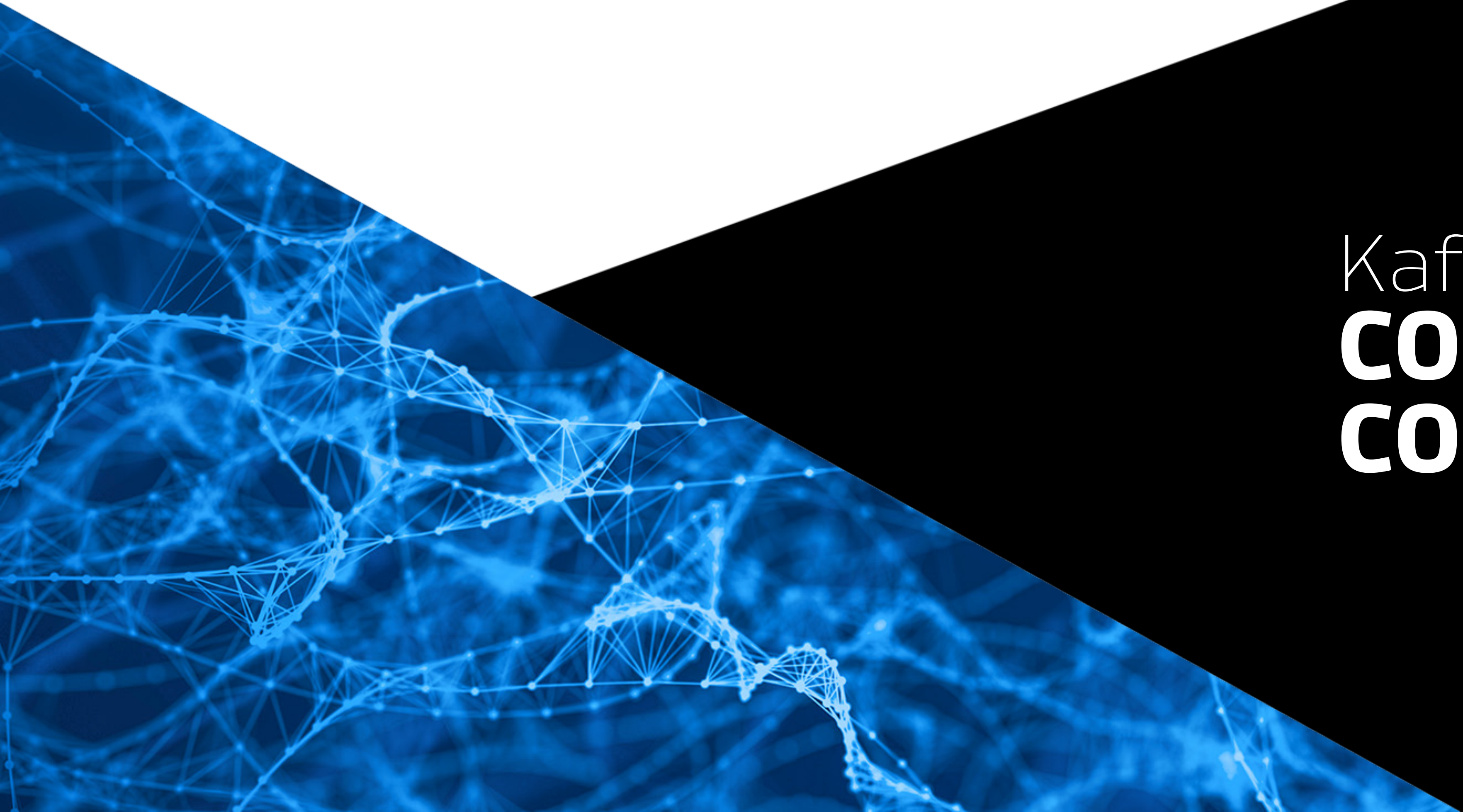
DEEPSHORE

Kafka: CONSUMER CONCEPT



The general Kafka design puts a lot of responsibility to its producers, but also to its consumers.

