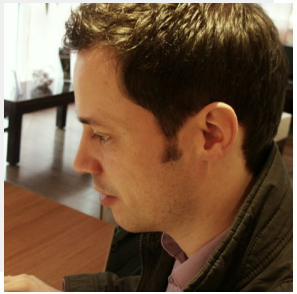
Apache Kafka - Lessons Learned

Holger Adam – Andreas Kluth

REWE digital

Holger Adam
Softwareentwickler @ BigData
https://github.com/holgeradam

Andreas Kluth
Softwareentwickler @ eCom
https://github.com/AndreasKl



Agenda

1) Intro
    a) Wie funktioniert Kafka?
    b) Wozu nutzen wir Apache Kafka?
2) Config & Broker
    a) Keiner weiß so richtig, wie ein Topic funktioniert
    b) Kopfrechnen ist schwer, Konfiguration auch
3) Java-API
    a) Consumer sind einfach
    b) Producer aber bitte zuverlässig
4) Q&A

Wie funktioniert Kafka?

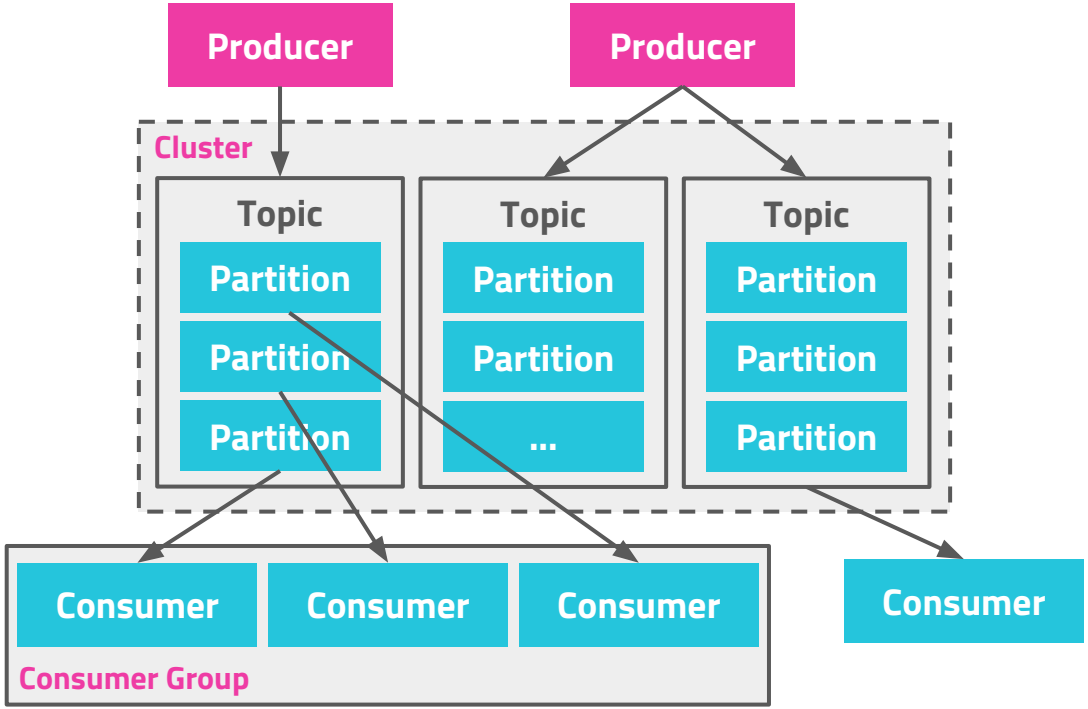
Kafka @ REWE Digital

Eine verteilte “streaming” Platform

(persistentes Commit-Log)

