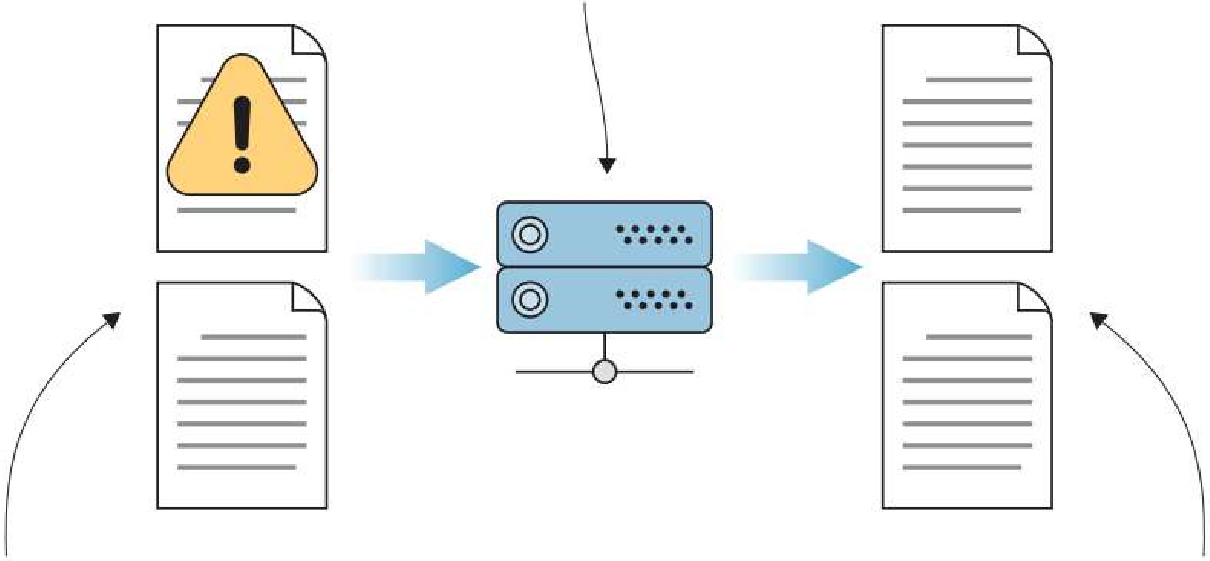
Kafka

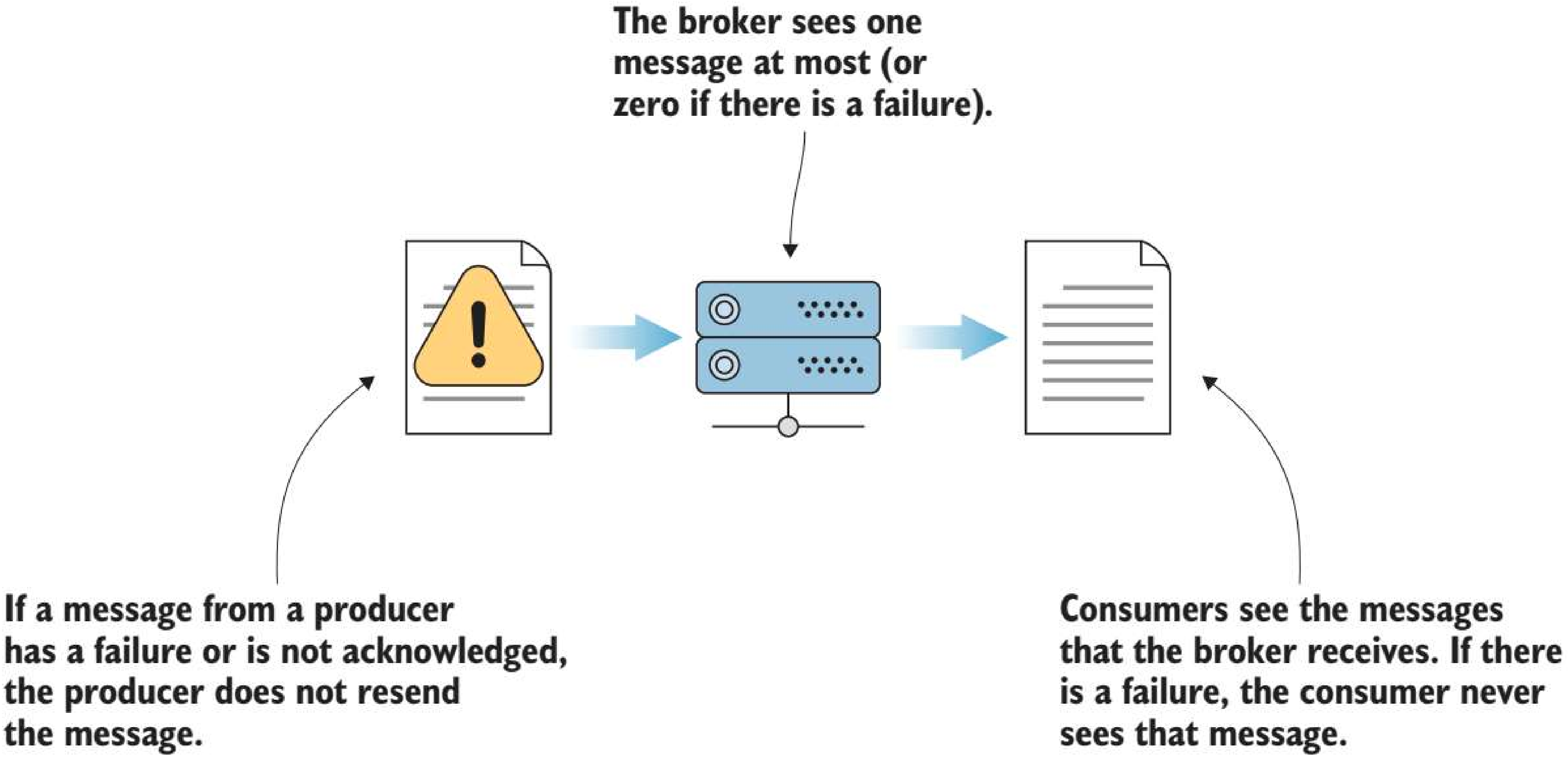
The broker sees two messages at least once (or only one if there is a failure).



If a message from a producer has a failure or is not acknowledged, the producer resends the message.

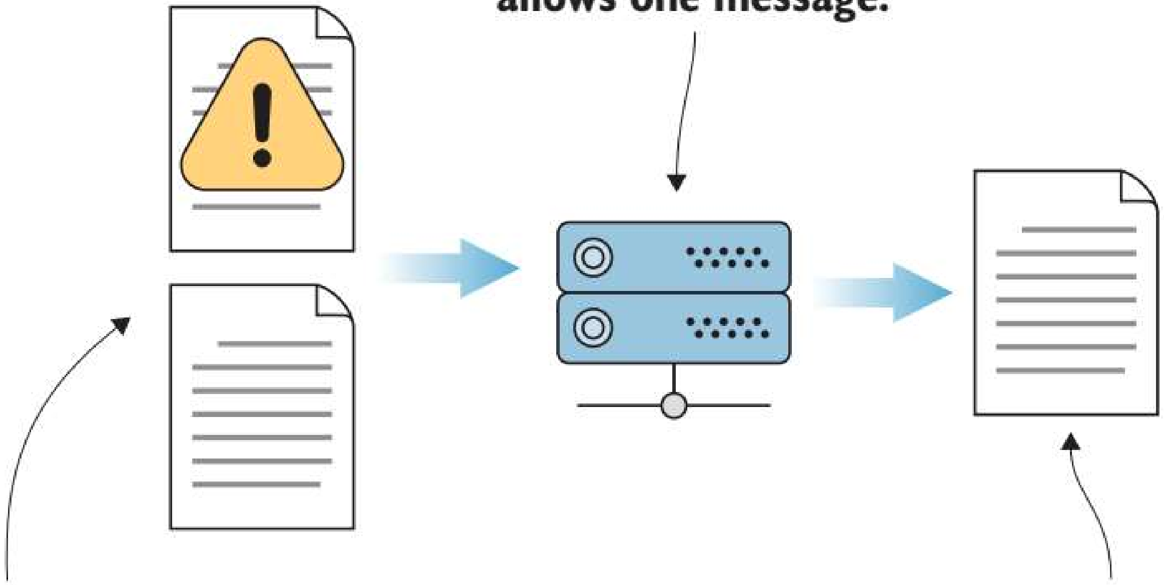
Consumers get as many messages as the broker receives. Consumers might see duplicate messages.

At-least-once message flow


The broker sees one message at most (or zero if there is a failure).
If a message from a producer has a failure or is not acknowledged, the producer does not resend the message.
Consumers see the messages that the broker receives. If there is a failure, the consumer never sees that message.

At-most-once message flow

The broker only allows one message.