KAFKA CONSUMERS

Lets understand ...

- How to consume data from Kafka?
- Why using a consumer group?
- How to scale consumtion?
- Some Consumer configurations.

01

HOW TO CONSUME DATA FROM KAFKA

SET UP A CONSUMER…

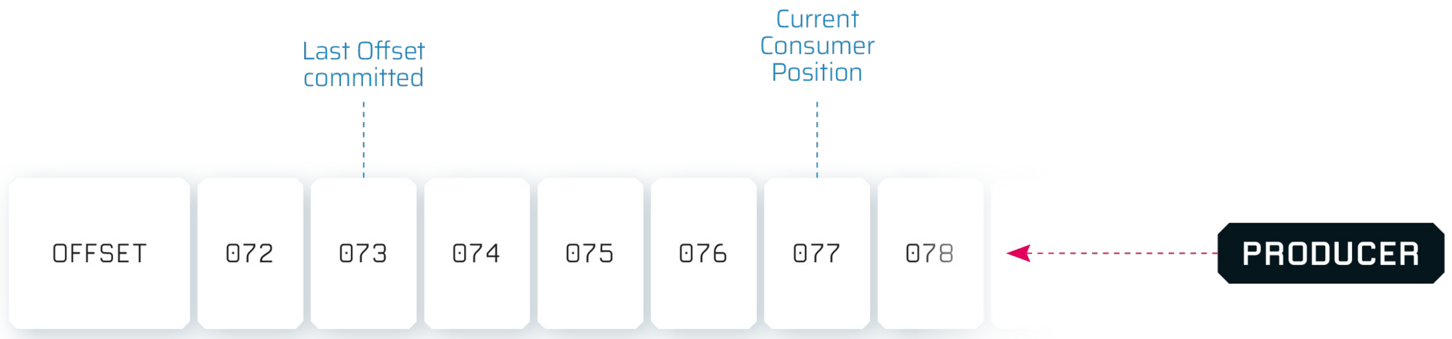
- Create a Java Properties instance (with some properties)…

- 3 mandatory Consumer properties:
  - bootstrap.servers (for cluster connection)
  - key.deserializer (first part of a Kafka record)
  - value.deserializer (second part of a Kafka record)

- Now you can subscribing to a Topic…

HOW A CONSUMER READS MESSAGES

- Calling a poll() will returns records from a topic to a consumer
- Kafka allows consumers to track their position (offset) in each partition.
- There is a default way of tracking which records were read by a consumer (enable.auto.commit=true).
- A consumer commit an offset back to a special _consumer_offsets topic.

