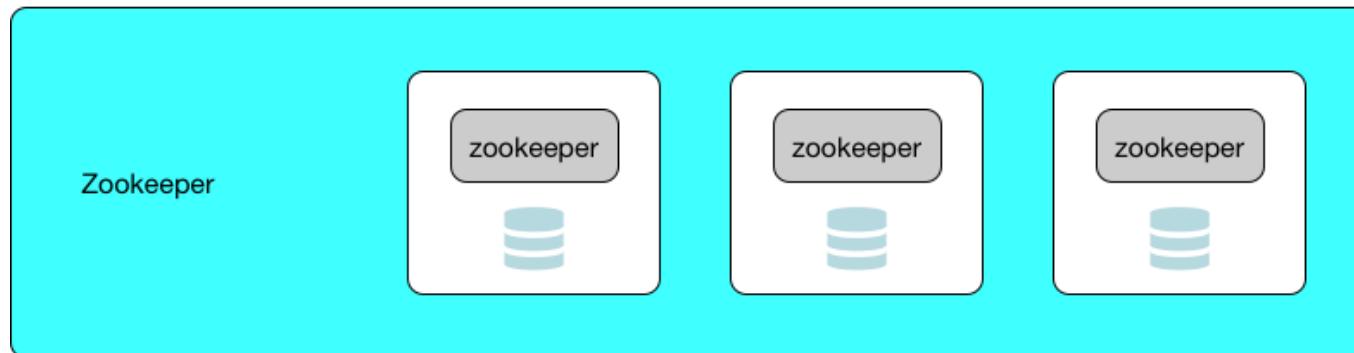
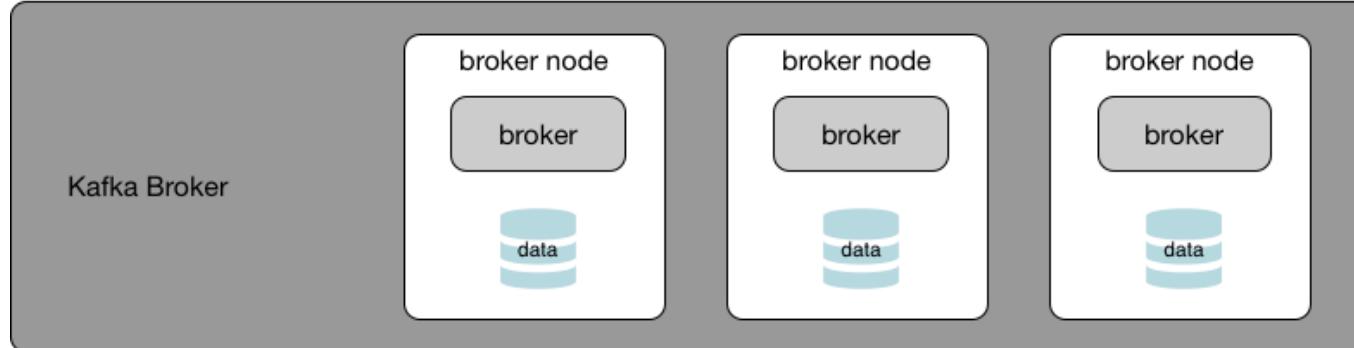
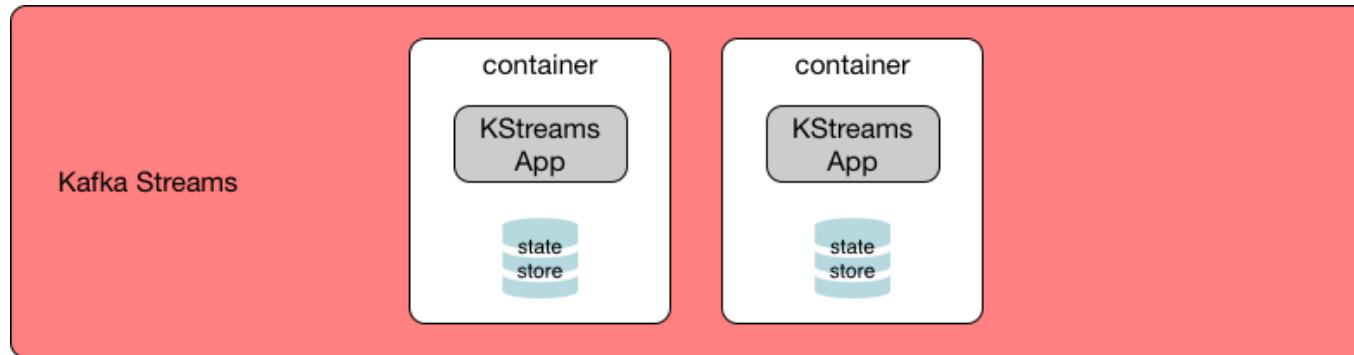
“Kubernetes on top of Mesos through DC/OS allows our customers to **deploy the popular container orchestrator on top of a powerful distributed systems platform**. With this unique architectural approach, Mesosphere DC/OS can provide an experience like the public cloud providers’ container engines within our customers’ data centers or across hybrid cloud.”