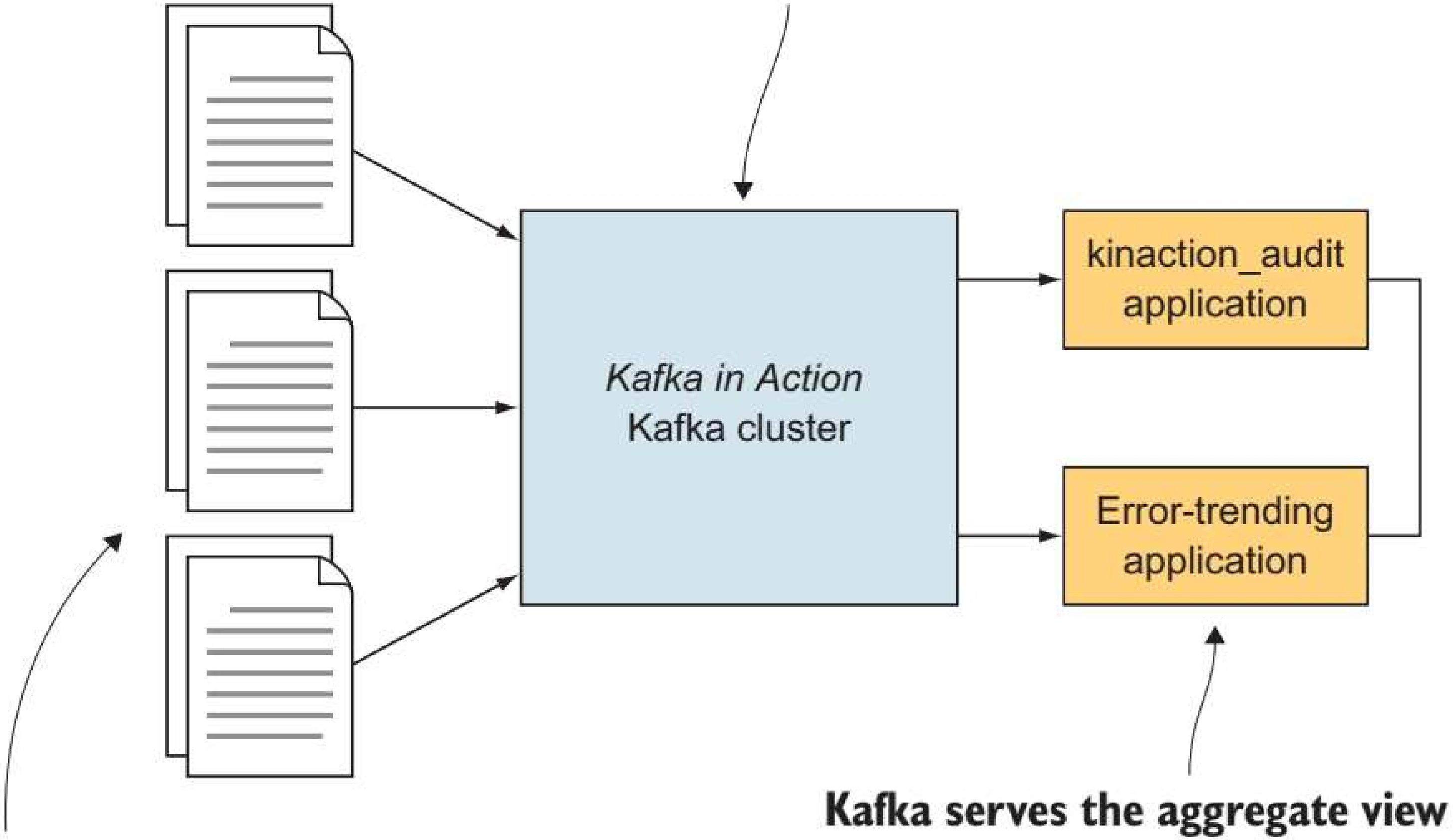


If a message from a producer fails or is not acknowledged, the producer resends the message.

Consumers only see the message once.

Exactly-once message flow

Kafka acts as a logical central point for all of the server logs and stores that information on the brokers.

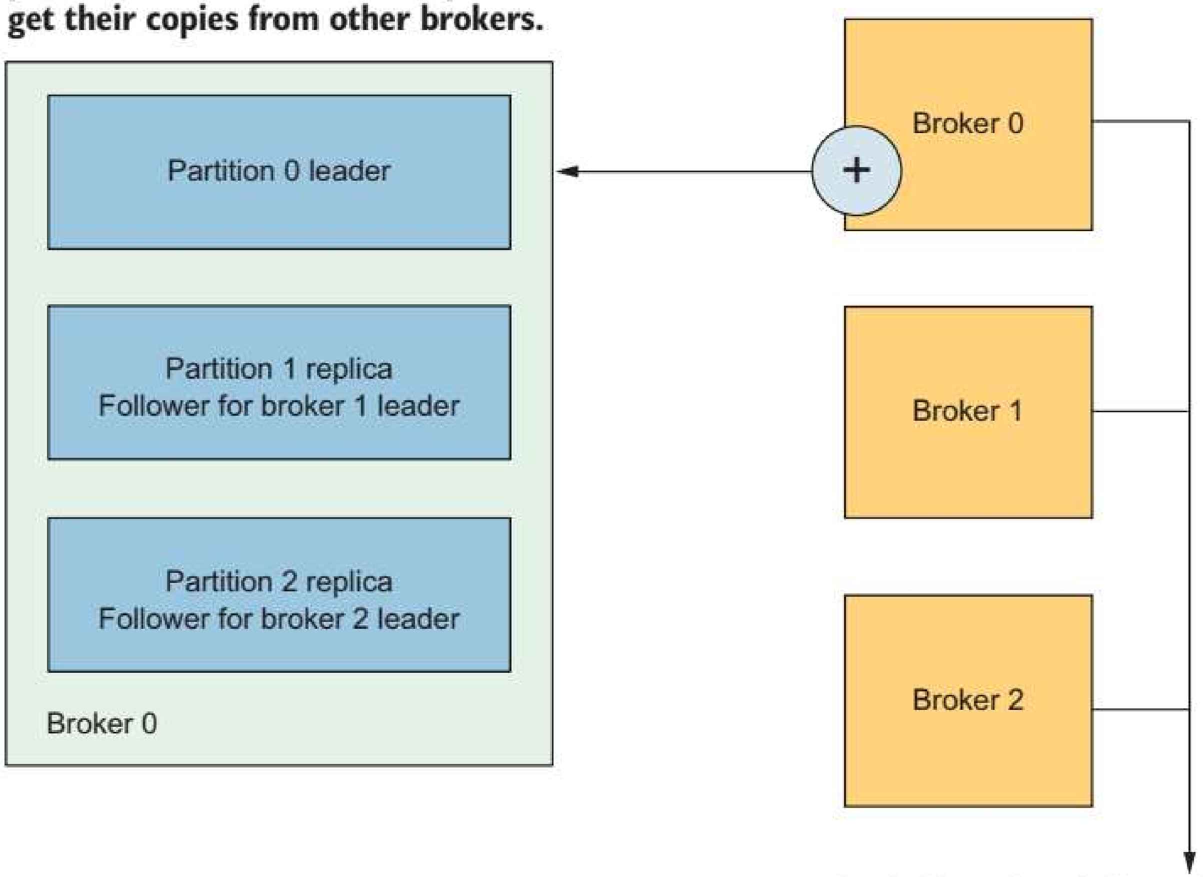


Various server logs are gathered into Kafka.

Kafka serves the aggregate view to each application (assuming each application is part of its own group).

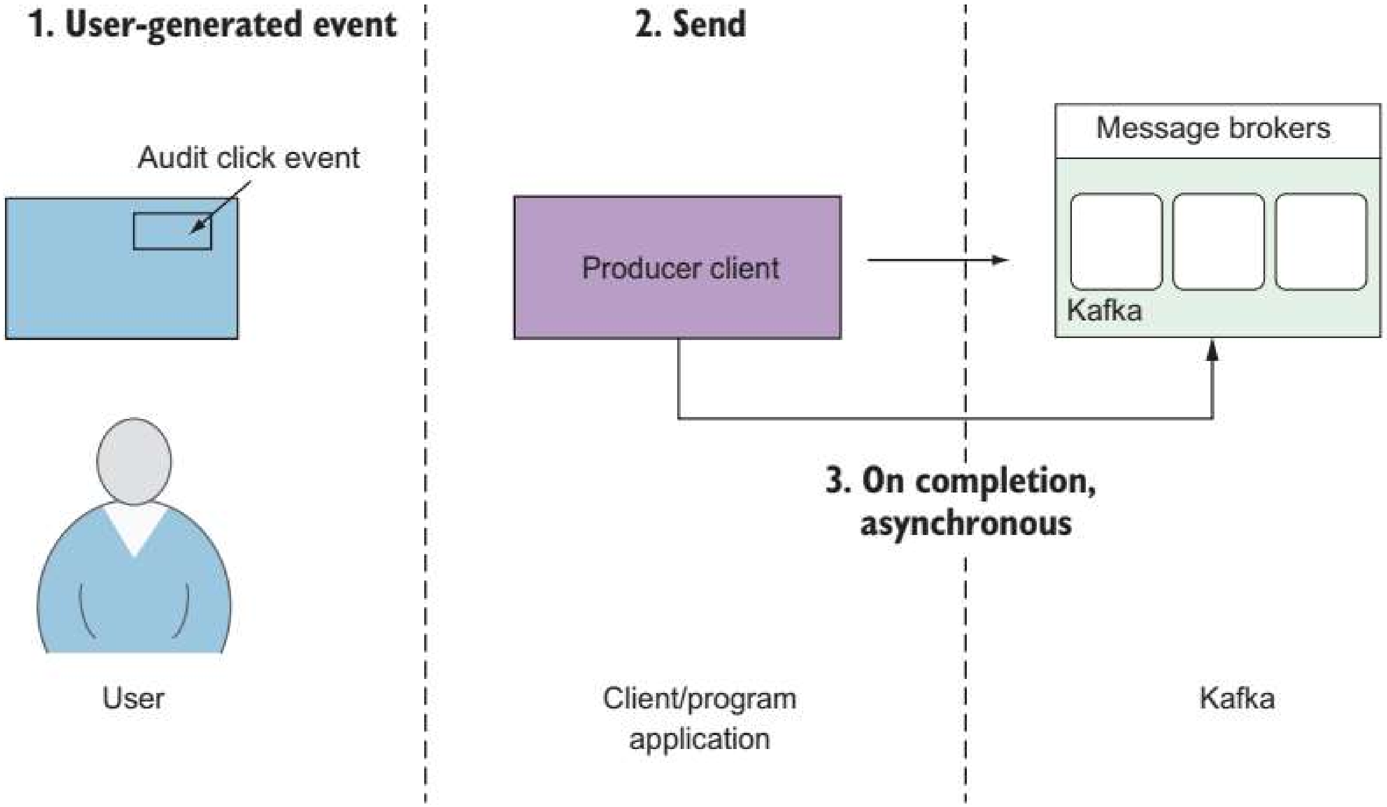
Kafka log aggregation

Broker 0 only reads and writes for partition 0. The rest of the replicas get their copies from other brokers.