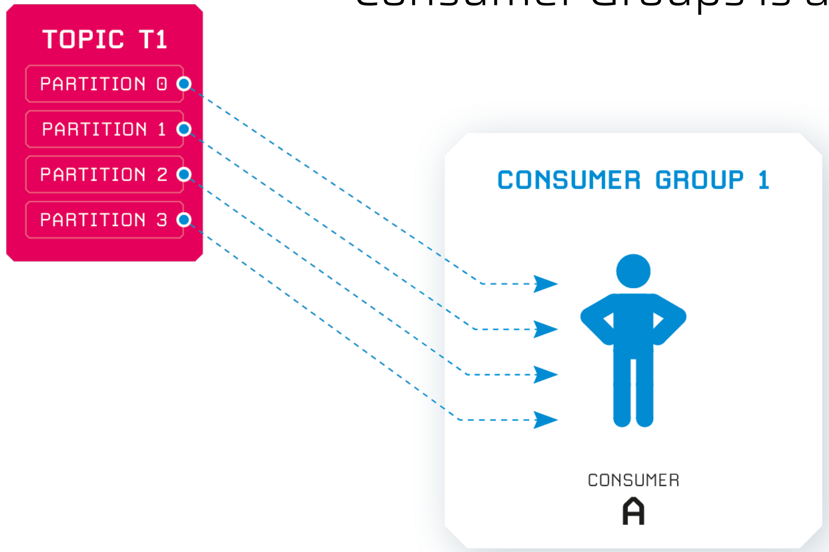


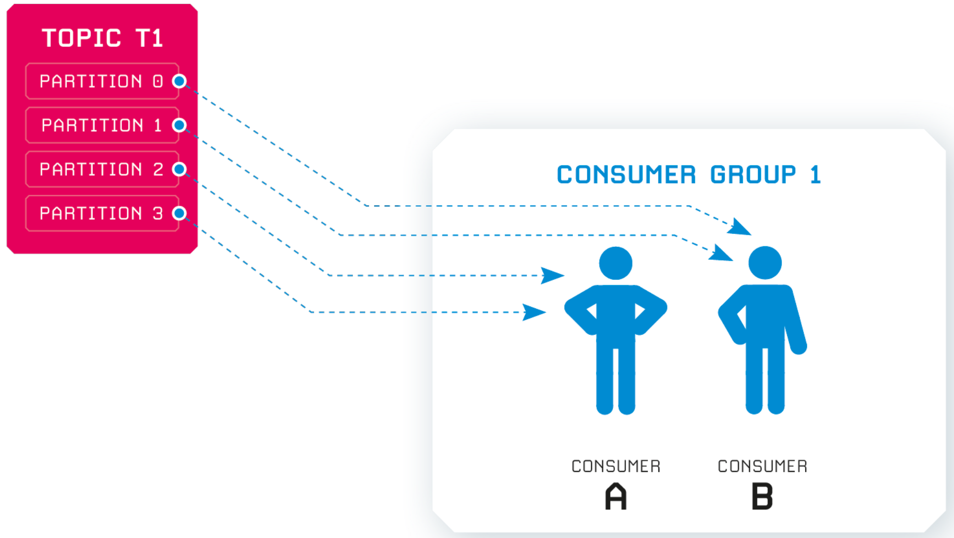
02

WHY USING A CONSUMER GROUP?

CONSUMER GROUPS

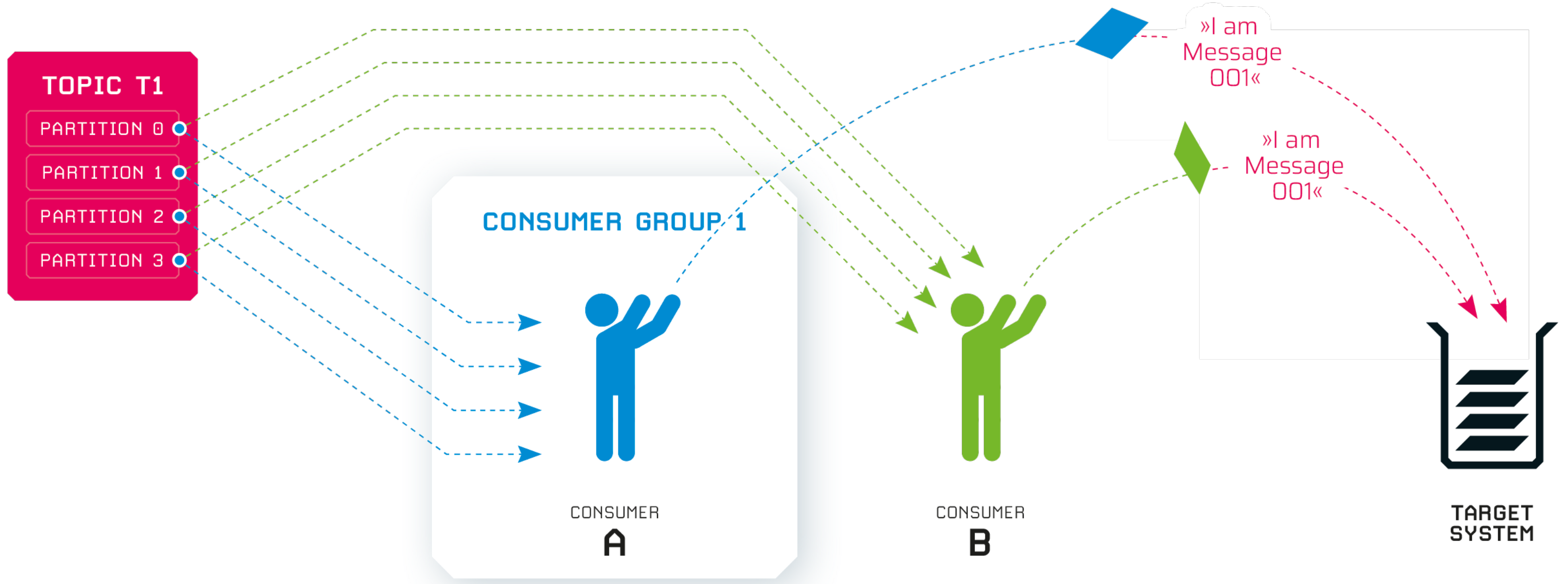
- A consumer group is the technical equivalent to a data sink (e.g. an application, target system, …).
- Each Kafka partition, will be consumed by exactly one consumer.
- Each Kafka partition, will also be consumed by exactly one consumer per consumer group.
- Consumer Groups is a concept for scaling consumption.





