

Kafka Notes:

1. Topic

2. Topics, Partitions and Offsets.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Topics, partitions and offsets

Stephane Maarek

- Topics: a particular stream of data
 - Similar to a table in a database (without all the constraints)
 - You can have as many topics as you want
 - A topic is identified by its name
- Topics are split in partitions
 - Each partition is ordered
 - Each message within a partition gets an incremental id, called offset

Kafka Topic

Partition	Offsets
Partition 0	0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1	0 1 2 3 4 5 6 7 8
Partition 2	0 1 2 3 4 5 6 7 8 9 10

writes

02:32 77% 06:47

2. Topics, Partitions and Offsets.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Topics, partitions and offsets

Stephane Maarek

Kafka Topic

Partition	Offsets
Partition 0	0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1	0 1 2 3 4 5 6 7 8
Partition 2	0 1 2 3 4 5 6 7 8 9 10

writes

- Offset only have a meaning for a specific partition.
 - E.g. offset 3 in partition 0 doesn't represent the same data as offset 3 in partition 1
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is one week)
- Once the data is written to a partition, it can't be changed (immutability)
- Data is assigned randomly to a partition unless a key is provided (more on this later)


Okay? So see you in the next lecture.

05:46 77% 06:47

2. Cluster and Broker

3. Brokers and Topics.mp4 - VLC media player

Brokers



- A Kafka cluster is composed of multiple brokers (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is 3 brokers, but some big clusters have over 100 brokers
- In these examples we choose to number brokers starting at 100 (arbitrary)

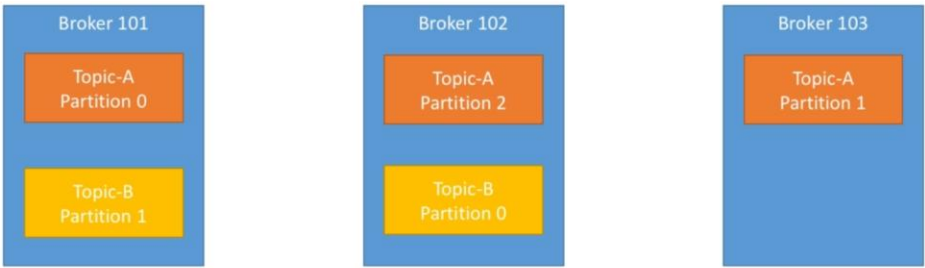
So 101, 102, and 103.

01:37 03:19

Brokers and Topic: Like server and data

3. Brokers and Topics.mp4 - VLC media player

Brokers and topics



- Example of **Topic-A** with **3 partitions**
- Example of **Topic-B** with **2 partitions**

• Note: Data is distributed and Broker 103 doesn't have any **Topic B** data

03:19 03:19

3. Topic replication factor

Replication factor of 2 means that have 2 copies for each data(partition)

4. Topic Replication.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help

Topic replication factor

- Topics should have a replication factor > 1 (usually between 2 and 3)
- This way if a broker is down, another broker can serve the data
- Example: Topic-A with 2 partitions and replication factor of 2

Diagram illustrating Topic-A with 2 partitions and a replication factor of 2. Broker 101 contains Partition 0. Broker 102 contains Partition 1 and Partition 0. Broker 103 contains Partition 1. Arrows indicate replication between brokers.

01:40 04:12 75%

When a broker(server) shutdown, another broker still serve the data

4. Topic Replication.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help

Topic replication factor

- Example: we lost Broker 102
- Result: Broker 101 and 103 can still serve the data

Diagram illustrating Topic-A with 2 partitions and a replication factor of 2. Broker 101 contains Partition 0. Broker 102 contains Partition 1 and Partition 0. Broker 103 contains Partition 1. A red circle with a diagonal line through it is placed over Broker 102, indicating it is down.

to ensure that data wouldn't be lost,

02:15 04:12 75%

4. Producer

Producers

- Producers write data to topics (which is made of partitions)
- Producers automatically know to which broker and partition to write to
- In case of Broker failures, Producers will automatically recover