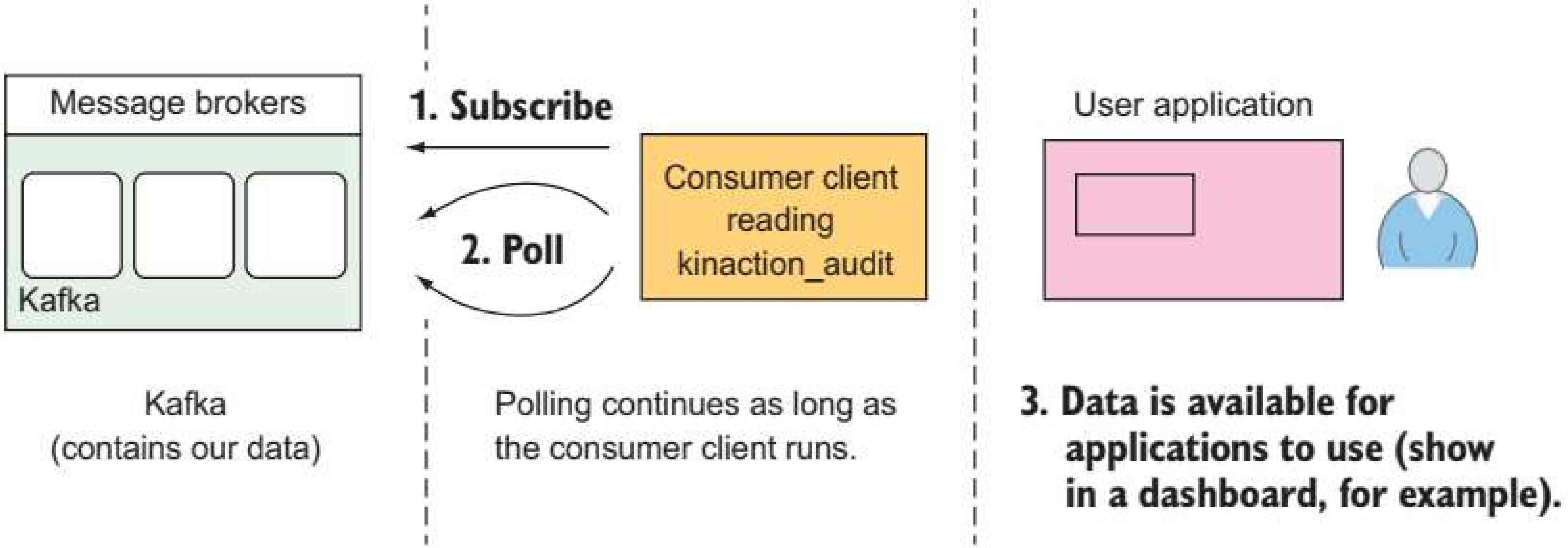


Topic kinaction_helloworld is actually made up of the leaders of each partition. In our case, that involves each broker holding a partition leader.

View of one broker



Producer example for user event



Message brokers

Kafka

1. **Subscribe**

2. **Poll**

Consumer client reading kinaction_audit

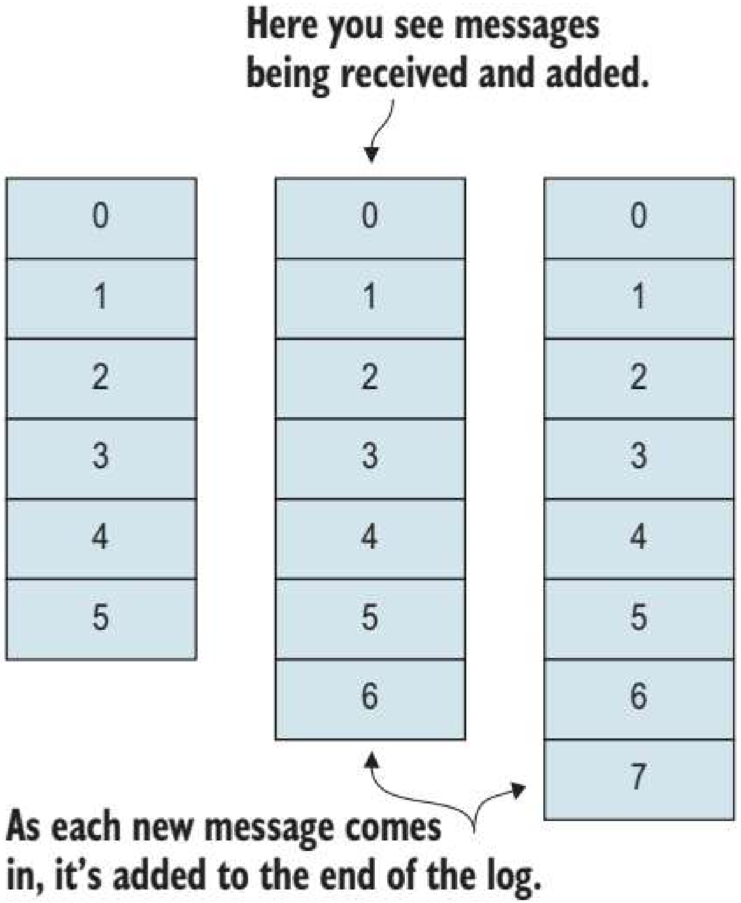
User application

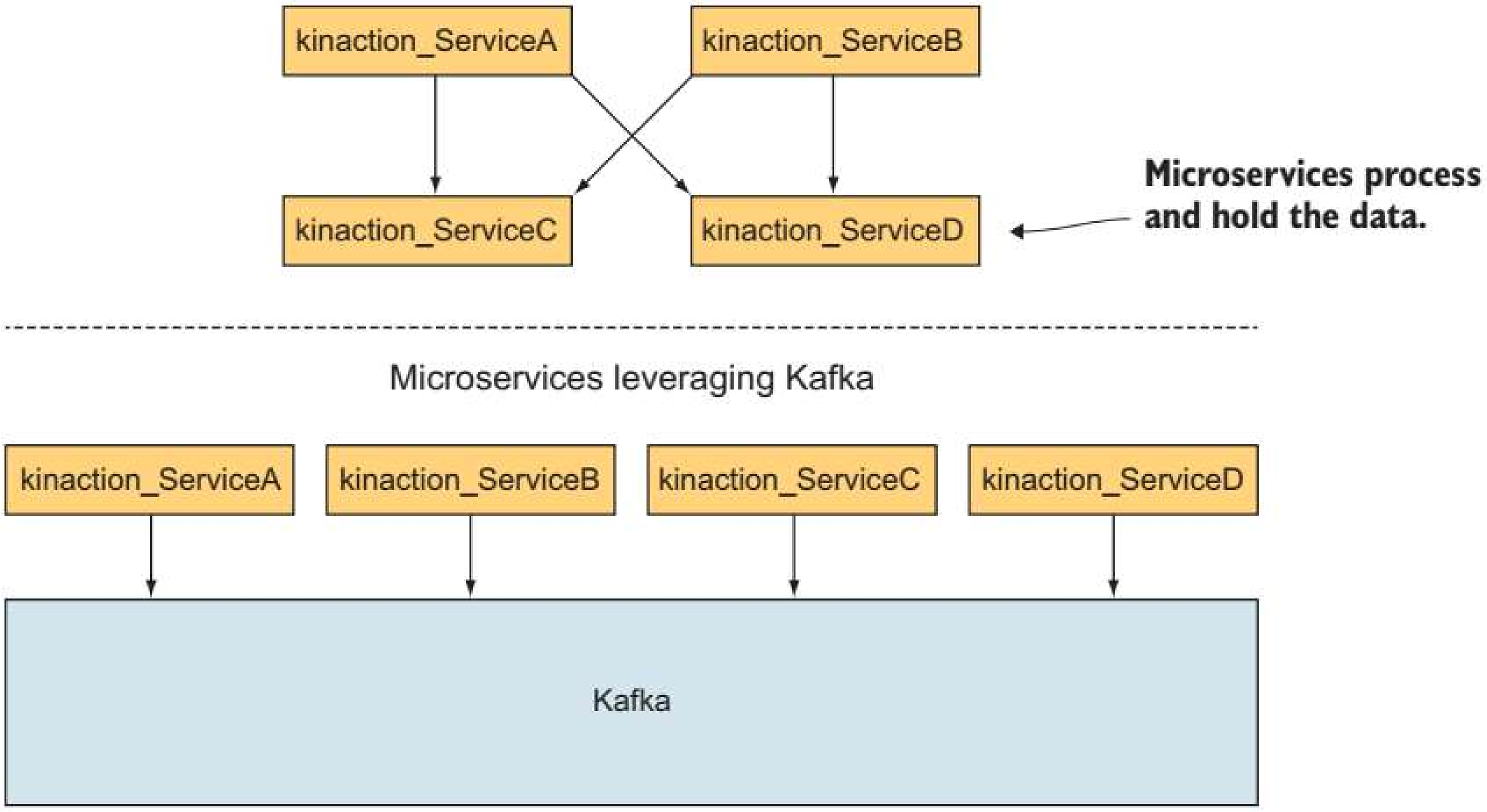
Kafka (contains our data)

Polling continues as long as the consumer client runs.