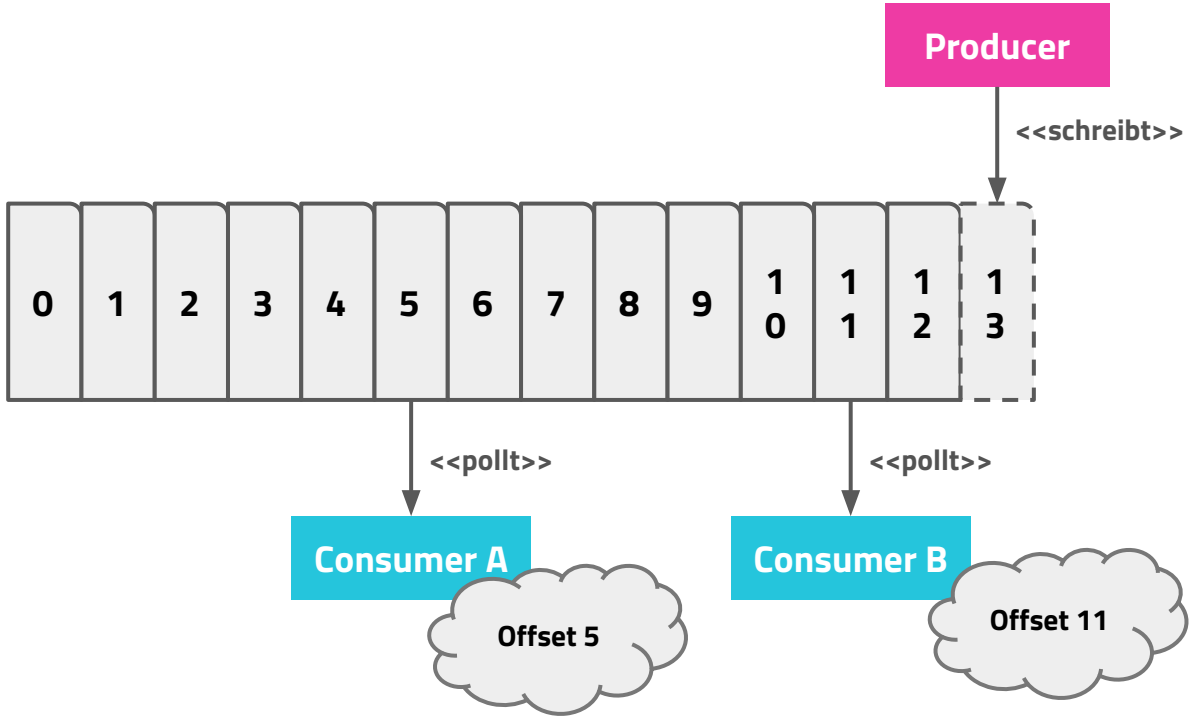
Wie funktioniert Kafka?

Kafka @ REWE Digital



Wie funktioniert Kafka?

Kafka @ REWE Digital



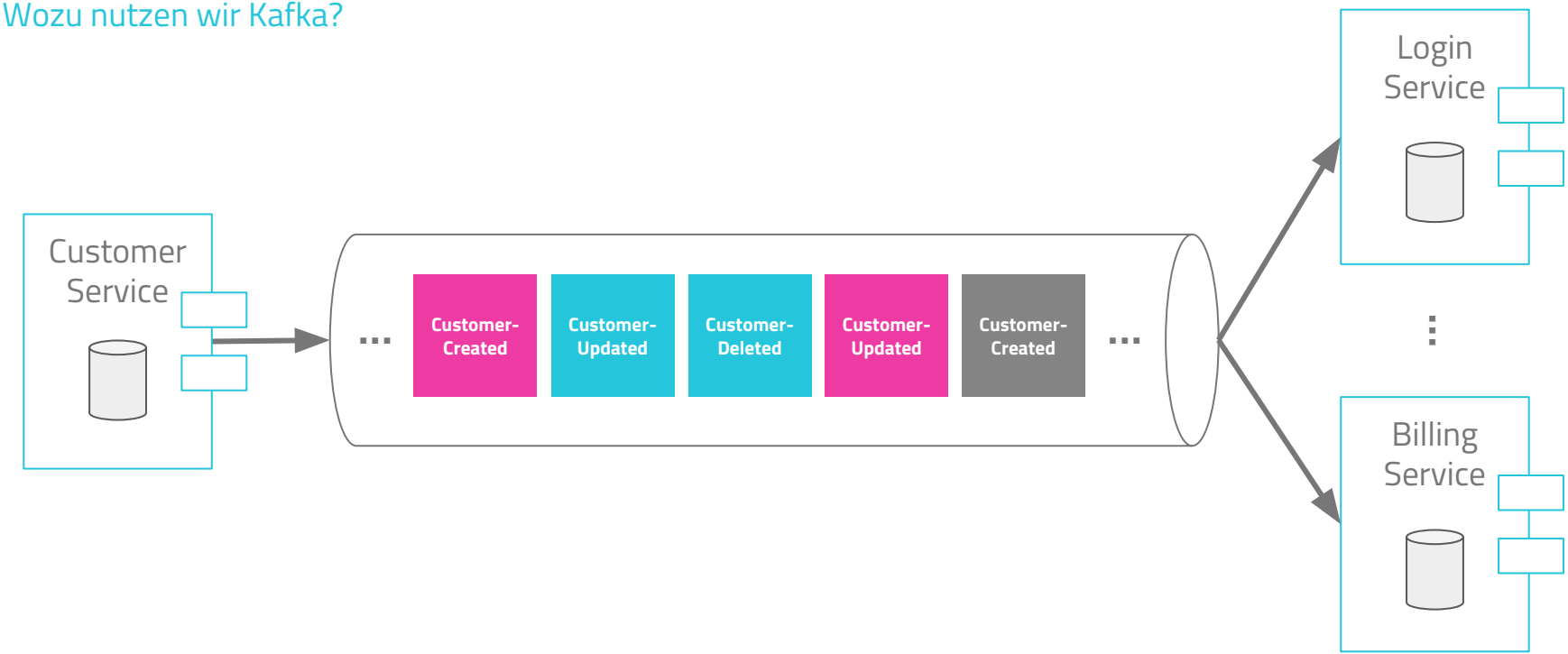
Wie funktioniert Kafka?

