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

Highly Scalable Microservices with Apache Kafka + Mesos
Independently Deployable

Highly Scalable Microservices with Apache Kafka + Mesos
Highly Scalable Microservices with Apache Kafka + Mesos

Allows us to scale
Scale in people terms

Highly Scalable Microservices with Apache Kafka + Mesos
Scale in infrastructure terms

Highly Scalable Microservices with Apache Kafka + Mesos
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
Highly Scalable Microservices with Apache Kafka + Mesos

Apache Kafka – A Distributed, Fault-Tolerant, Scalable Commit Log
Scale in infrastructure terms

Cluster (many machines)

- Service A instance 1
- Service A instance 2
- Service B instance 1
- Service B instance 2
Single, Shared Source of Truth
A MAJOR NEW ECOSYSTEM
CONFLUENT OPEN SOURCE
Highly Scalable Microservices with Apache Kafka + Mesos
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

Highly Scalable Microservices with Apache Kafka + Mesos
Stream Processing Pipeline

**Highly Scalable Microservices with Apache Kafka + Mesos**

**Stream Ingest**
- Messaging
- APIs
- Integration
- Adapters / Channels

**Stream Preprocessing**
- Normalization
- Filtering
- Transformation
- Enrichment
- Aggregation

**Stream Analytics**
- Contextual Rules
- Windowing
- Patterns
- Analytics
- Machine Learning
- ...

**Stream Outcomes**
- Index / Search
- Analytics / DW Reporting
- Process Management
- Applications & APIs
- Analytics (Real Time)

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
Kafka Streams (shipped with Apache Kafka)

Input Stream (Kafka Topic) -> Stream Processing Microservice (Kafka Streams) -> Output Stream (Kafka Topic)

Map, filter, aggregate, apply analytic model, "any business logic"

Deployed Anywhere
Java App, Docker, Kubernetes, Mesos, "you-name-it"
A complete streaming microservice, ready for production at large-scale

```
public static void main(String[] args) throws Exception {
    Properties config = new Properties();
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-example");
    config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092");
    config.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    config.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());

    KStreamBuilder builder = new KStreamBuilder();
    KStream<String, String> textlines = builder.stream("TextlinesTopic");
    KStream<String, Long> wordCounts = textlines
        .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
        .groupBy(key, word) -> word
        .count("Counts")
        .toStream();
    wordCounts.to(Serdes.String(), Serdes.Long(), "WordsWithCountsTopic");

    KafkaStreams streams = new KafkaStreams(builder, config);
    streams.start();
}
```
Highly Scalable Microservices with Apache Kafka + Mesos

---

**KSQL – A Streaming SQL Engine for Apache Kafka**

![Diagram of KSQL architecture](image)

---

```sql
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;
```

<table>
<thead>
<tr>
<th>timeWindow</th>
<th>productId</th>
<th>hourlyOrders</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>08:00:00</td>
<td>20</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>09:00:00</td>
<td>10</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>09:00:00</td>
<td>40</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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

Framework A
Scheduler

Framework B
Scheduler

Mesos Master Quorum

Leader

Master 1

ZK

Master 2

ZK

Master 3

ZK

Standby

Framework A
Executor (Task)

Framework B
Executor (Task)

Slave 1

Slave N

Offer

Offer

Offer

Offer
Components of a Kafka Cluster
Highly Scalable Microservices with Apache Kafka + Mesos

Mesos Architecture

- Marathon
  - Scheduler
- Kubernetes
  - Scheduler

Mesos Master Quorum

- Leader
  - Master 1
- Standby
  - ZK
  - Standby
  - Master 2
  - Master 3
  - ZK

Slaves

- Slave 1
  - Marathon
  - Executor (Kafka Broker)
- Slave N
  - Kubernetes
  - Executor (Kafka Streams)
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

Highly Scalable Microservices with Apache Kafka + Mesos
H2O.ai Model + Kafka Streams

1) Create H2O ML Model

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

2) Configure Kafka Streams Application

```java
// 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

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

4) Start Kafka Streams App

```java
KStream<String, Object> transformedMessage = airilineInputLines.mapValues((value) -> "KAI").
transformedMessage.to("AirlineOutputTopic");
final KafkaStreams streams = new KafkaStreams(builder, streamsConfiguration);
streams.start();
```
DC/OS on AWS

Highly Scalable Microservices with Apache Kafka + Mesos

Kafka Brokers on DC/OS

**Description**

Apache Kafka by Confluent


**Pre-Install Notes**

This DC/OS Service is currently a beta candidate undergoing testing as part of a formal beta test program. There may be bugs, incomplete features, incorrect documentation, or other discrepancies. Contact Mesosphere and Confluent before deploying this beta candidate service. Product support is available to approved participants in the beta test program. Mutual Mesosphere and Confluent customers are eligible to participate in the beta program. Contact your rep at either company or partner-support@confluent.io.

**Information**

Maintainer: partner-support@confluent.io

**Licenses**

Apache License v2: [https://raw.githubusercontent.com/confluentinc/kafka-trunk/LICENSE](https://raw.githubusercontent.com/confluentinc/kafka-trunk/LICENSE)

```
$ dcos kafka --name confluent-kafka endpoints broker
{
   "address": [
      "10.0.2.118:1025",
      "10.0.2.119:1025",
      "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.4lib.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

Highly Scalable Microservices with Apache Kafka + Mesos
Highly Scalable Microservices with Apache Kafka + Mesos

KSQL on DC/OS – Why not?

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

<table>
<thead>
<tr>
<th>timeWindow</th>
<th>productId</th>
<th>hourlyOrders</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>08:00:00</td>
<td>20</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>09:00:00</td>
<td>10</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>09:00:00</td>
<td>40</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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