<https://mesosphere.com/blog/kubernetes-dcos/>

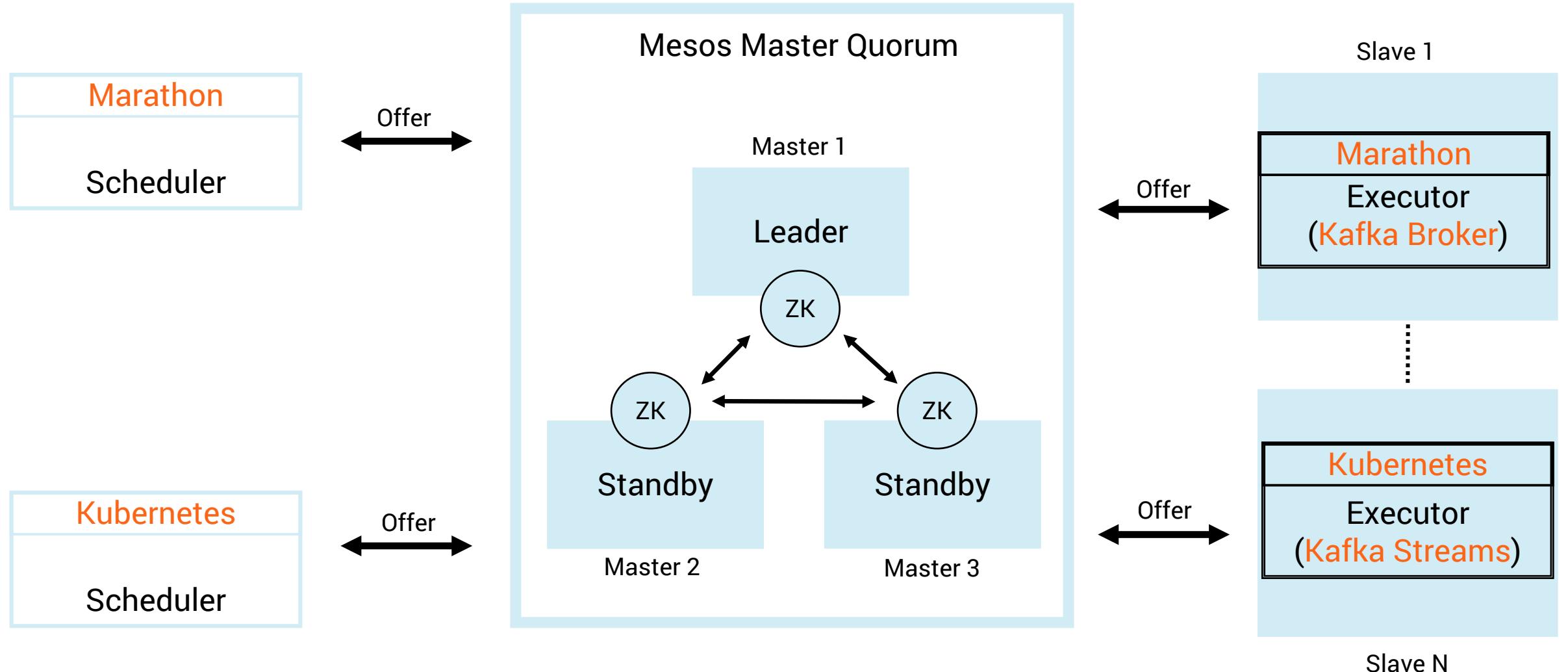
# Mesos Architecture



# 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) Use Case - Scalable Flight Prediction**

# Scenario for Highly Scalable Microservices

---

Use Case:

Airline Flight Delay Prediction

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

Technologies:

DC/OS

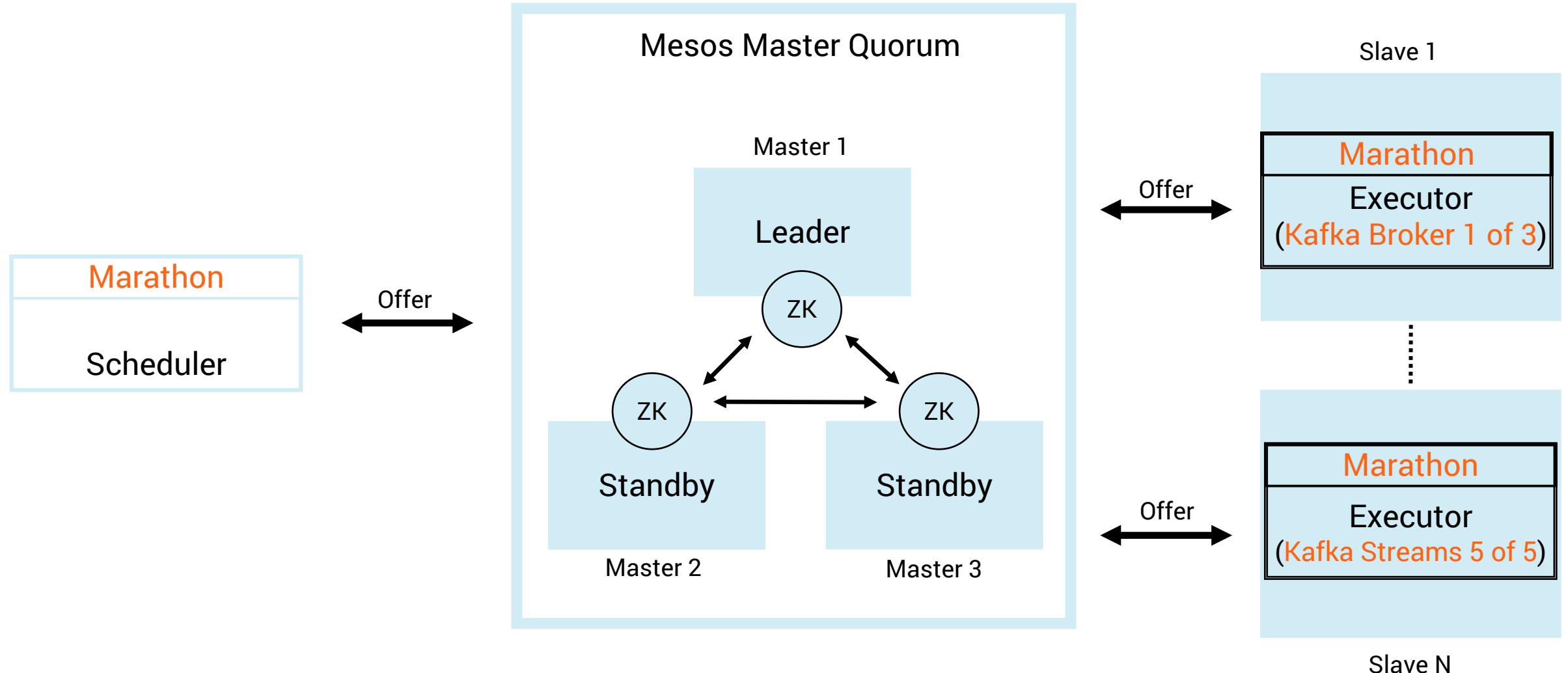
Kafka Broker

Kafka Streams

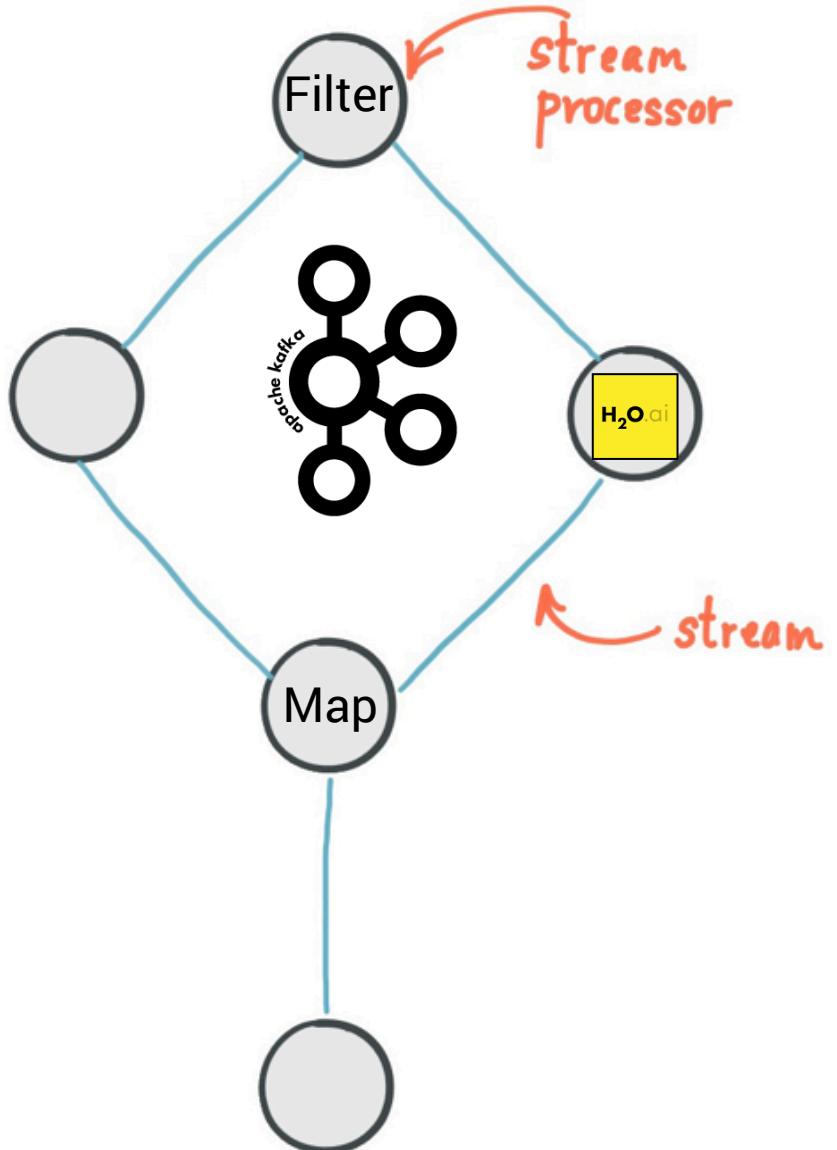
H2O.ai



# 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,YES,YES

        if (value != null && !value.equals("")) {
            System.out.println("#####"+value+"#####");
            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]);
            StreamModel<Prediction> 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

## DC/OS stable AWS CloudFormation

For more information on creating a DC/OS cluster, see the [DC/OS AWS documentation](#).