Kafka @ REWE Digital

- **at most once**, **at least once** und **exactly once**
- Reihenfolge pro Partition garantiert (beim Lesen)
- *Nachrichten die ein "ACK" von allen **in-sync-replicas** (ISR) im Cluster bekommen sind persistiert (optional)*
- Skaliert horizontal (petabyte scale), kann schnell

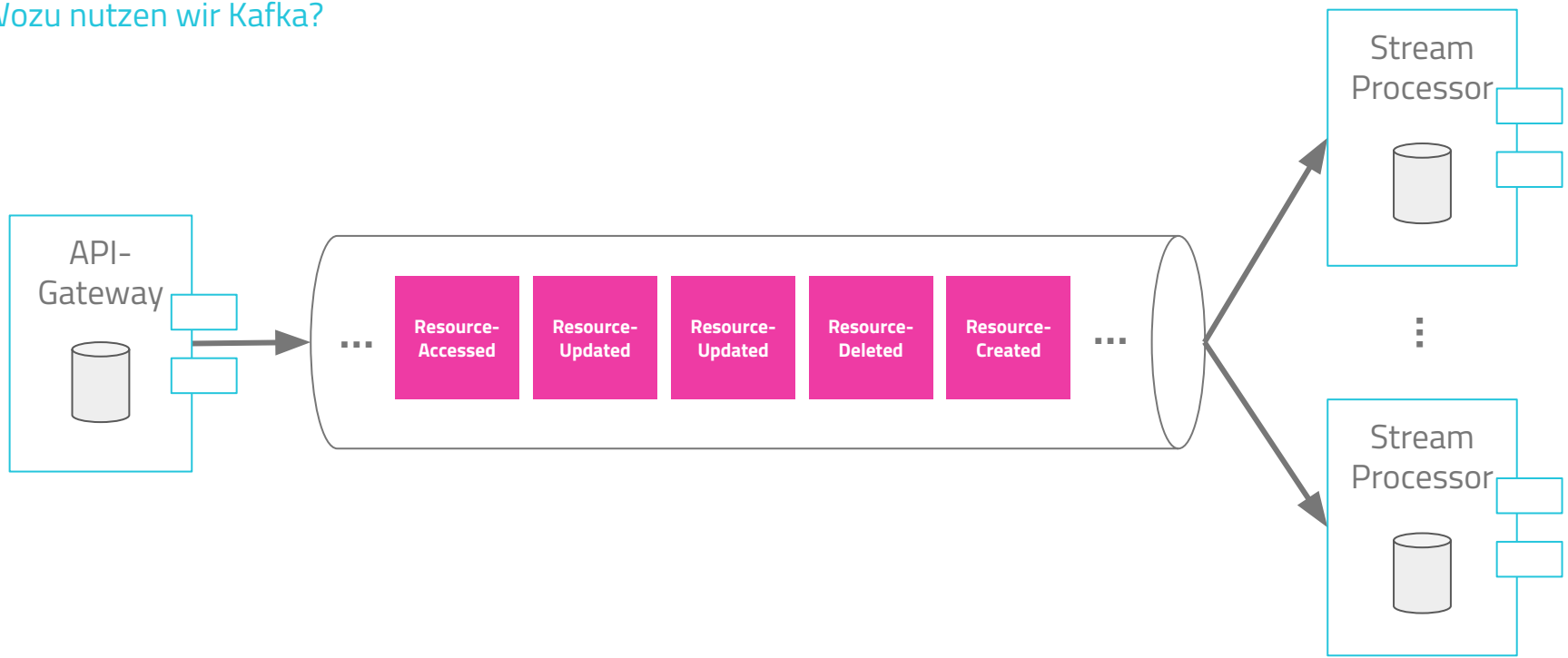
Entkopplung von Services via Events

Wozu nutzen wir Kafka?