3. **Data is available for applications to use (show in a dashboard, for example).**

Consumer example flow

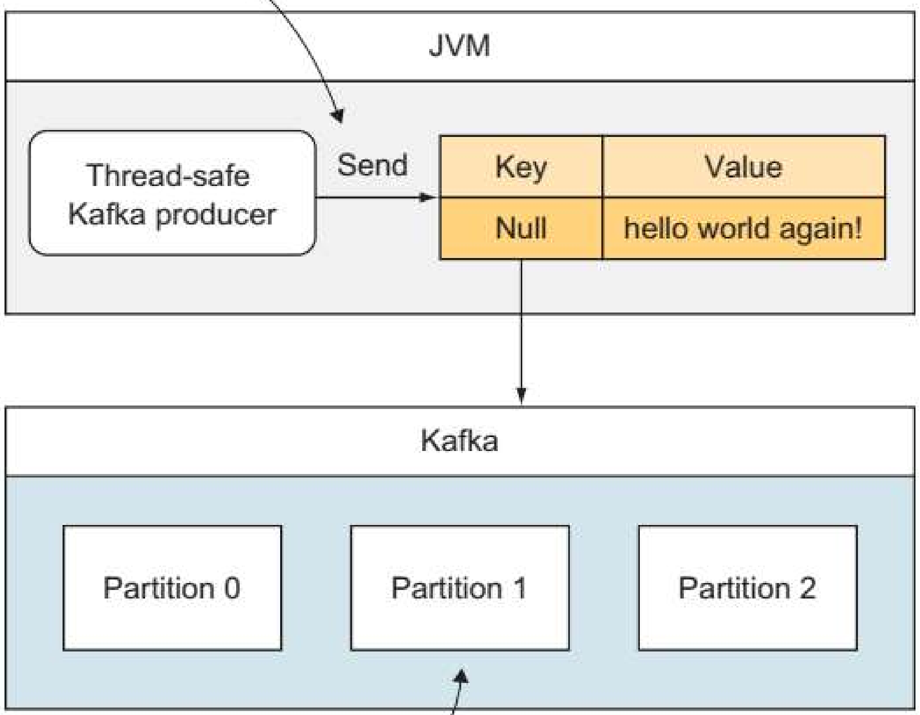

Here you see messages being received and added.
0
1
2
3
4
5
0
1
2
3
4
5
6
0
1
2
3
4
5
6
7
As each new message comes in, it's added to the end of the log.



Microservices process and hold the data.

Using Kafka Streams, you can share data, while processing is independent.

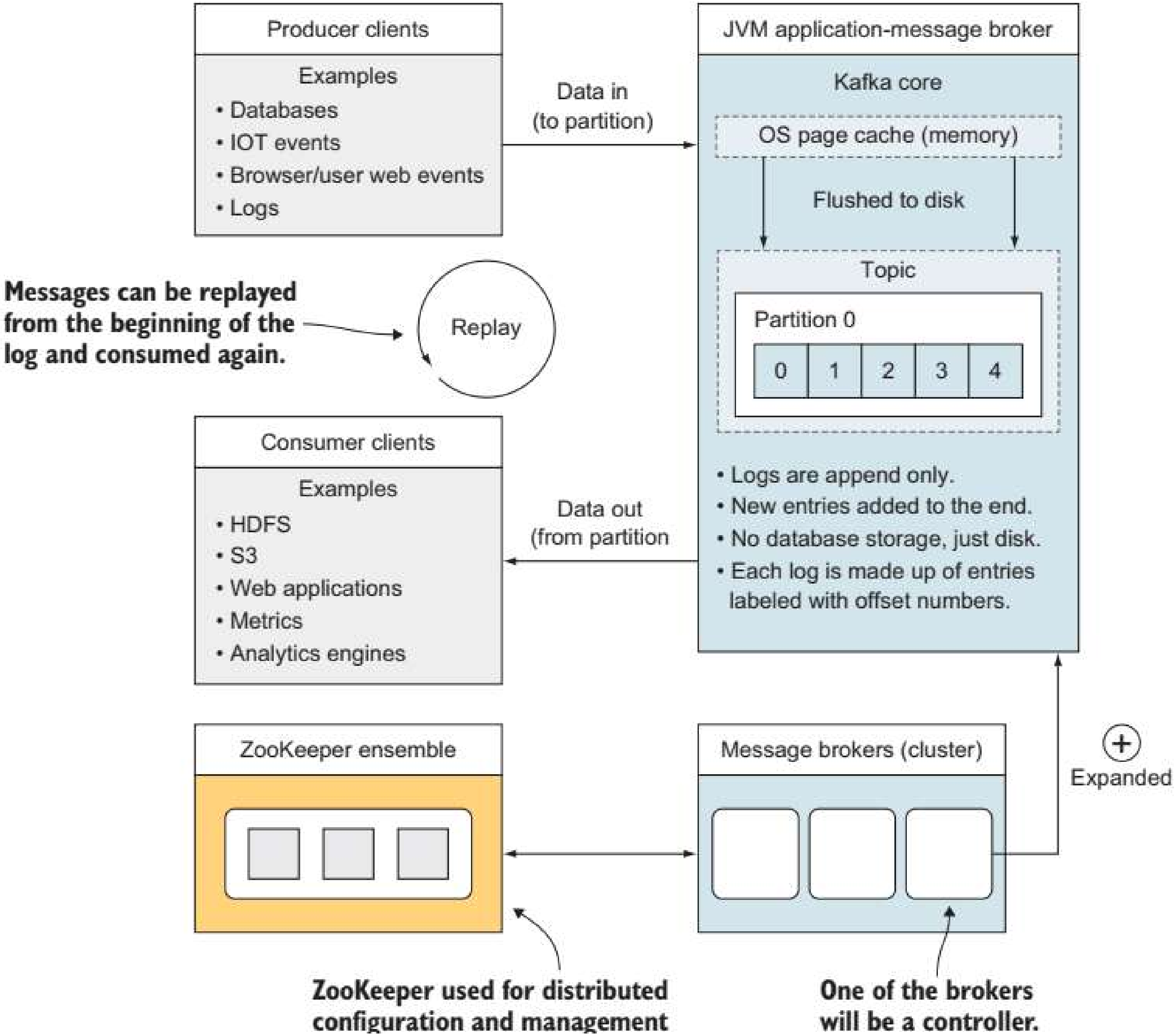
Producer record sent to topic kinaction_helloworld



Key	Value
Null	hello world again!

Producer flow

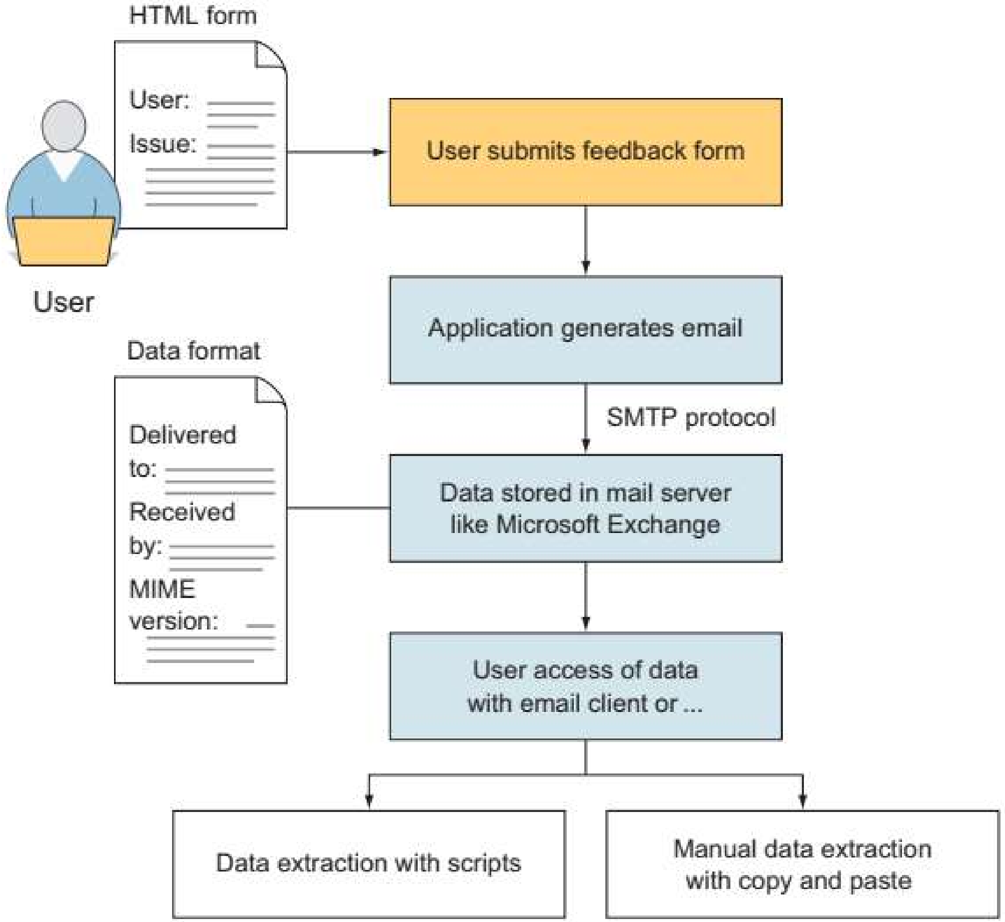
The call to send has already figured out which partition the producer record will be written to, although it is not defined in your client code explicitly. In this example, it is assigned to partition 1.