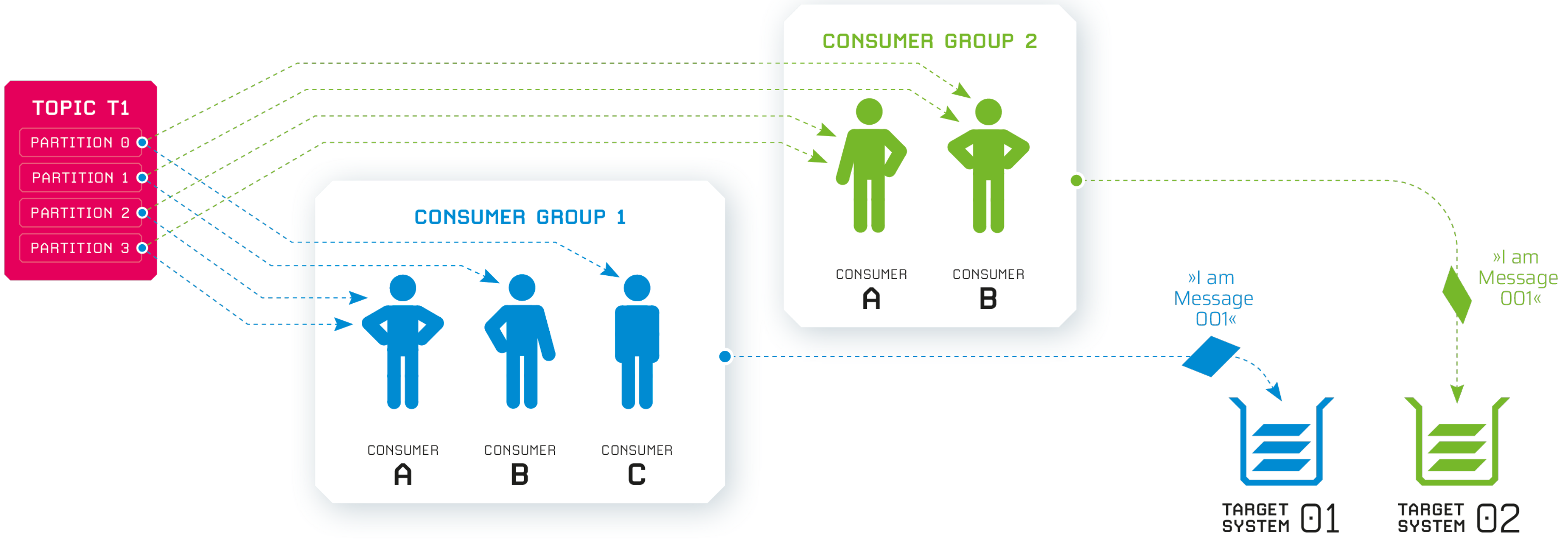
CONSUMER GROUPS

- Having two separated consumers for one technical data sink, would lead to duplicated messages in the target system.