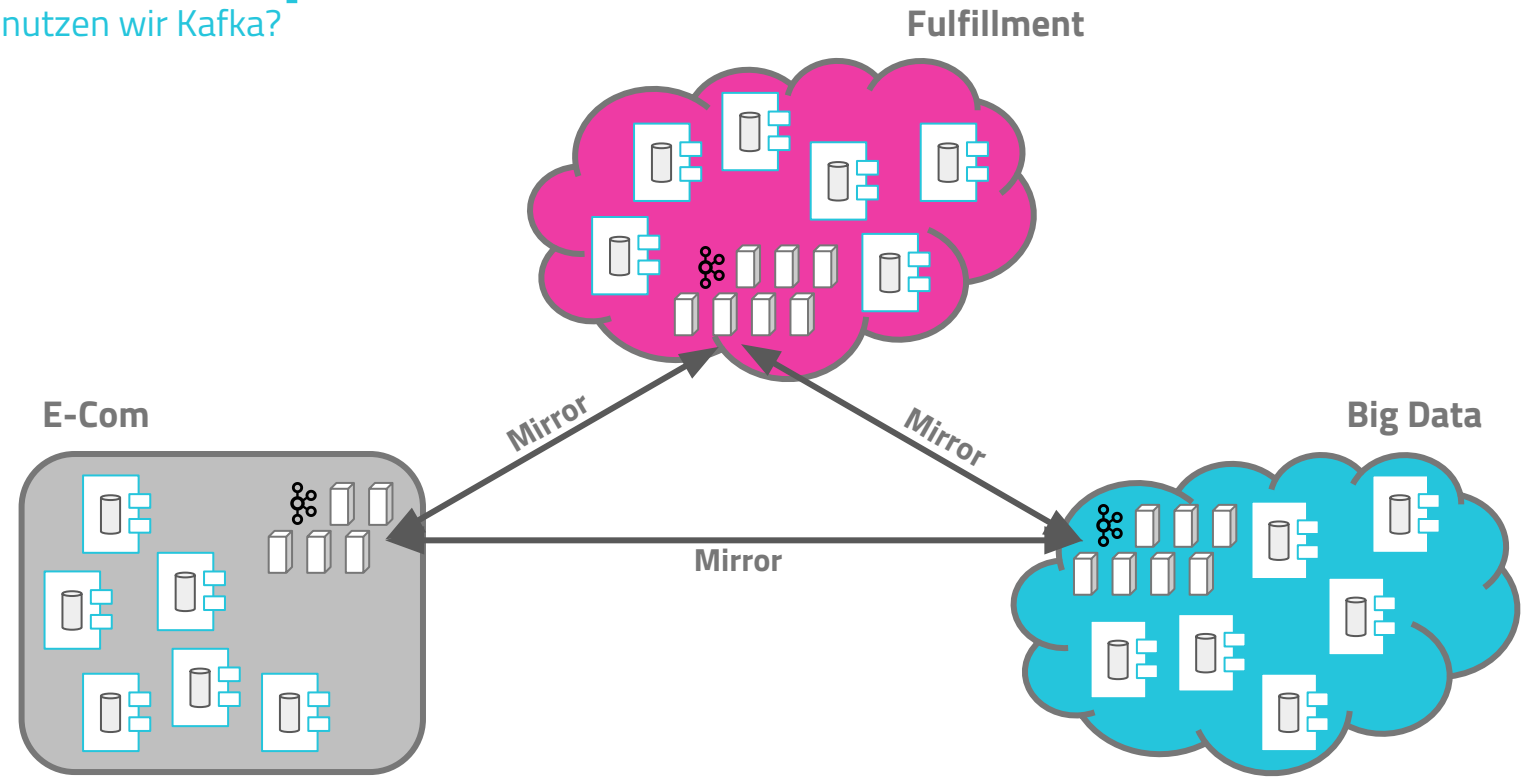
Access Logs / Click Streams

Wozu nutzen wir Kafka?



Unser Setup

Wozu nutzen wir Kafka?



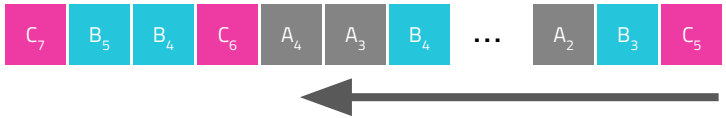

Keiner weiß so richtig, wie ein Topic funktioniert

Keiner weiß so richtig, wie ein Topic funktioniert

cleanup.policy=delete | compact

Compact

Letzte Version behalten
Nachrichten werden anhand des Keys in letzter Version gespeichert.