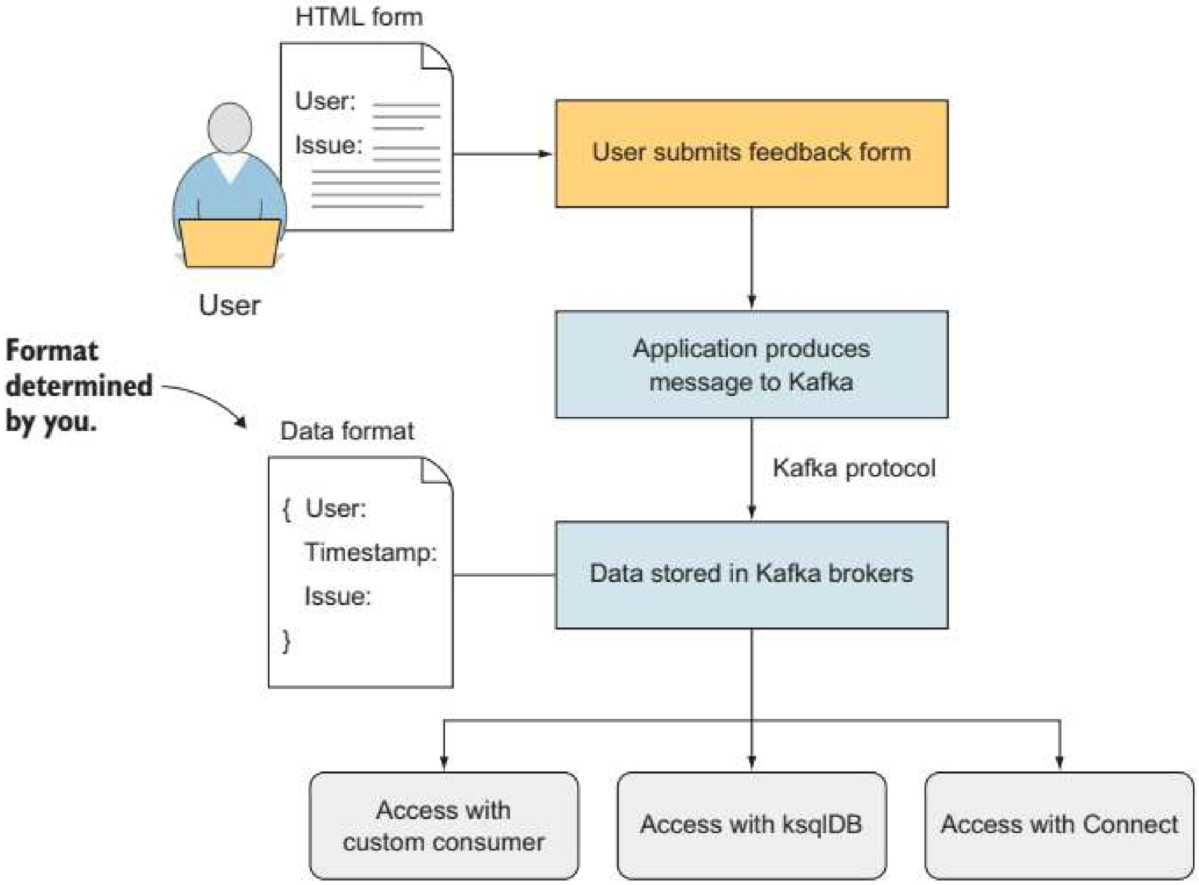
Messages can be replayed from the beginning of the log and consumed again.

ZooKeeper used for distributed configuration and management

One of the brokers will be a controller.


HTML form
User:
Issue:
User
User submits feedback form
Application generates email
SMTP protocol
Data format
Delivered to:
Received by:
MIME version:
Data stored in mail server like Microsoft Exchange
User access of data with email client or ...
Data extraction with scripts
Manual data extraction with copy and paste

Sending data in email

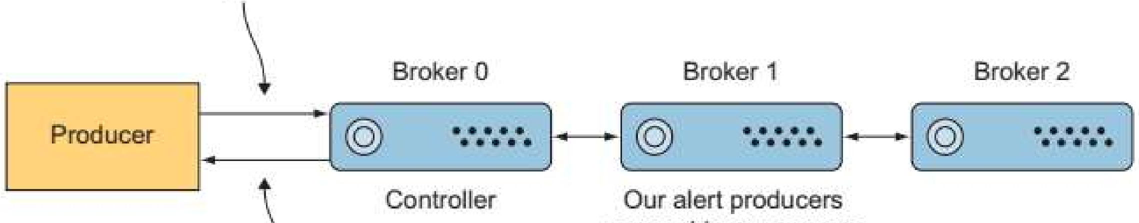

HTML form
User:
Issue:
User
User submits feedback form
Application produces message to Kafka
Kafka protocol
Format determined by you.
Data format
{ User:
Timestamp:
Issue:
}
Data stored in Kafka brokers
Access with custom consumer
Access with ksqlDB
Access with Connect

Sending data to Kafka

Important producer configurations

Key	Purpose
`acks`	Number of replica acknowledgments that a producer requires before success is established
`bootstrap.servers`	One or more Kafka brokers to connect for startup
`value.serializer`	The class that's used for serialization of the value
`key.serializer`	The class that's used for serialization of the key

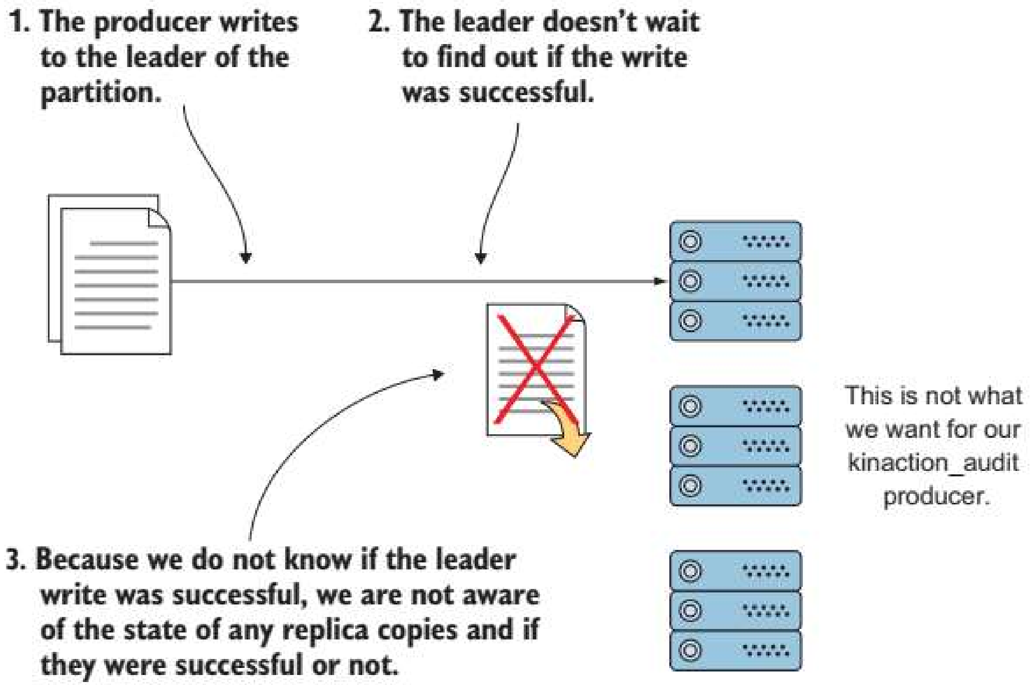
1. Producer connects to bootstrap servers



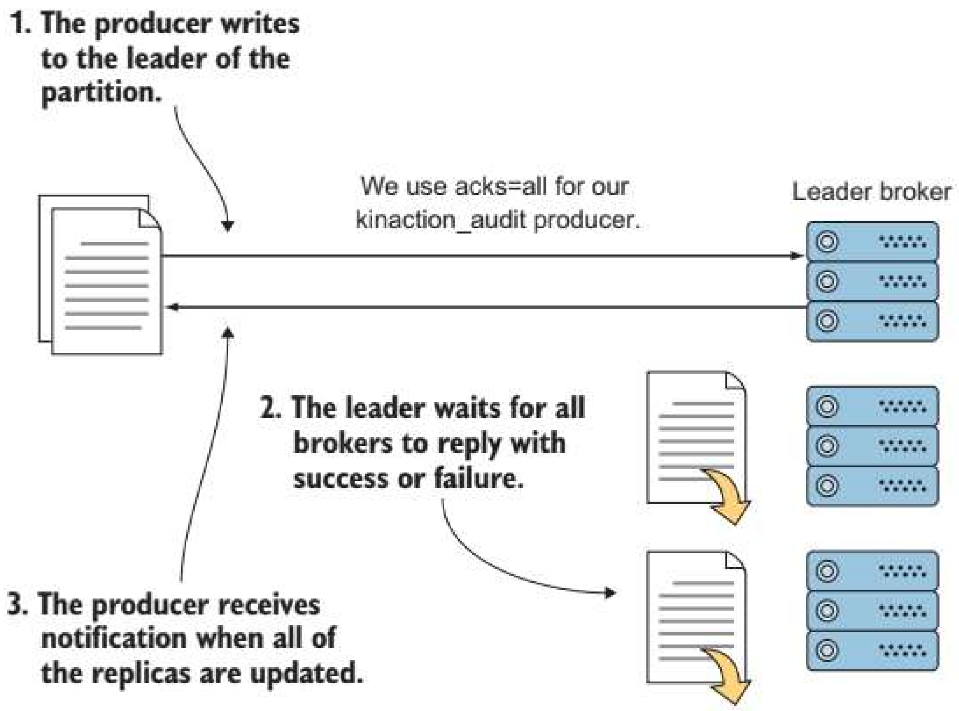
Our alert producers connect to our servers since they are local on different ports on localhost.

2. Metadata sent back to producer letting it know its leader resides on Broker 2, which it did not know about at first. Kafka knows about its other brokers.

