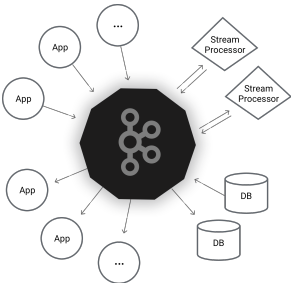


# Apache Kafka

Daniel, Fabian, Hauke und Tom

Modellierung von Informationssystemen  
Department Informatik  
HAW Hamburg

01. Dezember 2017



## 1 Konzept

- Einführung
- Grundlagen (Queue & Topic)
- Kafka Topic
- Kafka Eigenschaften
- Performance

## 2 Tutorial

- Quickstart
- Properties
- Kafka Clients
- Twitter App

# Was ist Apache Kafka?

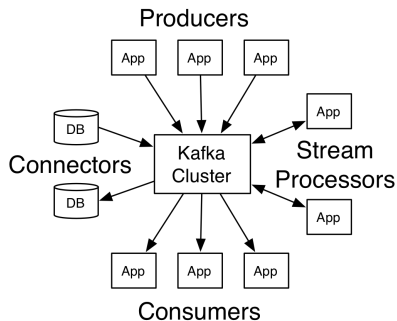


Abbildung 1.1: Apache Kafka [1]

Apache Kafka ist eine verteilte skalierbare Streaming Plattform.

# Eigenschaften

Kafka ...

- ist ein Message Queuing System
- kann Nachrichten speichern
- kann Nachrichten verarbeiten
- kann all das in Echtzeit

# Motivation

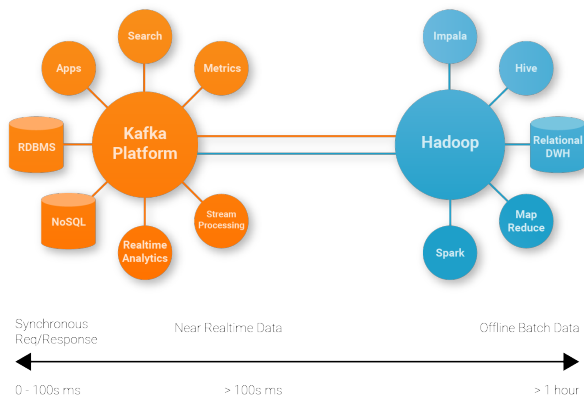


Abbildung 1.2: Kafka and Hadoop [2]

# Unternehmen und Use Cases



Operational Metrics



OpenSOC (Security Operations Center)