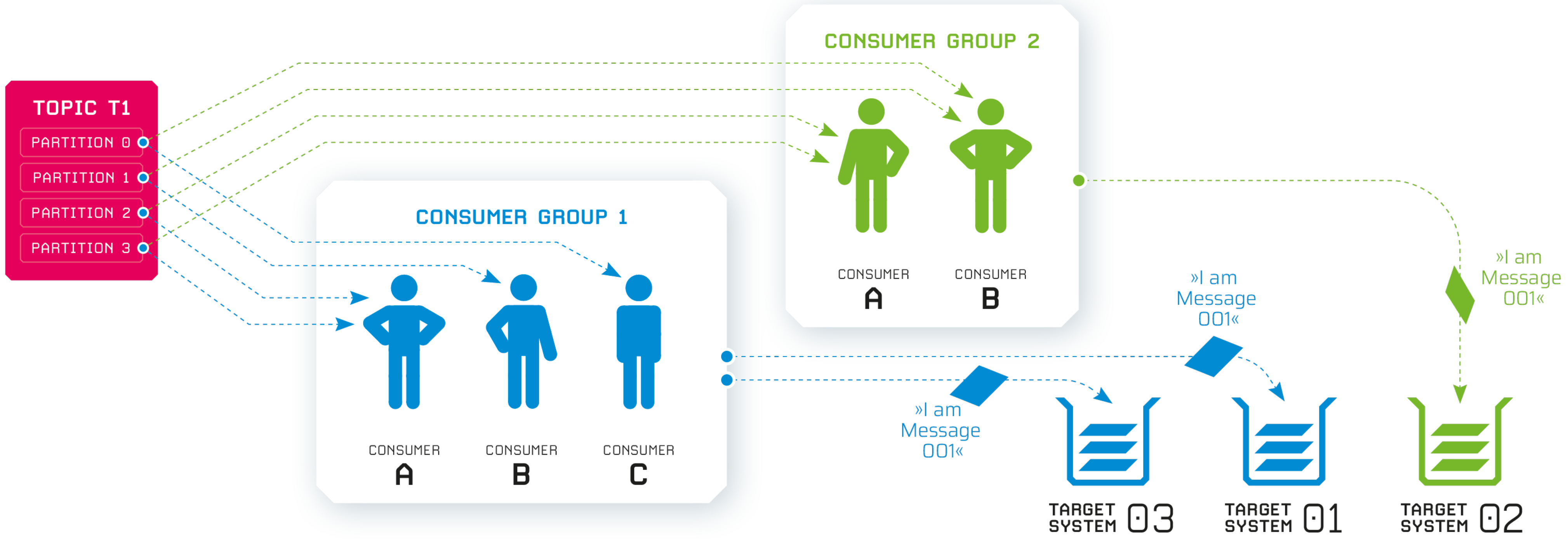
CONSUMER GROUPS

- Separated consumers (outside a group) or two consumer groups, are meant to be used for different sink-applications; not for scaling.



CONSUMER GROUPS

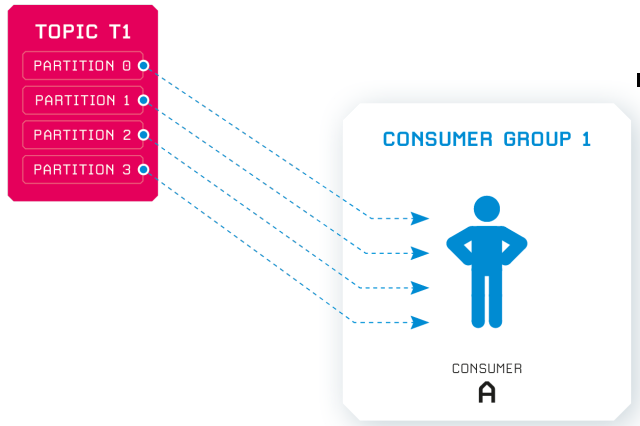
- Separated consumers (outside a group) or two consumer groups, are meant to be used for different sink-applications; not for scaling.