Region Name	Region Code	Single Master	HA: Three Master
US West (N. California)	us-west-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
US West (Oregon)	us-west-2	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
US East (N. Virginia)	us-east-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
South America (Sao Paulo)	sa-east-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
EU (Ireland)	eu-west-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
EU (Frankfurt)	eu-central-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
Asia Pacific (Tokyo)	ap-northeast-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
Asia Pacific (Singapore)	ap-southeast-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
Asia Pacific (Sydney)	ap-southeast-2	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>

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::LoadBalancer	CREATE_COMPLETE
ExhibitorS3Bucket	kai-kafka-mesos-exhibitors3bucket-fvl6h731h5i8	AWS::S3::Bucket	CREATE_COMPLETE
GatewayToInternet	kai-k-Gatew-N67EBATROY0G	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE
InboundNetworkAclEntry	kai-k-Inbou-15NK4D5HF8YIL	AWS::EC2::NetworkAclEntry	CREATE_COMPLETE
InternalMasterLoadBal...	kai-kafka-Internal-1MH6OA6BQ9G5D	AWS::ElasticLoadBalancing::LoadBalancer	CREATE_COMPLETE
InternetGateway	igw-e2f4288a	AWS::EC2::InternetGateway	CREATE_COMPLETE
LbSecurityGroup	sg-4cdf0c26	AWS::EC2::SecurityGroup	CREATE_COMPLETE
MasterInstanceProfile	kai-kafka-mesos-MasterInstanceProfile-J0UX3T5WNROK	AWS::IAM::InstanceProfile	CREATE_COMPLETE
MasterLaunchConfig	kai-kafka-mesos-MasterLaunchConfig-1MB0MEPZ4DD7O	AWS::AutoScaling::LaunchConfiguration	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::AutoScalingGroup	CREATE_COMPLETE
MasterToMasterIngress	MasterToMasterIngress	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE
MasterToPublicSlaveIn...	MasterToPublicSlaveIngress	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE

Filter: Active		By Stack Name
<input type="checkbox"/>	Stack Name	
<input type="checkbox"/>	Created Time	
<input type="checkbox"/>	Status	
<input type="checkbox"/>	Description	

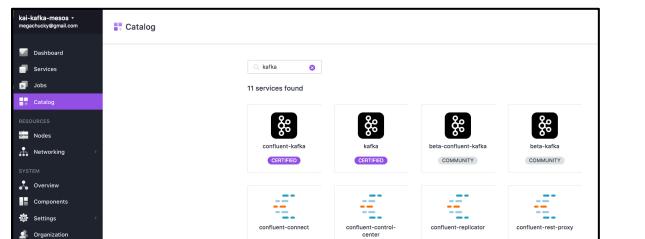
Showing 1 stack

Stack Name	Created Time	Status	Description
<input type="checkbox"/> kai-kafka-mesos	2017-10-16 08:07:49 UTC+0200	CREATE_COMPLETE	DC/OS AWS CloudFormation Template

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

Highly Scalable Microservices with Apache Kafka + Mesos | 41

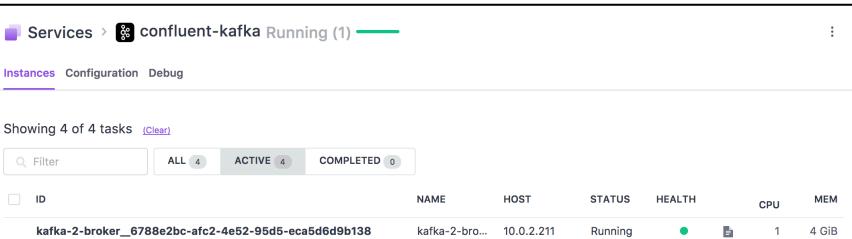
# Kafka Brokers on DC/OS



The screenshot shows the Kai Kafka Mesos dashboard. The left sidebar includes sections for Dashboard, Services, Jobs, Catalog, Resources (Notes, Networking), System (Overview, Components, Settings, Organization), and Catalog. The Catalog section is active, displaying a search bar with 'kafka' and a list of 11 services found. The services are categorized as Certified or Community, with icons representing their type (e.g., Kafka, Connect, Control Center). The services listed are: kafka, confluent-kafka, beta-confluent-kafka, beta-kafka, confluent-connect, confluent-control-center, confluent-replicator, confluent-rest-proxy, confluent-schema-registry, kafka-manager, and pörön-kafka.



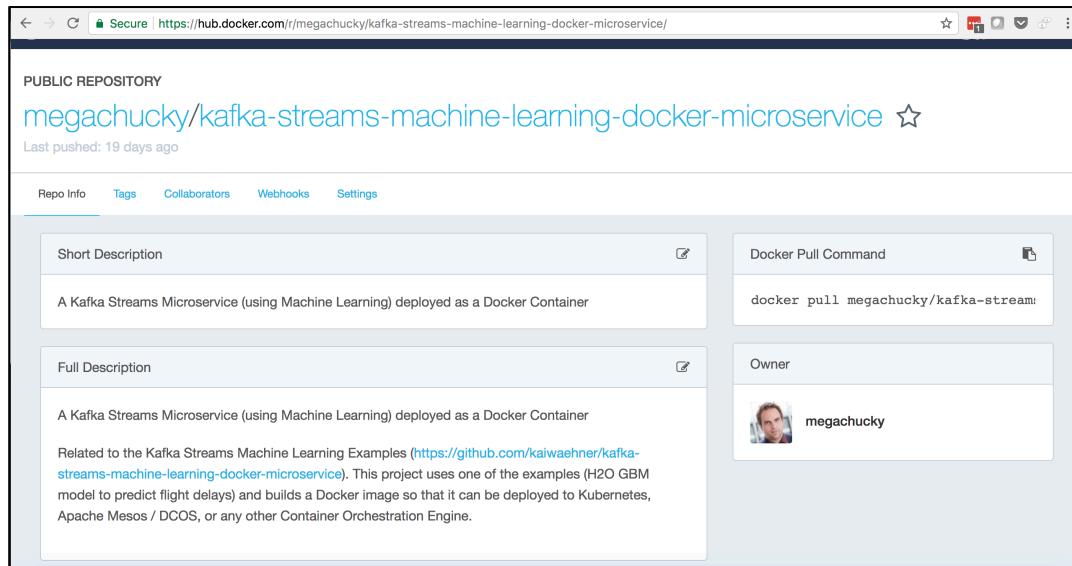
This screenshot shows the detailed view for the 'confluent-kafka' service. It features a large icon, a 'CERTIFIED' badge, and the version '2.0.1-3.3.0e'. Buttons for 'CONFIGURE' and 'DEPLOY' are present, along with a note about accepting terms and conditions. Below this is a 'Description' section for 'Apache Kafka by Confluent', documentation links, and a 'Pre-Install Notes' section. It also includes an 'Information' section with maintainer email and a 'Licenses' section with Apache License v2 details.