Send data

Producer

Broker 101
Topic-A / Partition 0

Broker 102
Topic-A / Partition 1

Broker 103
Topic-A / Partition 2

The load is balanced to many brokers thanks to the number of partitions

writes

Producer receive acknowledgment of data: Like tell and ask pattern in Akka

0: Like tell, no wait receive message

1: Like ask, wait only leader reply received data

All: Like ask, wait leader and all ISR reply received data

Producers

- Producers can choose to receive acknowledgment of data writes:
 - acks=0: Producer won't wait for acknowledgment (possible data loss)
 - acks=1: Producer will wait for leader acknowledgment (limited data loss)
 - acks=all: Leader + replicas acknowledgment (no data loss)

Send data

Producer

Broker 101
Topic-A / Partition 0

Broker 102
Topic-A / Partition 1

Broker 103
Topic-A / Partition 2

Automatic load balance by broker

writes

Producer message keys

5. Producers and Message Keys.mp4 - VLC media player

Producers: Message keys

- Producers can choose to send a **key** with the message (string, number, etc..)
- If key=null, data is sent round robin (broker 101 then 102 then 103...)
- If a key is sent, then all messages for that key will always go to the same partition
- A key is basically sent if you need message ordering for a specific field (ex: truck_id)

Example:

truck_id_123 data will always be in partition 0
truck_id_234 data will always be in partition 0

truck_id_345 data will always be in partition 1
truck_id_456 data will always be in partition 1

Key is truck_id

(Advanced: we get this guarantee thanks to key hashing, which depends on the number of partitions)

5. Consumers

6. Consumers & Consumer Groups.mp4 - VLC media player

Consumers

- Consumers read data from a topic (identified by name)
- Consumers know which broker to read from
- In case of broker failures, consumers know how to recover
- Data is read in order **within each partitions**

Read in order

Read in order

There is no guarantee.

Divide work

6. Consumers & Consumer Groups.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Consumer Groups

© Stephane Maarek

- Consumers read data in consumer groups
- Each consumer within a group reads from exclusive partitions
- If you have more consumers than partitions, some consumers will be inactive

Topic-A Partition 0 Topic-A Partition 1 Topic-A Partition 2

Consumer 1 Consumer 2 consumer-group-application-1

Consumer 1 Consumer 2 Consumer 3 consumer-group-application-2

Consumer 1 consumer-group-3

• Note: Consumers will automatically use a GroupCoordinator and a ConsumerCoordinator to assign a consumers to a partition.

04:12 04:37

Number of partitions should equal or higher than number of customer

6. Consumers & Consumer Groups.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Consumer Groups

What if too many consumers?

© Stephane Maarek

- If you have more consumers than partitions, some consumers will be inactive

Topic-A Partition 0 Topic-A Partition 1 Topic-A Partition 2

Consumer 1 Consumer 2 Consumer 3 Consumer 4 (inactive)

consumer-group-application

You just have as many consumers as partitions at most.

04:02 04:37

6. Customer Offset

Like bookmarking when reading a book

7. Consumer Offsets & Delivery Semantics.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Consumer Offsets

© Stephane Maarek

- **Kafka** stores the offsets at which a consumer group has been reading
- The offsets committed live in a Kafka **topic** named **__consumer_offsets**
- When a consumer in a group has processed data received from Kafka, it should be committing the offsets
- If a consumer dies, it will be able to read back from where it left off thanks to the committed consumer offsets!

Committed offset

Consumer from Consumer Group

Reads

01:54 04:29

You can choose when consumer commit offset, have 3 delivery semantics:

7. Consumer Offsets & Delivery Semantics.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Delivery semantics for consumers

© Stephane Maarek

- Consumers choose when to commit offsets.
- There are 3 delivery semantics:
- **At most once:**
 - offsets are committed as soon as the message is received.
 - If the processing goes wrong, the message will be lost (it won't be read again).
- **At least once (usually preferred):**
 - offsets are committed after the message is processed.
 - If the processing goes wrong, the message will be read again.
 - This can result in duplicate processing of messages. Make sure your processing is idempotent (i.e. processing again the messages won't impact your systems)
- **Exactly once:**
 - Can be achieved for Kafka => Kafka workflows using Kafka Streams API
 - For Kafka => External System workflows, use an idempotent consumer.