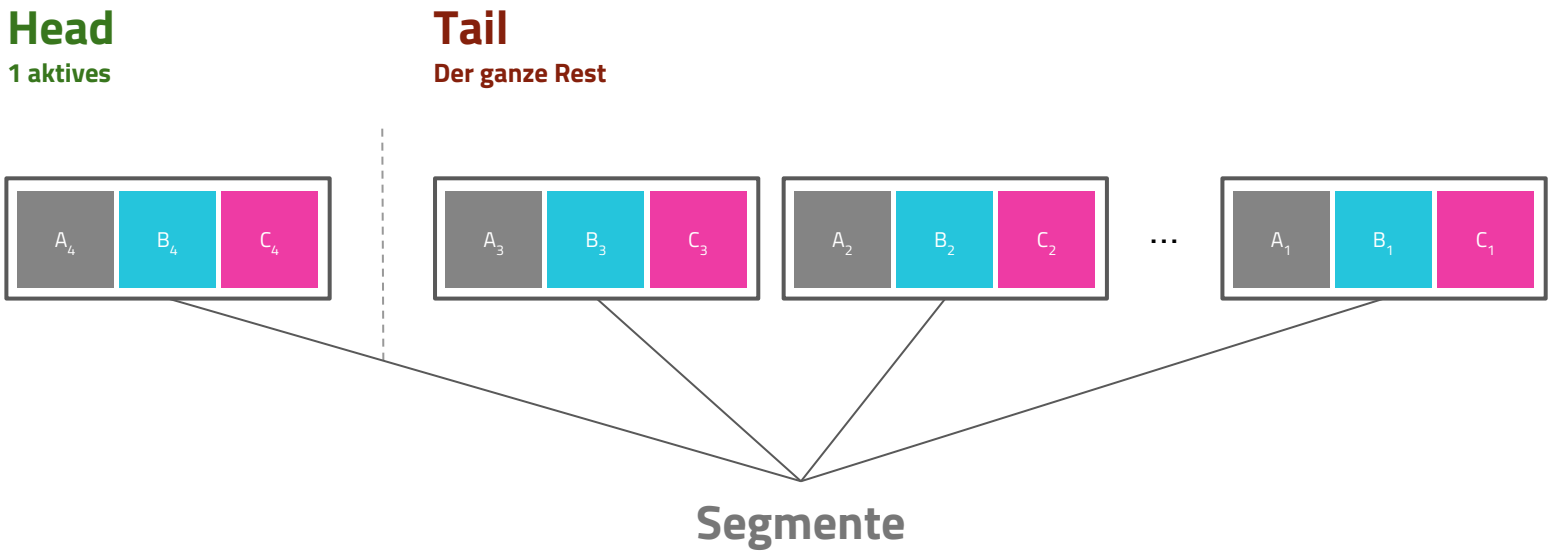


Delete

Nach Größe / Zeit
Nachrichten werden gelöscht, wenn ein eingestelltes Limit erreicht wird.

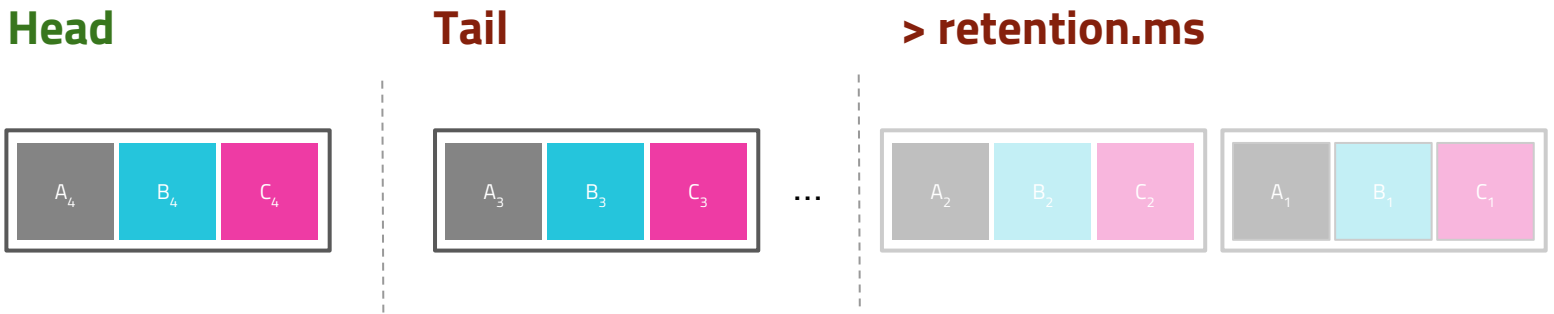
Keiner weiß so richtig, wie ein Topic funktioniert