Real-time Monitoring and Event-processing Pipeline

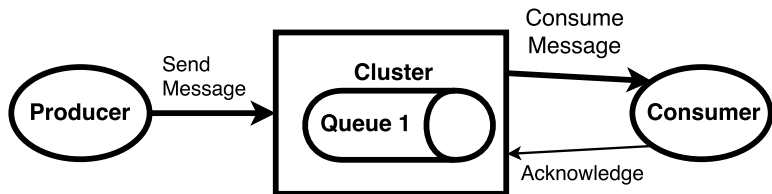


Log Delivery System

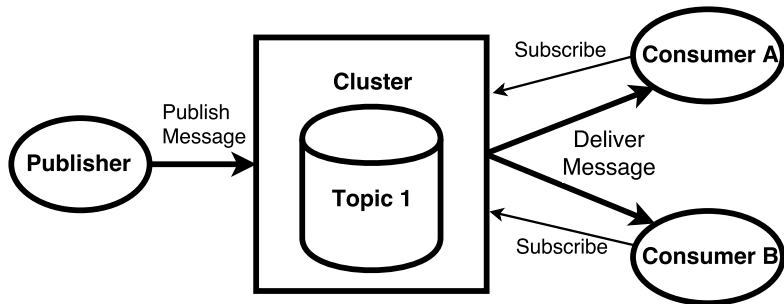


Part of Storm Stream Processing Infrastructure

# Queue



# Topic



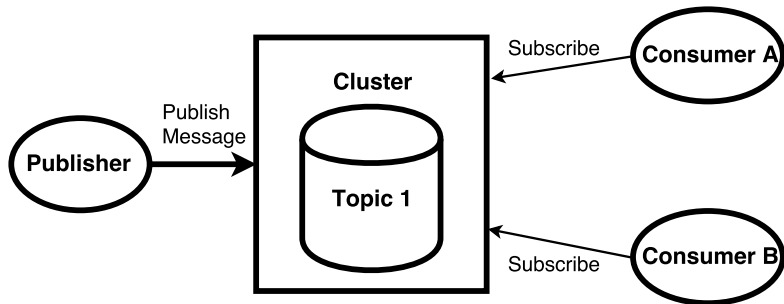


# Zusammenfassung

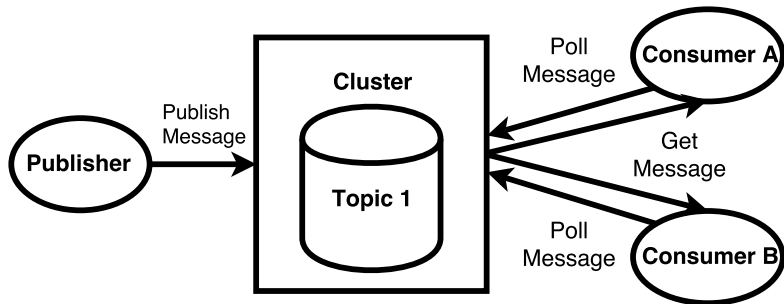
Bisher:

- Queueing
  - ▶ Nachricht 1:1 Consumer
  - ▶ Nachrichtenverarbeitung skaliert
  - ▶ Nachricht abgerufen = Nachricht weg
- Topic
  - ▶ Nachrichten 1:N Consumer
  - ▶ Nachrichten werden verteilt
  - ▶ Skaliert nicht

# Kafka Topic



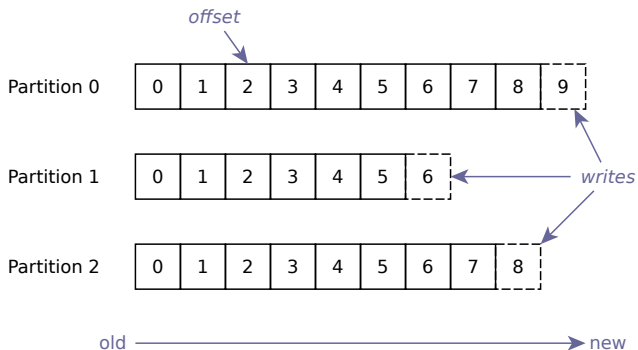
# Kafka Topic



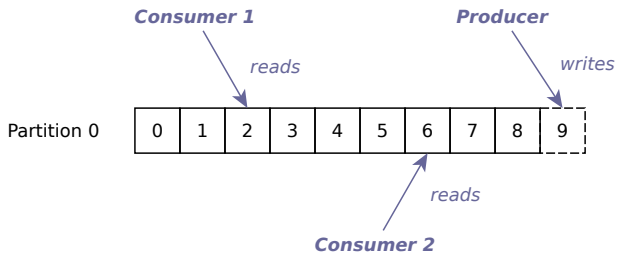
# Kafka Topics

- Vereinigt klassischen Queue- und Topicsansatz
- Multi-Subscribe (0 bis  $n$  Consumer)
- Records in Topics werden persistent gehalten
- Topics benötigen eine Cleanup-Policy
  - ▶ Retention-Time
  - ▶ Retention-Size
  - ▶ Log-Compaction
- Guarantees
  - ▶ Recordreihenfolge wird eingehalten
  - ▶ Records werden ausgeliefert, wie sie im Log stehen
  - ▶ N-1 Serverausfälle bei N Replikationen ohne Datenverluste

# Partitionen



# Partitionen



# Partitionen

- 1.. $n$  Partitionen für jedes Topic
- Eine Partition ist
  - ▶ Geordnet
  - ▶ Nicht-Veränderbare Sequenz von Records
  - ▶ Records können angehängt werden
- Records sind nummeriert
- Records werden nach Cleanup-Policy entfernt
- Sequentielle Abarbeitung ist Standard
- Sprung im Record-Log möglich

# Partitionen

- Verteilung der Partitionen ermöglicht
  - ▶ Gute Skalierung
  - ▶ Parallele Abarbeitung



# Kafka als Nachrichtensystem

- Consumer Groups
  - ▶ Kombiniert Queueing und Publish-Subscribe
  - ▶ Nachrichtenverarbeitung in Gruppen
  - ▶ Mehrere Consumer in einer Gruppe
- Vorteile?
  - ▶ Nachrichtenverarbeitung skaliert?
- Reihenfolge wird eingehalten?