04:10 04:29

7. Kafka Broker Discovery

8. Kafka Broker Discovery.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Kafka Broker Discovery

© Stephane Maarek

- Every Kafka broker is also called a “bootstrap server”
- That means that **you only need to connect to one broker**, and you will be connected to the entire cluster.
- Each broker knows about all brokers, topics and partitions (metadata)

```
graph LR
    KClient[Kafka Client] -- "1. Connection + Metadata request" --> B101[Broker 101 (bootstrap)]
    B101 -- "2. List of all brokers" --> KClient
    KClient -- "3. Can connect to the needed brokers" --> B101
    subgraph Kafka_Cluster [Kafka Cluster]
        B101
        B102[Broker 102 (bootstrap)]
        B103[Broker 103 (bootstrap)]
        B104[Broker 104 (bootstrap)]
        B105[Broker 105 (bootstrap)]
    end
```

01:00 03:23

8. Zookeeper

The Producer and the Consumer don't read/write on Zookeeper. They read/write to Kafka

8. Zookeeper.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Zookeeper

© Stephane Maarek

- Zookeeper manages brokers (keeps a list of them)
- Zookeeper helps in performing leader election for partitions
- Zookeeper sends notifications to Kafka in case of changes (e.g. new topic, broker dies, broker comes up, delete topics, etc....)
- **Kafka can't work without Zookeeper**
- Zookeeper by design operates with an odd number of servers (3, 5, 7)
- Zookeeper has a leader (handle writes) the rest of the servers are followers (handle reads)

01:20 02:40

9. Zookeeper.mpg - VLC media player

Zookeeper

The diagram illustrates the Zookeeper and Kafka architecture. At the top, three Zookeeper servers are shown: Zookeeper Server 1 (Follower), Zookeeper Server 2 (Leader), and Zookeeper Server 3 (Follower). They are connected by double-headed arrows, indicating a distributed system. Below the Zookeeper servers, five Kafka brokers are shown: Kafka Broker 1, Kafka Broker 2, Kafka Broker 3, Kafka Broker 4, and Kafka Broker 5. Lines connect Zookeeper Server 1 to Kafka Broker 1 and 2, Zookeeper Server 2 to Kafka Broker 3 and 4, and Zookeeper Server 3 to Kafka Broker 5. This shows that each Kafka broker is managed by a specific Zookeeper server.