segment.ms | segment.bytes



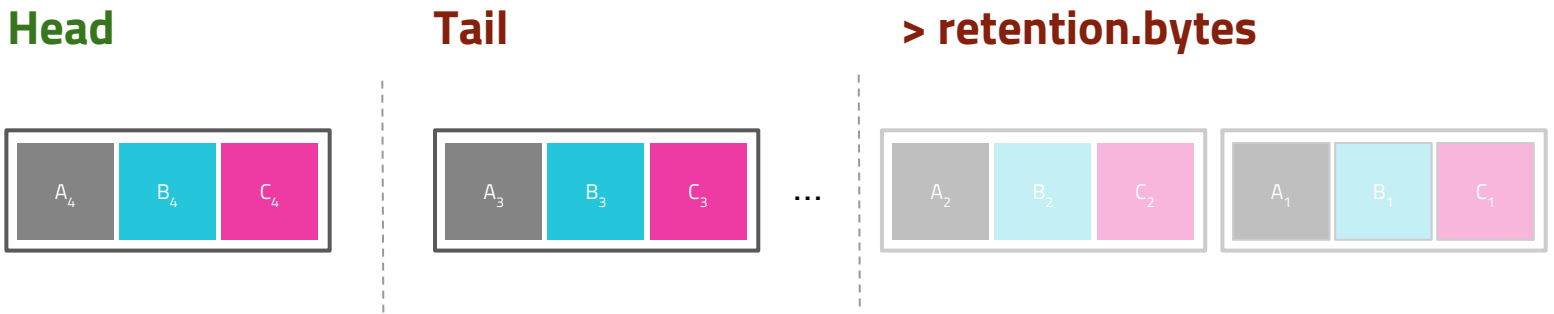
Keiner weiß so richtig, wie ein Topic funktioniert

retention.ms=1024102400



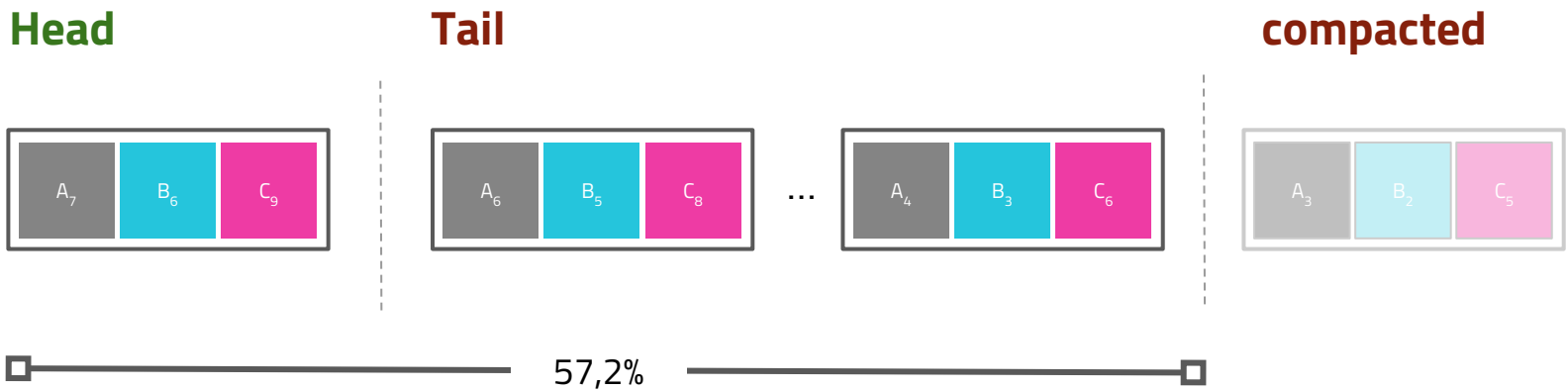
Keiner weiß so richtig, wie ein Topic funktioniert

retention.bytes=1024102400



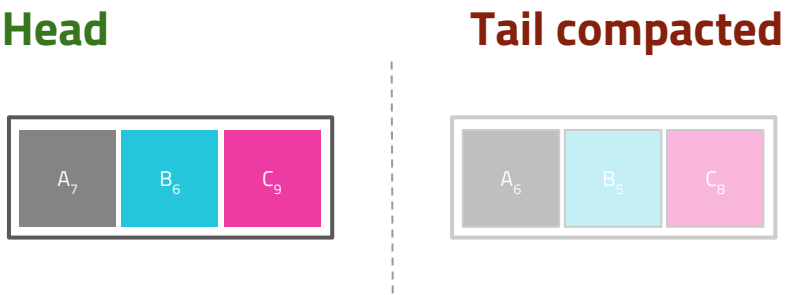
Keiner weiß so richtig, wie ein Topic funktioniert