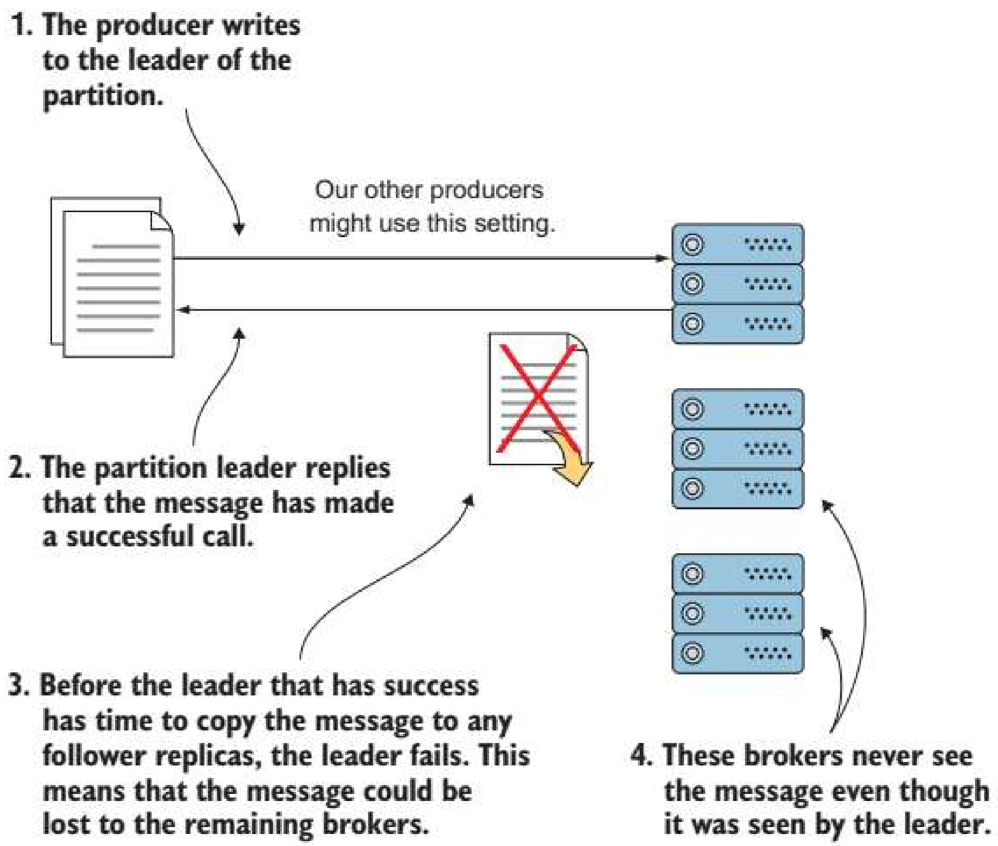
Bootstrap servers


1. The producer writes to the leader of the partition.
2. The leader doesn't wait to find out if the write was successful.
3. Because we do not know if the leader write was successful, we are not aware of the state of any replica copies and if they were successful or not.
This is not what we want for our kinaction_audit producer.

The property `acks` equals `0`.


1. The producer writes to the leader of the partition.
We use acks=all for our kinaction_audit producer.
Leader broker
2. The leader waits for all brokers to reply with success or failure.
3. The producer receives notification when all of the replicas are updated.

The property `acks` equals `all`.



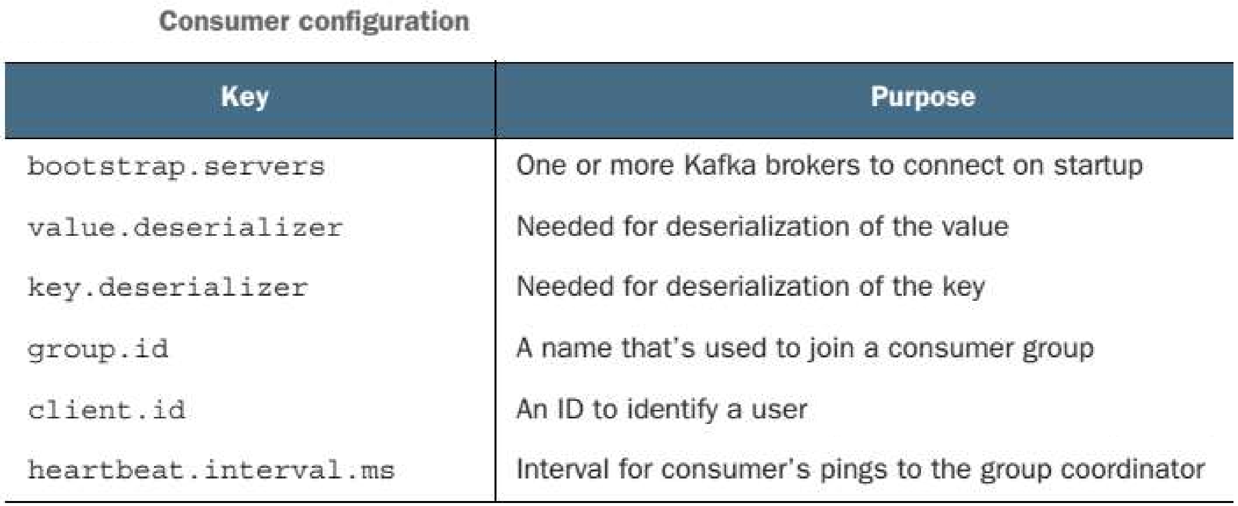
1. The producer writes to the leader of the partition.

2. The partition leader replies that the message has made a successful call.

3. Before the leader that has success has time to copy the message to any follower replicas, the leader fails. This means that the message could be lost to the remaining brokers.

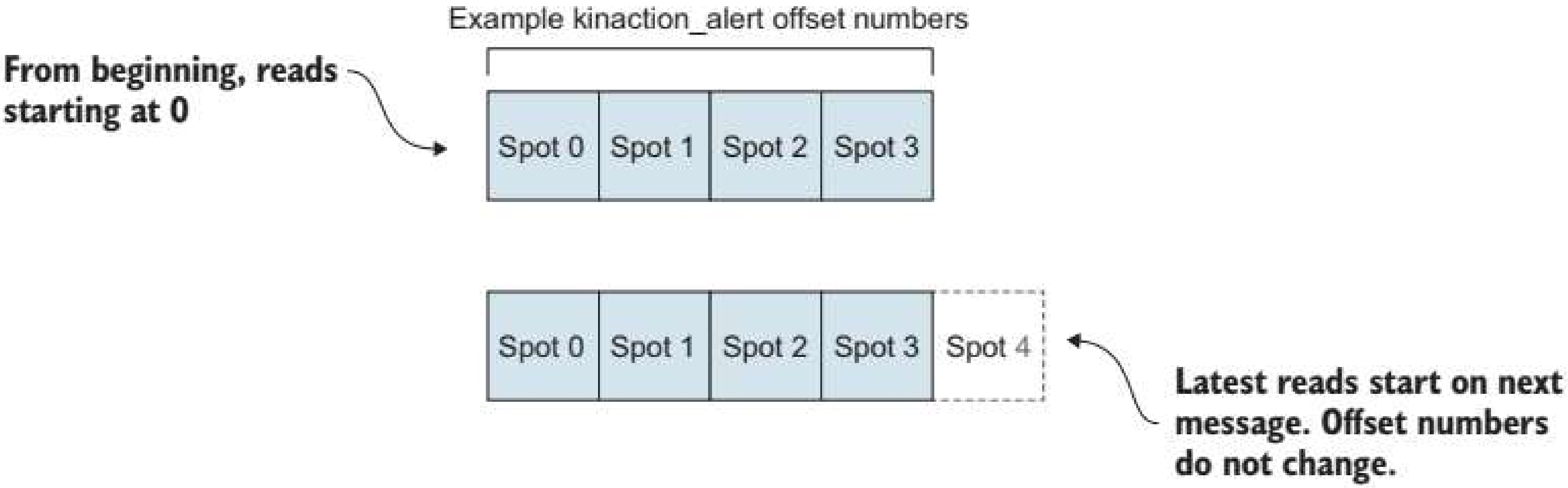
4. These brokers never see the message even though it was seen by the leader.

The property `acks` equals `1`.