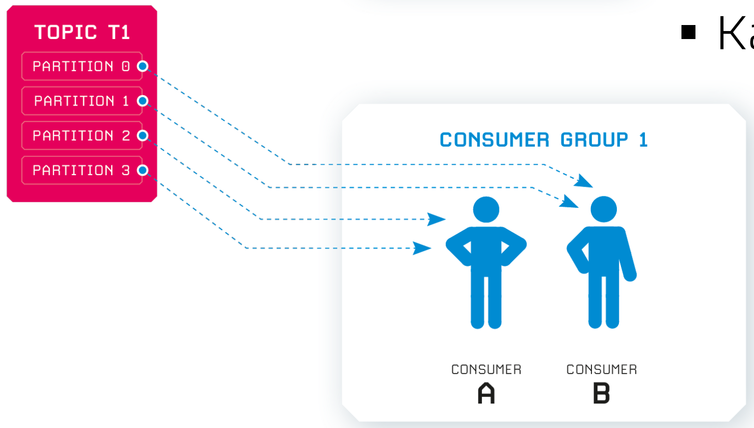


03

SCALING CONSUMER

SCALING WHITH CONSUMER GROUPS – GOOD TO KNOW





- To scale the workload on the consumer side: simply add an additional consumer to a group.
  - This can be done by registering a new consumer with an existing group.id (consumer configuration)
  - The new consumer will connect to the *Group Coordinator* of that existing group → see also: poll() loop
  - Kafka will manage the rest…

# Consumer properties example

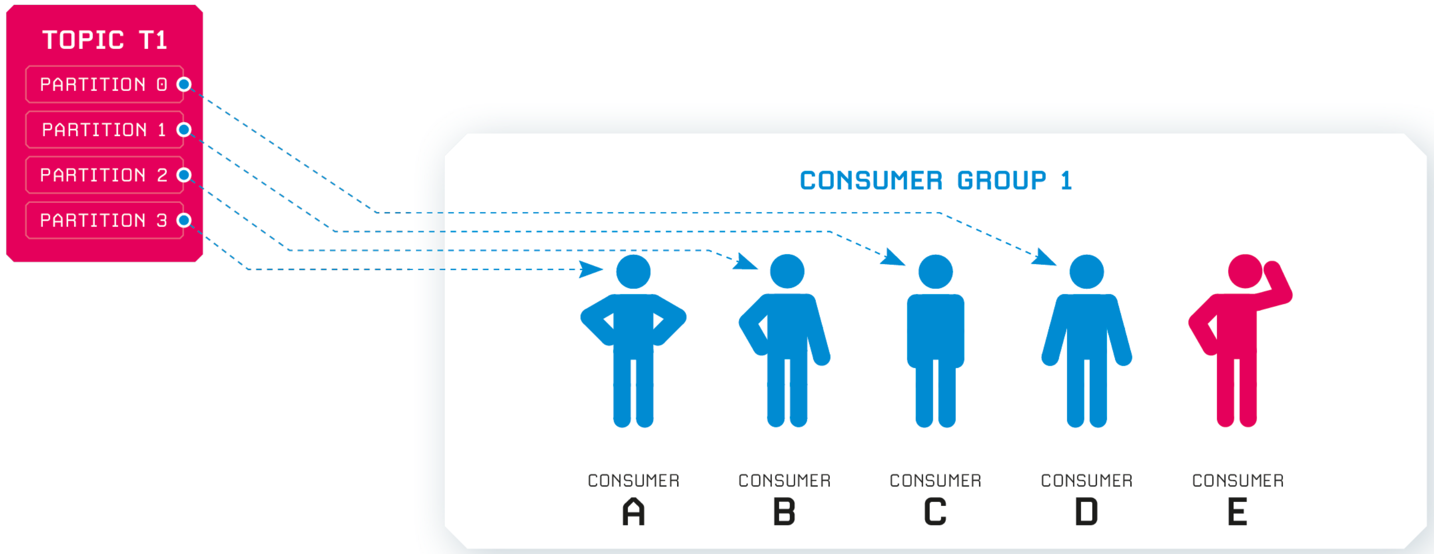
```
props.put("bootstrap.servers", "broker1:9092,broker2:9092");
props.put("group.id", “ExampleGroup");
props.put("key.deserializer","org.apache.kafka.common.serialization.
StringDeserializer");
props.put("value.deserializer","org.apache.kafka.common.serialization.
StringDeserializer");
```