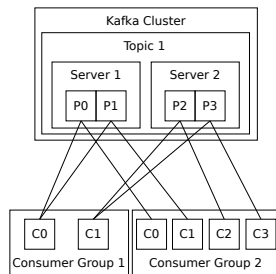


Abbildung 1.3: Consumer Groups [1]

# Kafka als Datenbank

## Kafka as a Storage System

"Kafka [is] a kind of special purpose distributed filesystem dedicated to high-performance, low-latency commit log storage, replication, and propagation." [1]

- Durch Funktionalität bedingt
  - ▶ Entkopplung von Consumer und Producer sorgt für Speicherbedarf
- Daten werden repliziert
  - ▶ Bestätigungsmechanismen sind vorhanden
  - ▶ Wird erst bestätigt, wenn Replication abgeschlossen ist

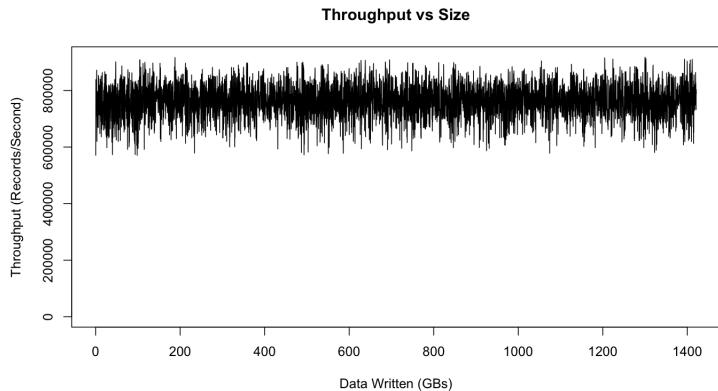
# Kafka für Stream Processing

- Anforderung: Streamverarbeitung in *Echtzeit*!
- Ein Stream Processor ...
  - ▶ nimmt kontinuierlich Daten aus einem *Input* Topic,
  - ▶ bearbeitet die Daten und
  - ▶ schreibt kontinuierlich Daten in ein *Output* Topic

# Kafka für Stream Processing - Stream API

- Stream API wird für nicht-triviales Stream Processing angeboten, z.B. zur Aggregation oder Joins von Streams.
- Stream API unterstützt
  - ▶ *Exactly-once* Verarbeitung von Daten
  - ▶ Statusbehaftete Operationen, wie Joins und Aggregationen über Bereiche
  - ▶ Erneute Verarbeitung von Daten, wenn sich die Operation ändert
  - ▶ *One-record-at-a-time Processing*, um Verarbeitungslatenz im Millisekundenbereich garantieren zu können

# Performance



# Zusammenfassung

- geeignet für:
  - ▶ Bearbeitet die Daten
  - ▶ Schreibt kontinuierlich Daten in ein Topic

# Tutorial

- Download Kafka [3]

```
$ tar -xzf kafka_2.11-1.0.0.tgz
$ cd kafka_2.11-1.0.0
```

```
$ bin/zookeeper-server-start.sh config/zookeeper.
  properties
$ bin/kafka-server-start.sh config/server.properties
```

# Tutorial

```
$ bin/kafka-topics.sh --create --zookeeper localhost
:2181 --replication-factor 1 --partitions 1 --
topic test
$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
> This is a message
> This is another message
```