min.cleanable.dirty.ratio=0.5



Keiner weiß so richtig, wie ein Topic funktioniert

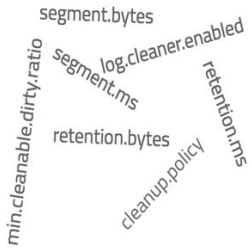
min.cleanable.dirty.ratio=0.5

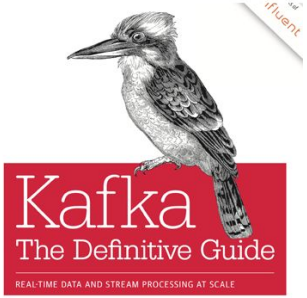


Keiner weiß so richtig, wie ein Topic funktioniert

Fazit

- Wer konfiguriert, muss wissen, was er tut.
- Lieber ein schneller Test als ein verzweifelter Admin.
- Lesen, lesen, lesen.






Kopfrechnen ist schwer, Konfiguration auch

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

```
~>./kafka-topics.sh --create --topic froscon --config cleanup.policy=delete
--config retention.ms=___ --config segment.ms=___ --config segment.bytes=___
```

Löschung nach 2 Wochen?

1000 * 60 * 60 * 24 * 14

= 120960000

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

```
~>./kafka-topics.sh --create --topic froscon --config cleanup.policy=delete
--config retention.ms=120960000 --config segment.ms=___ --config segment.bytes=___
```

10 Stunden Segmente?

1000 * 60 * 60 * 10

= 36000000

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

```
~>./kafka-topics.sh --create --topic froscon --config cleanup.policy=delete
--config retention.ms=120960000 --config segment.ms=36000000 --config segment.bytes=___
```

500MiB Segmente?

1024 * 1024 * 512

= 536870912

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

```
~>./kafka-topics.sh --create --topic froscon
--config cleanup.policy=delete
--config retention.ms=120960000
--config segment.ms=36000000
--config segment.bytes=536870912
```

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

```
~>./kafka-topics.sh --create --topic froscon
--config cleanup.policy=delete
--config retention.ms=120960000
--config segment.ms=36000000
--config segment.bytes=107374182
```

Kopfrechnen ist schwer, Konfiguration auch

./kafka-topics.sh

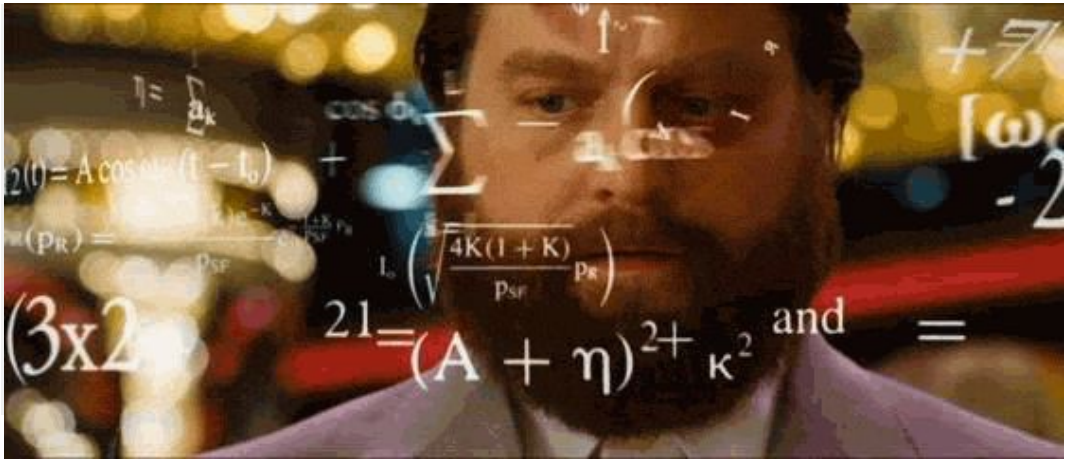
```
# 28d retention, 24h/1gb segments
store-my-very-long-please:cleanup.policy=delete,retention.ms=2073600000,segment.bytes=1073741824,segment.ms=86400000
me-too:cleanup.policy=delete,retention.ms=2073600000,segment.bytes=1073741824,segment.ms=86400000

# 7d retention, 24h/1gb segments
i-have-a-large-throughput:cleanup.policy=delete,retention.ms=604800000,segment.bytes=1073741824,segment.ms=86400000
i-am-a-clickstream:cleanup.policy=delete,retention.ms=604800000,segment.bytes=1073741824,segment.ms=86400000
access-log:cleanup.policy=delete,retention.ms=604800000,segment.bytes=1073741824,segment.ms=86400000
audit-log:cleanup.policy=delete,retention.ms=604800000,segment.bytes=1073741824,segment.ms=86400000
```