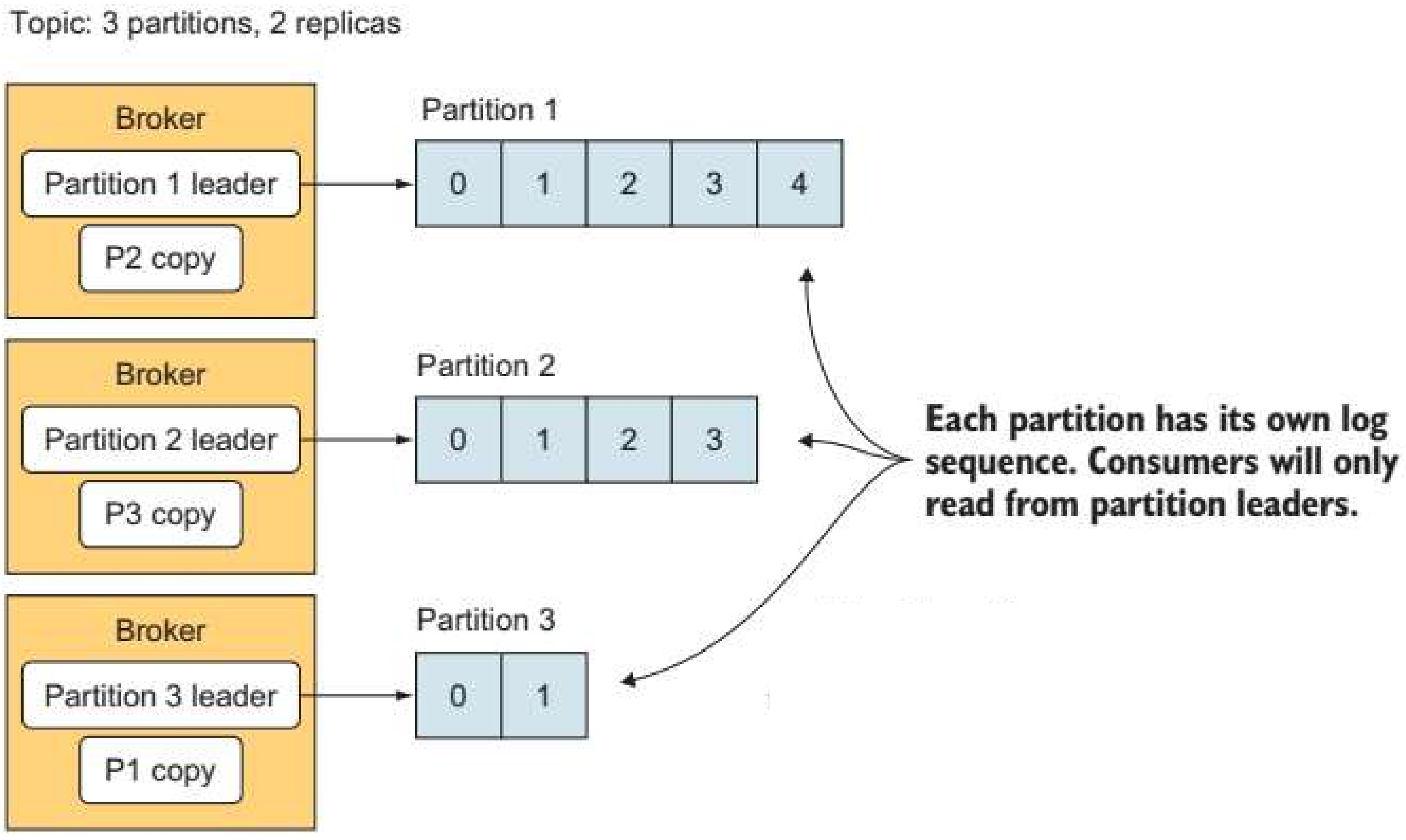


Consumer configuration

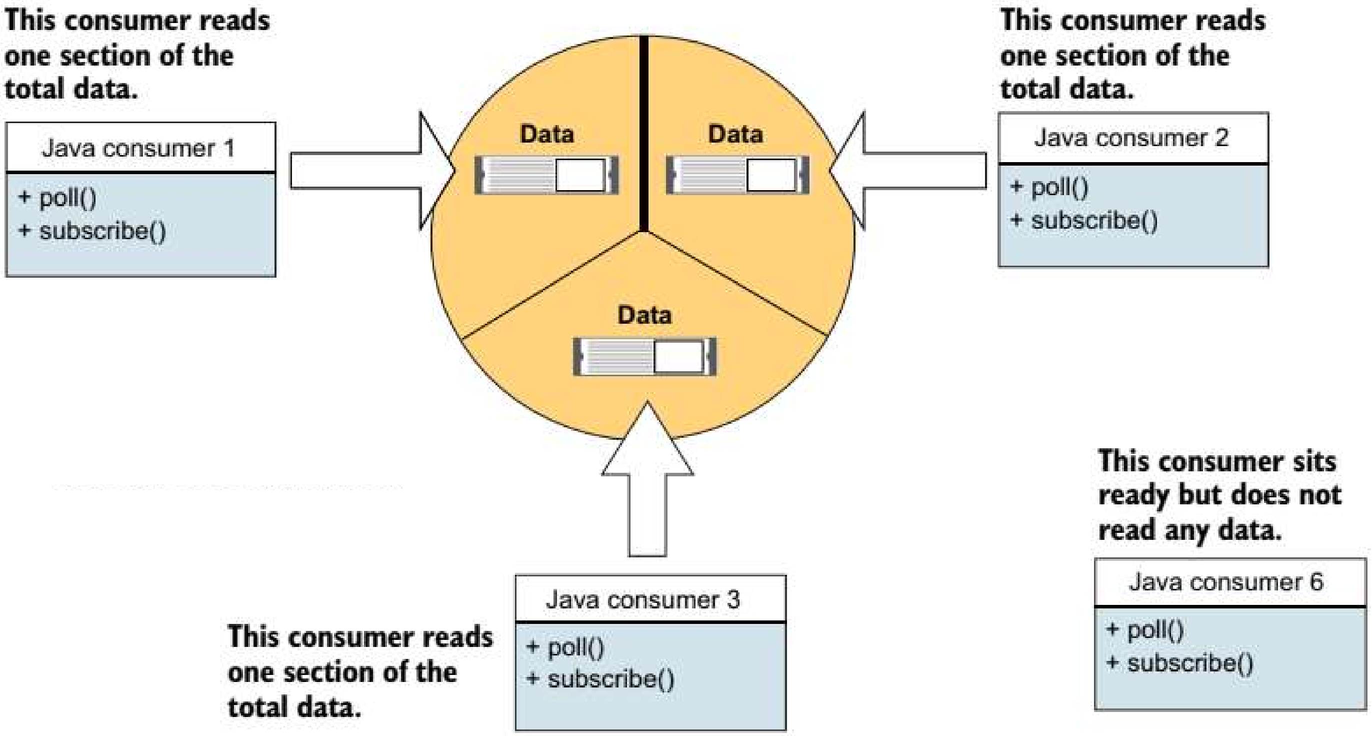
Key	Purpose
`bootstrap.servers`	One or more Kafka brokers to connect on startup
`value.deserializer`	Needed for deserialization of the value
`key.deserializer`	Needed for deserialization of the key
`group.id`	A name that's used to join a consumer group
`client.id`	An ID to identify a user
`heartbeat.interval.ms`	Interval for consumer's pings to the group coordinator


Example kinaction_alert offset numbers
From beginning, reads starting at 0
Spot 0
Spot 1
Spot 2
Spot 3
Spot 0
Spot 1
Spot 2
Spot 3
Spot 4
Latest reads start on next message. Offset numbers do not change.

Kafka offsets [6]



Partition leaders



This consumer reads one section of the total data.

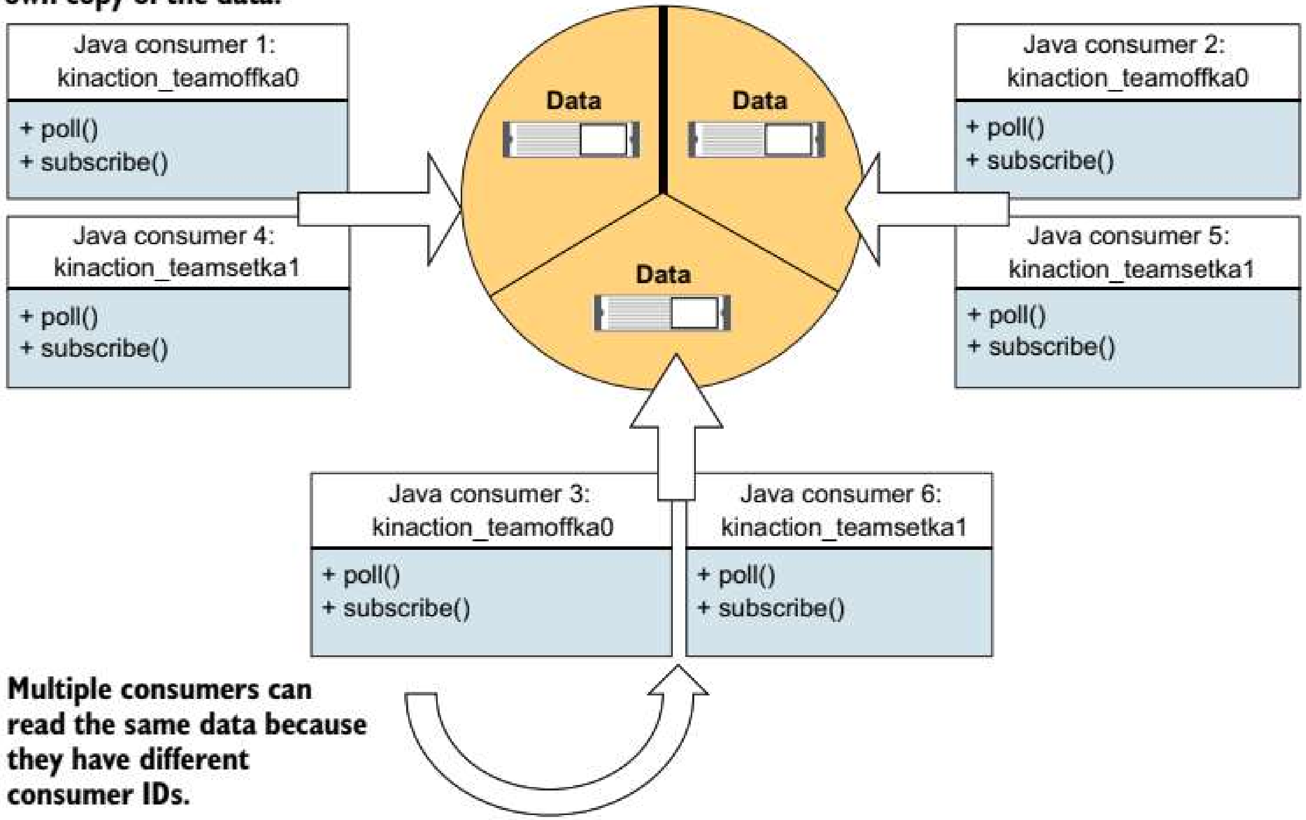
This consumer reads one section of the total data.

This consumer reads one section of the total data.

This consumer sits ready but does not read any data.

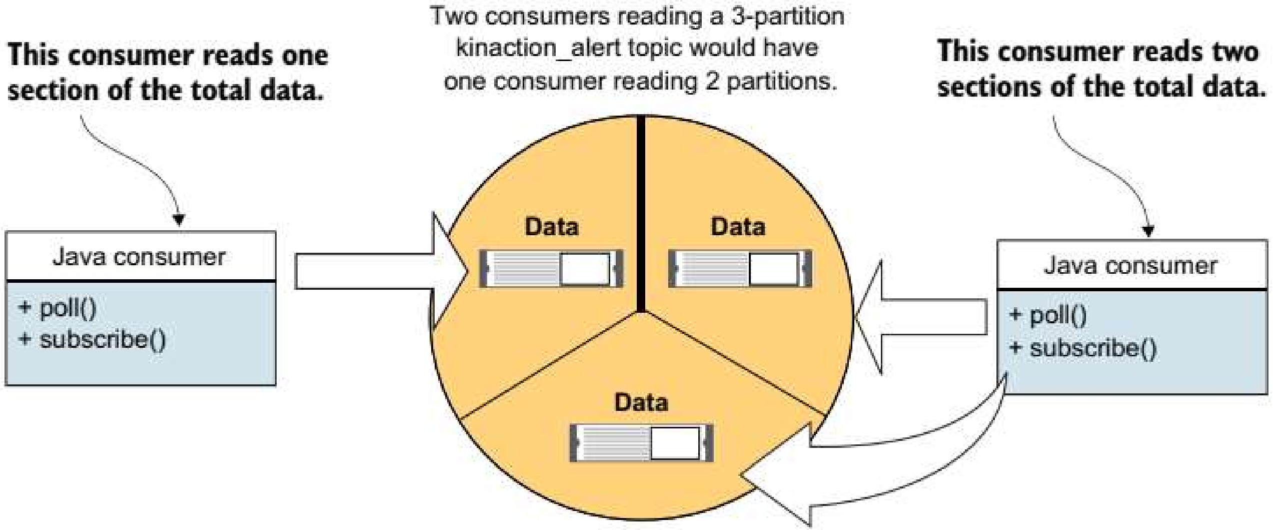
An extra Kafka consumer

Consumers from different groups ignore each other, getting their own copy of the data.