SCALING WHITH CONSUMER GROUPS – GOOD TO KNOW

- Good to know… Adding a new consumer to an existing group…
  - The new consumer will start consuming messages from partitions previously consumed by another consumer(s)
  - moving partition ownership from one consumer to another is called a *rebalance*.
  - Rebalances provide the consumer group with higher availability and scalability (allowing to add and remove consumers)
  - During a rebalance, consumers cannot consume messages
  - A rebalance is a short window of unavailability of the entire consumer group.

SCALING WHITH CONSUMER GROUPS – GOOD TO KNOW

- Scaling to a number of consumers (per group) that is higher than the number of a topic´s partitions → makes no sense!
  - Remember: Each partition will be consumed by exactly one consumer



04

CONSUMER CONFIGURATIONS

CONSUMER – EXAMPLES OF CONFIGURATIONS

- Offset Configurations
- enable.auto.commit=true (default)
  → if you need End2End process guaranty and avoid duplicated messages, you might want to control that commit yourselfe or even store the offset outside Kafka
- auto.commit.interval.ms=5000 (5 Sec. default)
  → if enable.auto.commit=true, make shure that this config is fitting your use case
- auto.offset.reset=latest (default)
  → read the newest record first when reading a partition without a valid offset; „earliest“ would start with the oldest offset

CONSUMER – EXAMPLES OF CONFIGURATIONS

- Data Load/Network related Configurations
- fetch.max.wait.ms=500 (default)
  → how long the consumer will wait to poll()
- fetch.min.bytes=1 (default)
  → Kafka will wait until it has enough data before responding to the consumer (can be used to reduce network communication)
- max.poll.records=500 (default)
  → controlls the number of records per call
- max.partition.fetch.bytes= 1048576 (1MB default)
  → maximum number of bytes the server will return per partition

CONSUMER – EXAMPLES OF CONFIGURATIONS

- partition.assignment
- partition.assignment.strategy – will be done by the „group leader“
- By this, the consumer can decide how partitions are distributed between consumers of a group.
- (…).RangeAssignor – might distribute partitions more unbalanced (default) when having more than one topic per consumer group
- (…).RoundRobinAssignor – distributing number of partitions more evenly when having more than one topic per consumer group

CONSUMER – EXAMPLES OF CONFIGURATIONS

- session.timeout.ms=45.000 (45 Sec. default)
  → timeout before a consumer is considered to be offline (without a heartbeat); Note: the time out window on the Broker is configured with:
  - group.min.session.timeout.ms=6.000 (6 Sec. default)
  - group.max.session.timeout.ms=1.800.000 (30 Min default)
- heartbeat.interval.ms (3 Sec. default)
  → connection between a consumer and the group coordinator
- group.id → define your own consumer group

05

QUIZ

QUIZ – Example 1

Given:

- Topic with 2 partitions and a group with 2 consumers; Both have…
- enable.auto.commit=true
- auto.commit.interval.ms=5 Sec.
- Poll() every 3 Sec

- Then one Consumer looses connection for at least 46 sec. …
- What will happen?
- What can happen?

QUIZ – Example 2

Given:

- Topic with 2 partitions and a group with 1 consumer
- enable.auto.commit=true
- auto.commit.interval.ms=5 Sec.
- Poll() every 3 Sec

- Then another Consumer joins the group…
- What will happen?
- What can happen?

Who said:

CONSUMER CONCEPT IS EASY?

CONTACT

Deepshore GmbH · Van-der-Smissen-Straße 9, 22767 Hamburg
Telefon +49 40 46664-296 · Fax +49 40 46664-299
E-Mail info@deepshore.de · **www.deepshore.de**