The screenshot shows the 'Services' page with the 'confluent-kafka' service running (1 instance). It lists four tasks: kafka-2-broker, kafka-1-broker, kafka-0-broker, and confluent-kafka. Each task is shown with its ID, host (kafka-2-bro..., kafka-1-bro..., kafka-0-bro..., confluent-ka...), status (Running), health (green), and resource usage (CPU: 1, Mem: 4 GiB or 1 GiB).

```
[~: 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 Streams Microservice



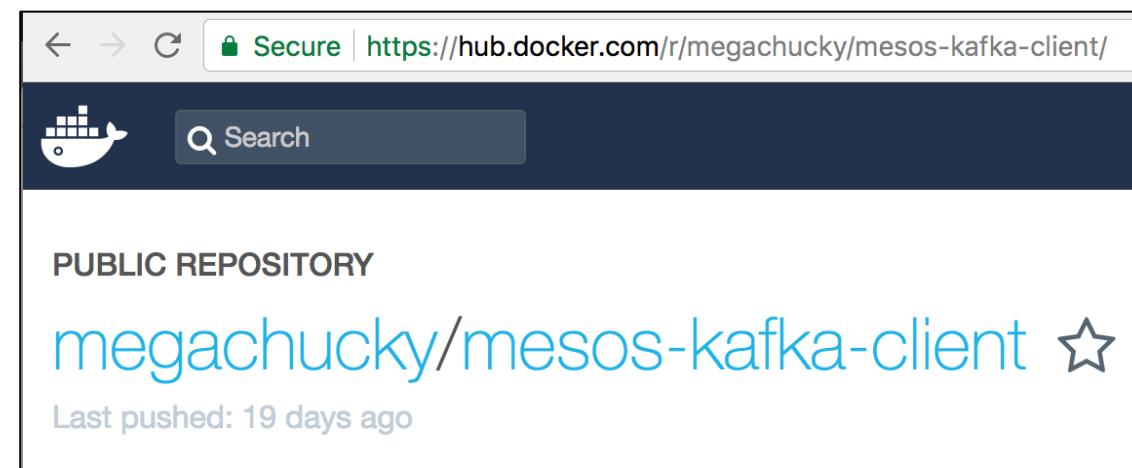
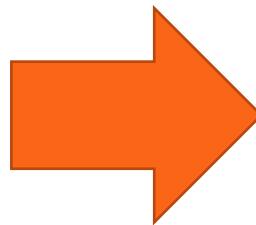
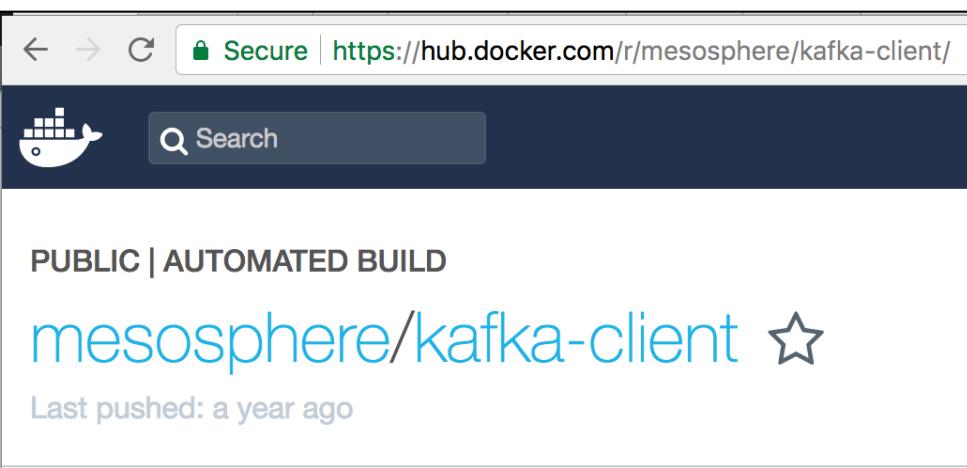
The screenshot shows the GitHub repository page for `kaiwaechner/kafka-streams-machine-learning-examples`. The description states: "This project contains examples which demonstrate how to deploy analytic models to mission-critical, scalable production environments leveraging Apache Kafka and its Streams API. Models are built with Python, H2O, TensorFlow, DeepLearning4 and other technologies." The repository has 45 commits, 1 branch, 0 releases, and 1 contributor. The last commit was 7 days ago.