We'll just be dealing with Kafka brokers

02:33 02:42

9. Kafka Guarantees

10. Kafka Guarantees.mpg - VLC media player

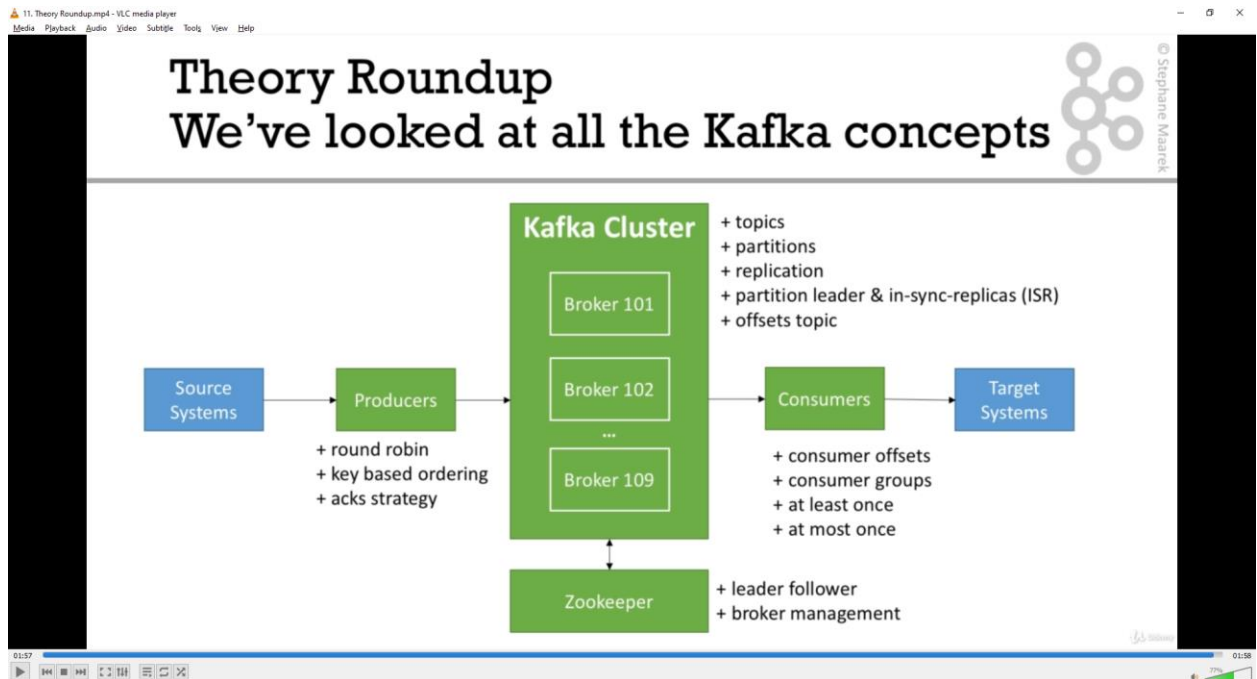
Kafka Guarantees

- Messages are appended to a topic-partition in the order they are sent
- Consumers read messages in the order stored in a topic-partition
- With a replication factor of N, producers and consumers can tolerate up to N-1 brokers being down
- This is why a replication factor of 3 is a good idea:
 - Allows for one broker to be taken down for maintenance
 - Allows for another broker to be taken down unexpectedly
- As long as the number of partitions remains constant for a topic (no new partitions), the same key will always go to the same partition

So super-important.

01:01 01:09

10. Roundup



11. All Kafka CMD for Windows:

Start zookeeper and kafka server

zookeeper-server-start.bat config\zookeeper.properties

kafka-server-start.bat config\server.properties

Create topic

kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic first_topic --create --partitions 3 --replication-factor 1

// first_topic = tên của topic, tên này là unique, identity

// --partitions 3 = chỉnh số partitions, không nhập sẽ báo lỗi, số partitions là tùy ý.

// --replication-factor 1 = chỉnh số replication, không nhập sẽ báo lỗi, nếu chỉ chạy 1 broker thì nên nhập 1

List

kafka-topics.sh --zookeeper 127.0.0.1:2181 --list

Detail

kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic first_topic --describe

Delete

kafka-topics.sh --zookeeper 127.0.0.1:2181 --topic first_topic --delete

Vào console producer

```
kafka-console-producer.sh --broker-list 127.0.0.1:9092 --topic first_topic
// sau khi vào mode, có thể gõ bất kỳ message nào, cứ enter 1 lần, thì message sẽ được gửi đi.
```

Truyền property

```
kafka-console-producer.sh --broker-list 127.0.0.1:9092 --topic first_topic --producer-property
acks=all
// Trường hợp nhập tên 1 topic mới, mà không được tạo trước đó, thì sẽ tự động tạo topic
vừa nhập (có WARN cảnh báo).
// Topic vừa nhập, có các thông số như partition, replication được chỉnh default như ở trong file
config/server.properties
```

Vào console consumer

```
kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic first_topic
// Như command trên, không nhập groupId, thì cli sẽ tự động tạo ra 1 groupId mới, unique
// Để show ra các message từ lúc begin (chưa được "mark as read"), thì truyền thêm param:
--from-beginning kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic
first_topic --from-beginning
```

Để set groupId cho consumer

```
kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic first_topic --group my-
first-application
// trường hợp các consumer cùng chung group, thì các consumer sẽ được nhận message từ
topic 1 cách lần lượt, (round robin)
// trường hợp các consumer khác group, thì với mỗi 1 message, tất cả các consumer đều nhận
được message như nhau.
// khi consumer đã nhận được message, thì offset trên mỗi partition được "mark as read" sẽ
thay đổi (offset lên giá trị gần nhất)
```

Sử dụng reset offset, để khi consumer load lại, có thể call lại các message từ offset chưa được "mark"

```
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group my-first-application --
reset-offsets --to-earliest --execute --topic first_topic
// còn các option khác như:
--to-datetime
--by-period
--to-earliest
--to-latest
--shift-by
--from-file
--to-current
```