Kopfrechnen ist schwer, Konfiguration auch

Fazit

- Die Kommandozeile ist nicht sehr komfortabel, der Broker nicht viel besser.
- Der kleinste gemeinsame Nenner sollte die Standardkonfiguration im Broker sein.
- Topic kann beliebig neu konfiguriert werden.
- Optimal: Topics im Code anlegen, Computer rechnen lassen.




Consumer sind einfach

Consumer sind einfach

Stackoverflow FTW

```
val consumerConfig = Properties()
consumerConfig[BOOTSTRAP_SERVERS_CONFIG] = "127.0.0.1:9092"
consumerConfig[GROUP_ID_CONFIG] = "rewe-group"
consumerConfig[VALUE_DESERIALIZER_CLASS_CONFIG] = StringDeserializer::class.java.getName()
consumerConfig[KEY_DESERIALIZER_CLASS_CONFIG] = StringDeserializer::class.java.getName()

val consumer = KafkaConsumer<String, String>(consumerConfig)
consumer.subscribe(listOf("rewe-topic"))

while (true) {
   consumer.poll(Duration.ofSeconds(1)).forEach { record ->
       println(record.toString())
   }
}
```

Consumer sind einfach

Nach ein paar Stunden ausprobieren...

```
...
consumerConfig[ENABLE_AUTO_COMMIT_CONFIG] = false
...

consumer.use { con ->
   while (true) {
       con.poll(Duration.ofSeconds(1)).forEach { record ->
           println(record.toString())
       }
       con.commitSync()
   }
}
```

Consumer sind einfach

Fazit

- `autocommit.enable = false`
- **Manuelles setzen des Offsets** nachdem die Nachricht prozessiert/persistiert wurde
  im Batch oder pro Nachricht
- Doppelte Nachrichten berücksichtigen
- Nicht prozessierbare Nachrichten zur Seite legen (Dead Message Queue, Staging Table) oder überspringen
- Den Consumer bei nicht automatisch lösbaren Problemen stoppen
- Ein Consumer pro Thread
- Nur in der Größe beschränkte Queues verwenden mit geringen Limits und auf volle Queues reagieren
- **Gutes Monitoring (bspw. Lag, Anzahl der Nachrichten, Fehler) und Alerting**
- Kill Switch für die Sysops

Producer, aber bitte zuverlässig

Producer, aber bitte zuverlässig

Diesmal die JavaDoc von Kafka.

```
val props = Properties()
props["bootstrap.servers"] = "127.0.0.1:9092"
props["acks"] = "all"
props["retries"] = 0
props["batch.size"] = 1024
props["linger.ms"] = 0
props["buffer.memory"] = 8096
...

KafkaProducer<String, String>(props).use { p ->
    for (i in 1..10_000) {
        p.send(ProducerRecord("rewe-topic", i.toString(), i.toString()))
    }
}
```

Producer, aber bitte zuverlässig

Fazit

- **`acks = all`**

  unclean leader election deaktivieren, InSyncReplicas auf mind. 2, Replikation auf mind. 3
- **`retries = 0`** (reordering)
- **`max.in.flight.requests.per.connection = 1`** (reordering when **`retries`** aktiv)

- Ab Broker-Version 1.0.0:

  `enable.idempotence = true`

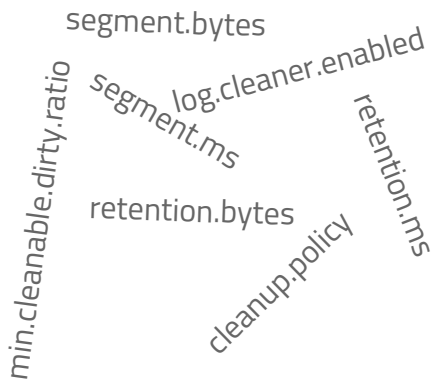

Fragen ?


Danke!


Would you like to know more?

Quellen

- Kafka Dokumentation [https://kafka.apache.org/documentation]
- Kafka - The Definitive Guide [https://www.confluent.io/resources/kafka-the-definitive-guide]
- https://github.com/apache/kafka
- https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html
- https://kafka.apache.org/10/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html


segment.bytes
log.cleaner.enabled
segment.ms
retention.ms
retention.bytes
cleanup.policy
min.cleanable.dirty.ratio