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

Kai Waehner
Technology Evangelist
kontakt@kai-waehner.de
LinkedIn
@KaiWaehner
www.confluent.io
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

Orders Service

Fulfillment Service

Stock Service

Basket Service

Payment Service
Microservices are Independently Deployable

- Orders Service
- Fulfillment Service
- Stock Service
- Basket Service
- Payment Service
Scale in People Terms
Scale in Infrastructure Terms

Cluster (many machines)

Service A instance 1
Service A instance 2
Service B instance 1
Service B instance 2
Highly 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) Use Case - Scalable Flight Prediction
Apache Kafka – A Distributed, Scalable, Fault-Tolerant Commit Log

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

Producer

Consumer

Consumer

Consumer
A MAJOR NEW ECOSYSTEM
Highly Scalable Microservices with Apache Kafka + Mesos
CONFLUENT OPEN SOURCE

Highly Scalable Microservices with Apache Kafka + Mesos
Highly Scalable Microservices with Apache Kafka + Mesos
Apache Kafka – A Streaming Platform

MySQL

Kafka Connect

Kafka Cluster

Kafka Streams

KSQL

Kafka Connect

Elasticsearch
Kafka as Single, Shared Source of Truth for Microservices

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) Use Case - Scalable Flight Prediction
Highly Scalable Microservices with Apache Kafka + Mesos

Data at Rest

Data in Motion
Stream Processing Pipeline

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)
When to use Kafka Streams for Stream Processing?
Lightweight, Independent, Scalable Streaming Microservices

Highly Scalable Microservices with Apache Kafka + Mesos
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)

Map, filter, aggregate, apply analytic model, “any business logic”

Input Stream (Kafka Topic) → Stream Processing Microservice (Kafka Streams) → Output Stream (Kafka Topic)

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

Kafka Cluster

Highly Scalable Microservices with Apache Kafka + Mesos
A complete streaming microservice, ready for production at large-scale

App configuration

```java
public static void main(final String[] args) throws Exception {
    Properties config = new Properties();
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-example");
    config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker: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();
}
```

Define processing (here: WordCount)

Start processing
KSQ – An Open Source Streaming SQL Engine for Apache Kafka
Highly Scalable KSQL Streaming Microservices

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

```
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';
```
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
"...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...

“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

Framework A
Scheduler

Framework B
Scheduler

Mesos Master Quorum
Leader
Master 1
ZK
ZK
ZK
Standby
Master 2
Standby
Master 3

Offer

Slave 1
Framework A
Executor (Task)

Offer

Slave N
Framework B
Executor (Task)
Components of a Kafka Cluster

- **Kafka Streams**
  - Container
  - KStreams App
  - State store

- **Kafka Broker**
  - Broker node
    - Broker
    - Data

- **Zookeeper**
  - Zookeeper
  - Data

Additional components:
- REST Proxy
- Schema Registry
- Connect worker
Highly Scalable Microservices with Apache Kafka + Mesos

Mesos Architecture

- Marathon Scheduler
- Kubernetes Scheduler
- Mesos Master Quorum
  - Leader
  - ZK
  - Standby
    - Master 2
    - Master 3
  - 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) 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

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.gbm.GenModel genModel = new hex.gbm.GenModel();
// Set up training data
hex.gbm.train(x, y)
// Train model
hex.gbm.H2OModel model = genModel.train();
// Save model
hex.model.Savel(model);
```

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
// Read in flight data
Map<String, String> flightData = new HashMap<>();
flightData.put("1", "flight1");
flightData.put("2", "flight2");
flightData.put("3", "flight3");
// Apply H2O ML Model to streaming data
Map<String, String> predictions = new HashMap<>();
predictions.put("1", "MLPrediction1");
predictions.put("2", "MLPrediction2");
predictions.put("3", "MLPrediction3");
// Print predictions
System.out.println("ML Prediction:");
```

4) Start Kafka Streams App

```java
KStream<String, Object> transformedMessage = airInPutLinesMapValues.values().flatMapValues((K, V) -> {
    if (V.toString().contains("flight1")) {
        return Arrays.asList(new SimpleEntry<>(K, V));
    } else {
        return Collections.emptyList();
    }
});
transformedMessage.toStream("KafkaOutputTopic");
```

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 Automation.


Logical ID | Physical ID | Type       | Status
---        | ---         | ---        | ---

AdminSecurityGroup
DHCOptons
ElasticLoadBalancer
ExhibitorS3Bucket
GatewayToInternet
InboundNetworkACL
InternalMasterLoadBalancer
InternetGateway
LbSecurityGroup
MasterInstanceProfile
MasterLaunchConfig
MasterRole
MasterSecurityGroup
MasterServerGroup
MasterToMasteringress
MasterToPublicSlaveingress


Highly Scalable Microservices with Apache Kafka + Mesos
Kafka Brokers on DC/OS

![Confluent Kafka on DC/OS](image)

**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": [
    "10.0.2.155:1025",
    "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"
}
```

**Highly Scalable Microservices with Apache Kafka + Mesos | 42**
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 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-jre
MAINTAINER Kai Waehner
curl http://apache.mirrors.spacedump.net/kafka/0.11.0.1/kafka_2.11-0.11.0.1.tgz | tar xfvz --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 on DC/OS

Highly Scalable Microservices with Apache Kafka + Mesos
Live Demo

Highly Scalable Microservices with Apache Kafka + Mesos

Kafka

Mesos
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
  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>
</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 and KSQL Microservices run and scale on DC/OS via Marathon or Kubernetes
Questions? Feedback?
Please contact me!