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test --from-beginning
> This is a message
> This is another message
```



# Tutorial

Name	Beschreibung	Typ
broker-list/bootstrap.servers	Host und Port für das Cluster	Liste
topic	cell5	cell6
cell7	cell8	cell9

Tabelle 1: Producer Properties

# Tutorial

- Broker Properties [4]

# Tutorial

- Diverse Clients vorhanden [5]
  - ▶ Java, Python, Go, C/C++, .NET, Ruby, ...
- Kafka in Java, daher der meiste Support

# Tutorial

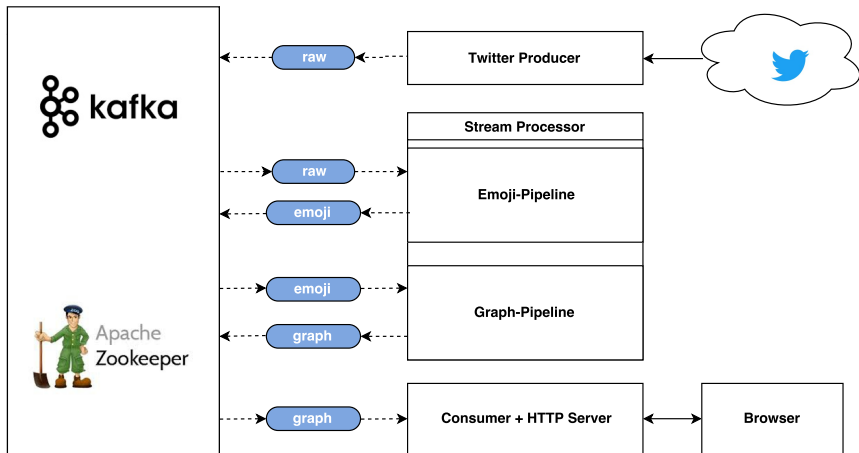


Abbildung 2.1: Architektur Twitter App

# Tutorial

```
# Create Producer instance
p = Producer(**conf) #from config file

# connect to twitter
api = connect_to_twitter(config)
stream = api.GetStreamFilter(track=[search_term])
for line in stream:
    tweet = line_to_text(line)
    p.produce(topic, bytes(tweet, 'utf-8'))
```

Listing 1: Python Producer

# Tutorial

```
# Kafka config
kafka_config.BOOTSTRAP_SERVERS = kafka_url
kafka_config.AUTO_OFFSET_RESET = 'earliest'

emoji_topic = src_topic + '-emoji'
target_topic = src_topic + '-chart'

with TopologyBuilder() as topology_builder1:
    topology_builder1. \
        source('tweets', [src_topic]). \
        processor('emoji', EmojiParserProcessor, 'tweets'). \
        sink('emojis', emoji_topic, 'emoji')

kafka_streams.KafkaStreams(topology_builder1, kafka_config).start
()

# close on termination
```

Listing 2: Python Processor

# Tutorial

```
# Create Consumer
c = Consumer(**conf) # external config

c.subscribe([topic])
running = True
print('Start polling..\nStop with ctrl-c..')
while running:
    msg = c.poll()
    try:
        if not msg.error():
            with open(filename, 'wb') as f:
                f.write(msg.value())
        else:
            pass
    except KeyboardInterrupt:
        running = False
```

Listing 3: Python Consumer

## Screencast Demo





Apache Foundation. *Apache Kafka Documentation*. 2017. URL: <https://kafka.apache.org/> (besucht am 20.11.2017).



Jun Rao. *The value of Apache Kafka in Big Data ecosystem*. 2017. URL: <https://www.confluent.io/blog/the-value-of-apache-kafka-in-big-data-ecosystem/> (besucht am 20.11.2017).



Apache Foundation. *Apache Download Mirror*. 2017. URL: [https://www.apache.org/dyn/closer.cgi?path=/kafka/1.0.0/kafka\\_2.11-1.0.0.tgz](https://www.apache.org/dyn/closer.cgi?path=/kafka/1.0.0/kafka_2.11-1.0.0.tgz) (besucht am 30.11.2017).



Apache Foundation. *Apache Kafka Broker Properties*. 2017. URL: <https://kafka.apache.org/documentation/#brokerconfigs> (besucht am 30.11.2017).



Apache Foundation. *Clients - Apache Kafka*. 2017. URL: <https://cwiki.apache.org/confluence/display/KAFKA/Clients> (besucht am 30.11.2017).



Jay Kreps. *Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines)*. 2014. URL: <https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines> (besucht am 20.11.2017).



Joel Koshy. *Kafka Ecosystem at LinkedIn*. 2016. URL: <https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin> (besucht am 20.11.2017).