```
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/kaiwaechner/kafka-streams-machine-learning-docker-microservice>

# 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
```

# Kafka Streams Microservice on DC/OS

Run a Service

Service

Networking

Volumes

Health Checks

Environment

**Service**

Configure your service below. Start by giving your service an ID.

SERVICE ID \*

INSTANCES

CONTAINER IMAGE \*

CPU \*

Memory (MiB) \*

COMMAND

A shell command for your container to execute.

**MORE SETTINGS**

**Container Runtime**  Docker Engine

The container runtime is responsible for running your service. We support the Mesos and Docker containerizers.

Docker Engine

Docker's container runtime. No support for multiple containers (Pods) or GPU resources.

Mesos Runtime

Universal Container Runtime (UCR) using native Mesos engine. Supports Docker file format, multiple containers (Pods) and GPU resources.

Services > kafka-streams Running (1) > kafka-streams

Details Files Logs

Search

ERROR (STDERR) OUTPUT (STDOUT)

```
INFO task [0_8] Initializing state stores (org.apache.kafka.streams.processor.internals.StreamTask:140)
INFO task [0_8] Initializing processor nodes of the topology (org.apache.kafka.streams.processor.internals.StreamTask:333)
INFO stream-thread [StreamThread-1] Creating active task 0_9 with assigned partitions [AirlinerInputTopic-9] (org.apache.kafka.streams.processor.internals.StreamThread:858)
INFO task [0_9] Initializing state stores (org.apache.kafka.streams.processor.internals.StreamTask:140)
INFO task [0_9] Initializing processor nodes of the topology (org.apache.kafka.streams.processor.internals.StreamTask:333)
INFO stream-thread [StreamThread-1] State transition from ASSIGNING_PARTITIONS to RUNNING. (org.apache.kafka.streams.processor.internals.StreamThread:163)
INFO stream-client [kafka-streams-h2o-gbm-example-20fbfce8-274d-4bee-9af5-4138f994f012] State transition from REBALANCING to RUNNING. (org.apache.kafka.streams.KafkaStreams:224)
INFO stream-thread [StreamThread-1] Committing all tasks because the commit interval 30000ms has elapsed (org.apache.kafka.streams.processor.internals.StreamThread:767)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_0 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_1 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_2 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_3 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_4 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_5 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_6 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_7 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_8 (org.apache.kafka.streams.processor.internals.StreamThread:805)
INFO stream-thread [StreamThread-1] Committing task StreamTask_0_9 (org.apache.kafka.streams.processor.internals.StreamThread:805)
#####
Flight Input: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
Label (dkr prediction) is flight departure delayed: YES
Class probabilities: 0.4319916897116479, 0.5680083102883521
#####
```

Services > kafka-streams Running (5)

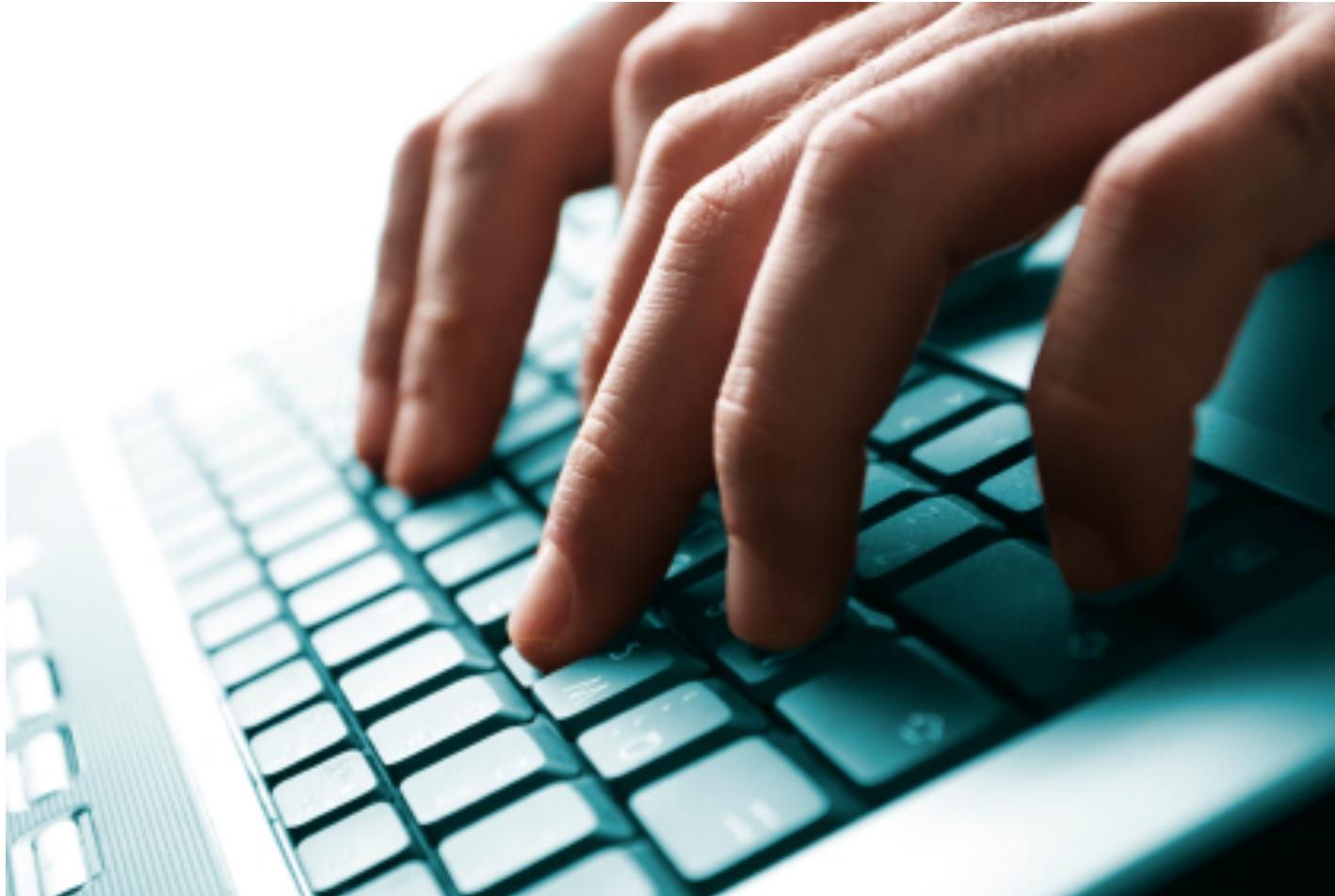
Instances Configuration Debug

Showing 5 of 38 tasks (Clear)

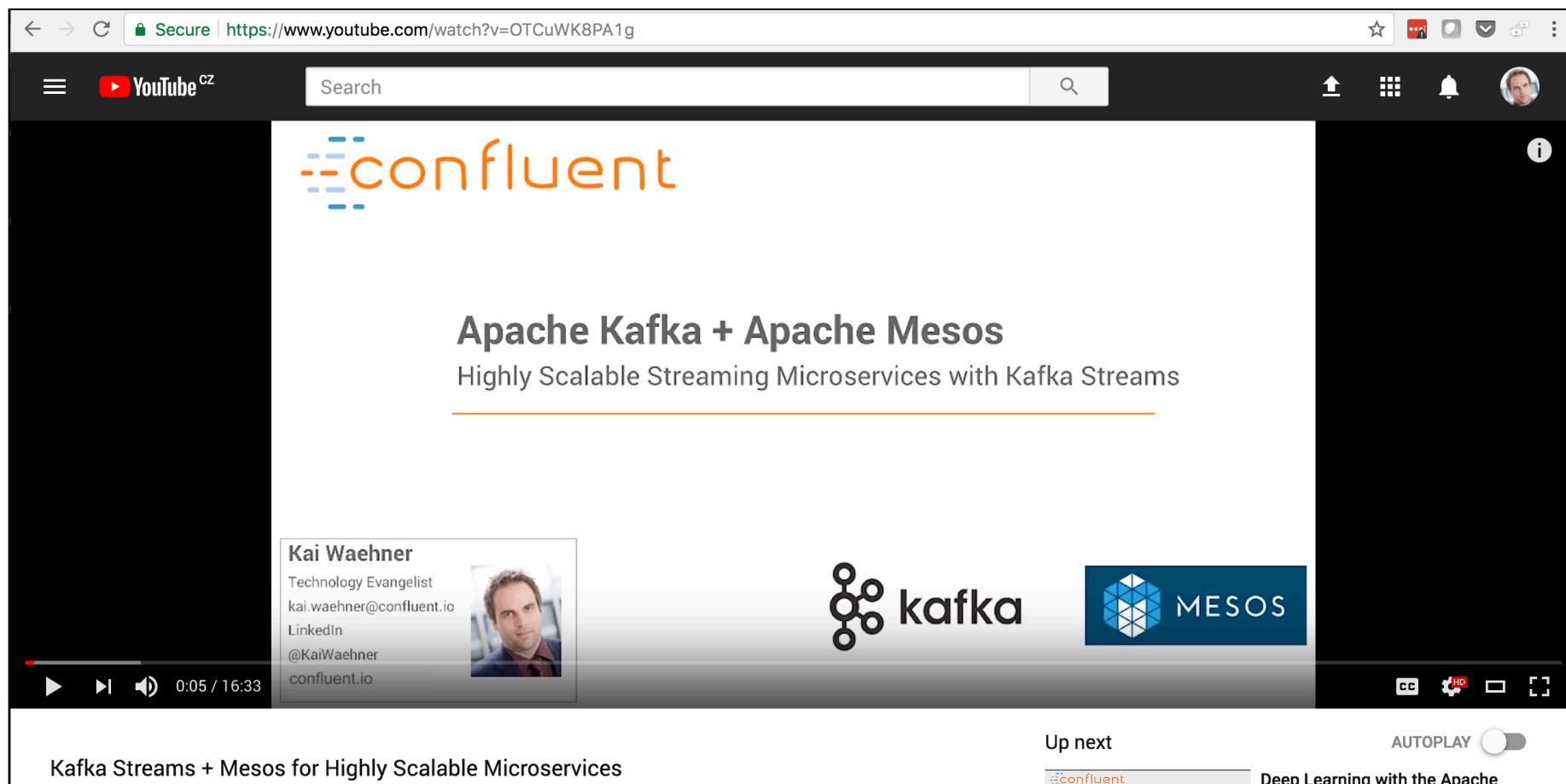
Filter	All 38	Active 5	Completed 33					
ID	Name	Host	Status	Health	CPU	Mem	Updated	Version
kafka-streams.470c70c7-b243-11e7-b6be...	kafka-streams	10.0.2.140	Running	●	2	1 GiB	16 seconds ago	10/16/2017, 9:26:08 AM
kafka-streams.470d3419-b243-11e7-b6be...	kafka-streams	10.0.2.140	Running	●	2	1 GiB	16 seconds ago	10/16/2017, 9:26:08 AM
kafka-streams.470c49b6-b243-11e7-b6be...	kafka-streams	10.0.1.40	Running	●	2	1 GiB	16 seconds ago	10/16/2017, 9:26:08 AM
kafka-streams.470cbee8-b243-11e7-b6be...	kafka-streams	10.0.2.118	Running	●	2	1 GiB	16 seconds ago	10/16/2017, 9:26:08 AM
kafka-streams.f313b4b5-b242-11e7-b6be...	kafka-streams	10.0.2.211	Running	●	2	1 GiB	3 minutes ago	10/16/2017, 9:23:47 AM

# Live Demo

---

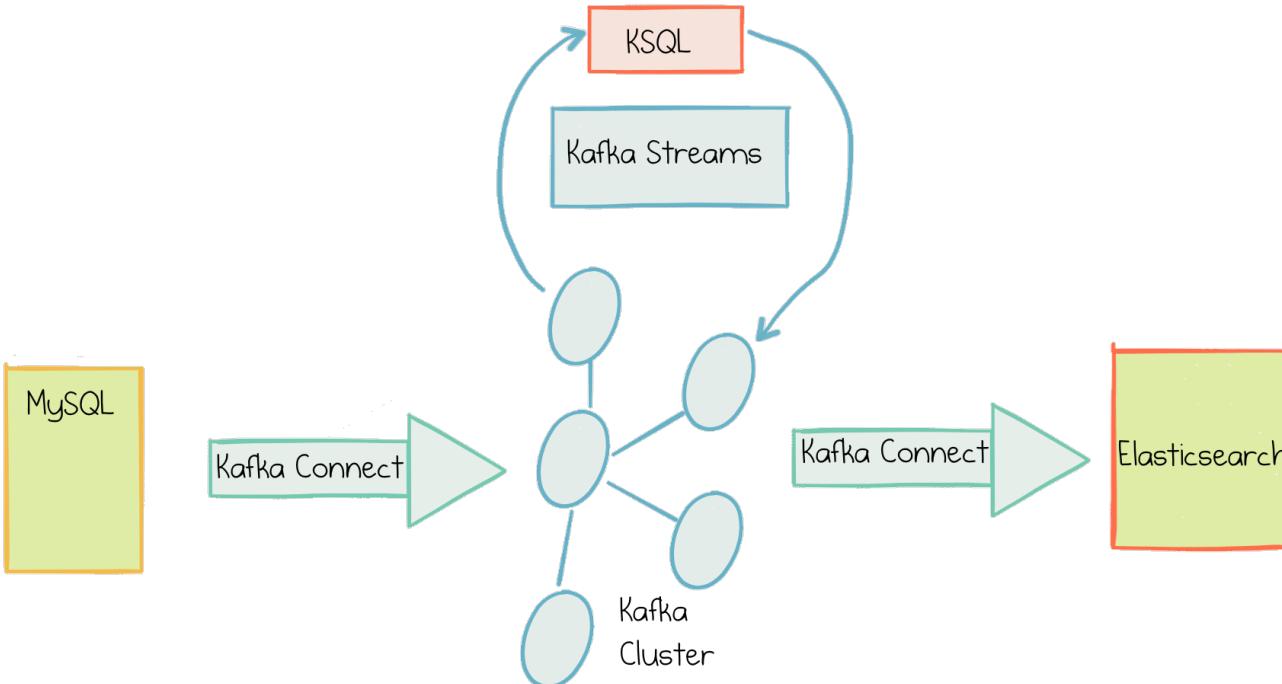
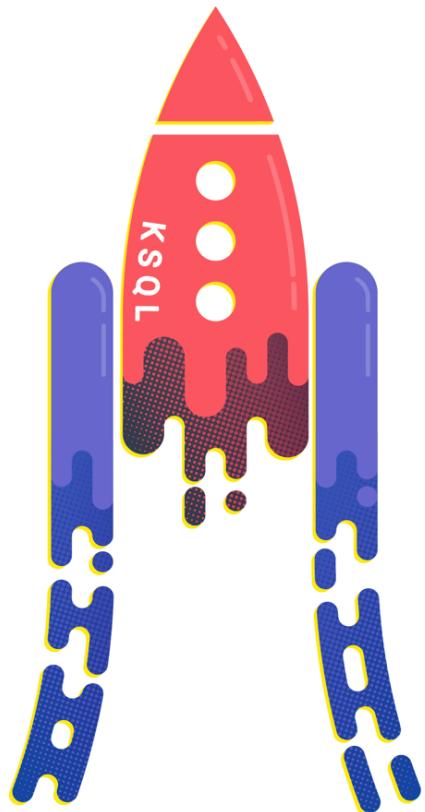


# Video Recording of the Live Demo



<https://www.youtube.com/watch?v=OTCuWK8PA1g>

# KSQL on DC/OS – It is a Kafka Streams app under the hood!



```
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 and KSQL Microservices run and scale on DC/OS via Marathon or Kubernetes



Questions? Feedback?  
Please contact me!



## Kai Waehner

Technology Evangelist

[kontakt@kai-waehner.de](mailto:kontakt@kai-waehner.de)

@KaiWaehner

[www.confluent.io](http://www.confluent.io)

[www.kai-waehner.de](http://www.kai-waehner.de)

[LinkedIn](#)