Multiple consumers can read the same data because they have different consumer IDs.

Consumers in separate groups [12]

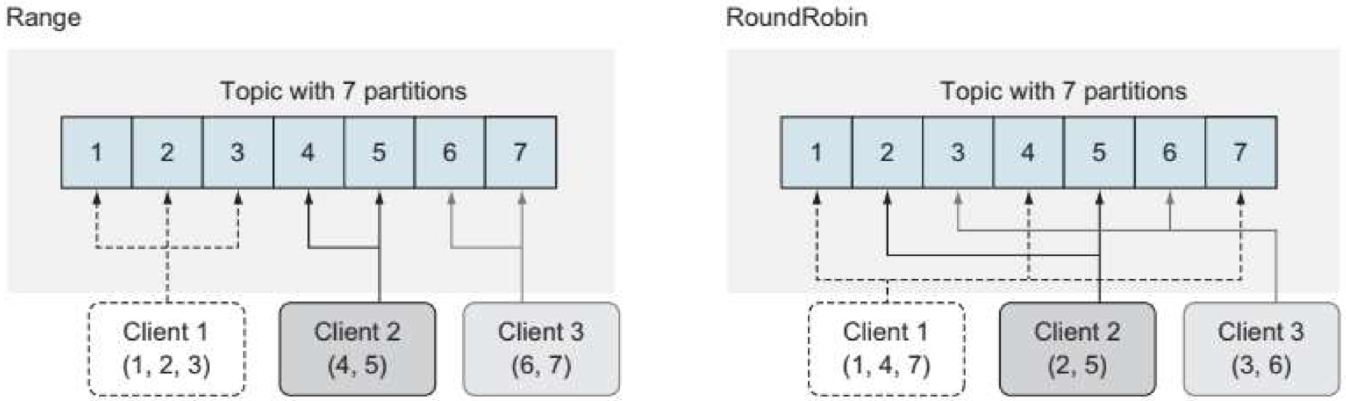


This consumer reads one section of the total data.

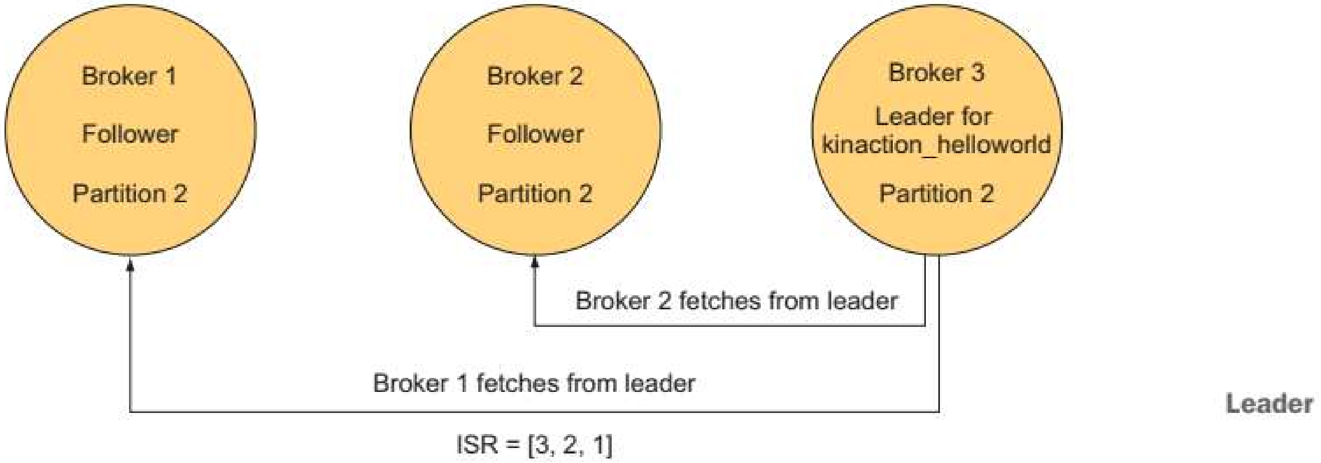
Two consumers reading a 3-partition kinaction_alert topic would have one consumer reading 2 partitions.

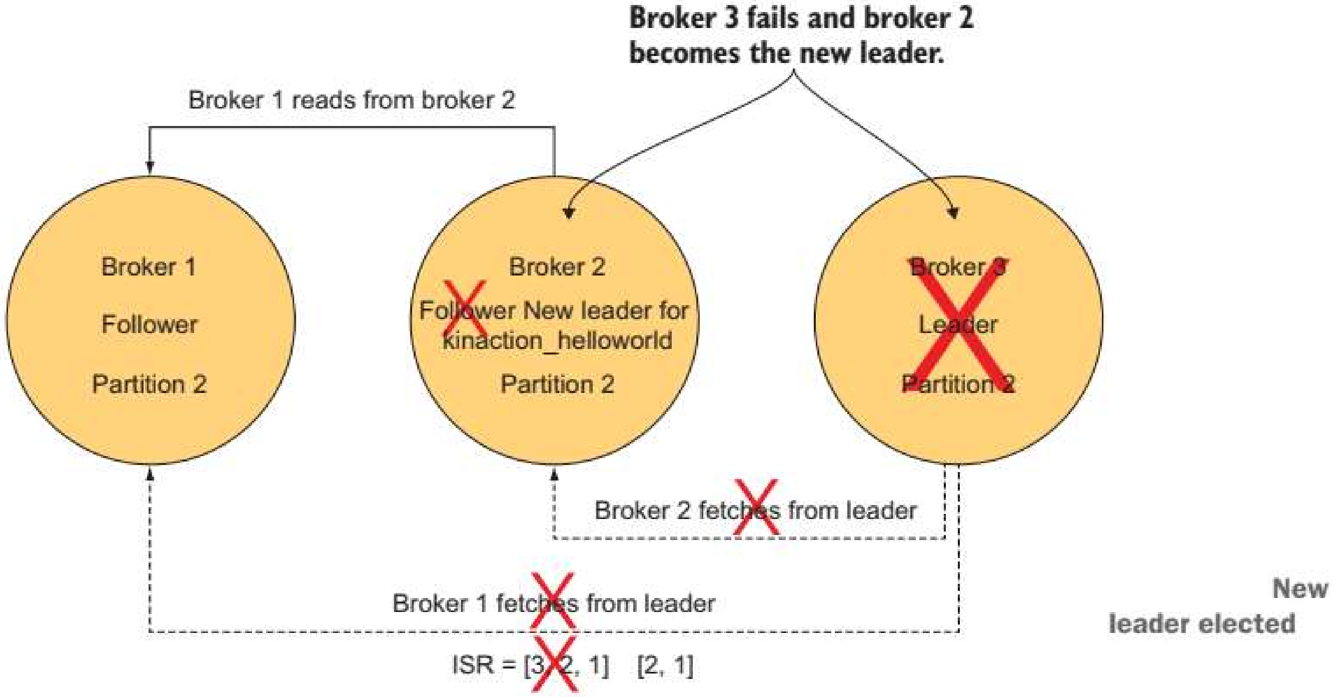
This consumer reads two sections of the total data.

Kafka consumers in a group



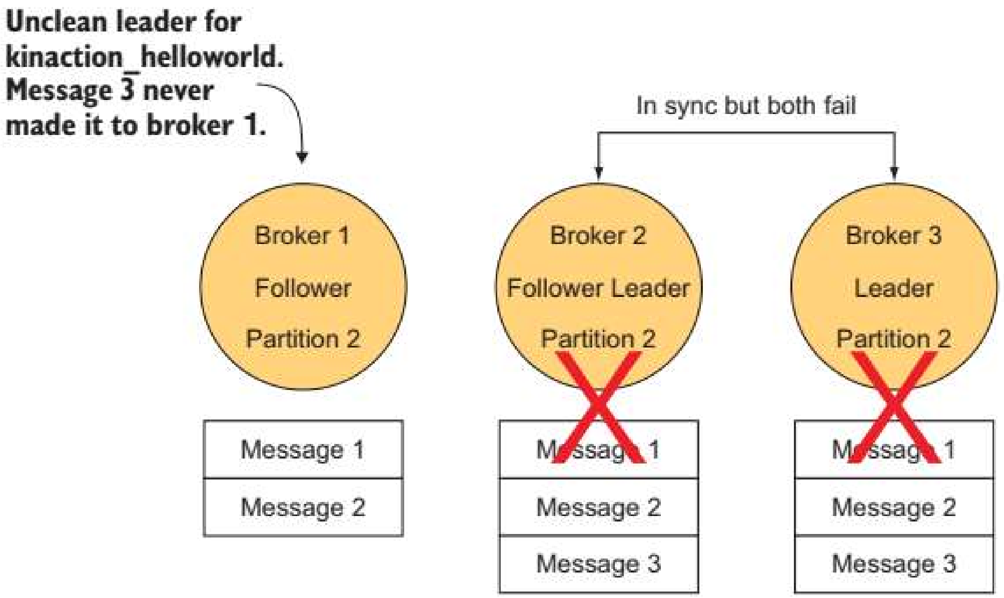
Partition assignments


Broker 1
Follower
Partition 2
Broker 2
Follower
Partition 2
Broker 3
Leader for
kinaction_helloworld
Partition 2
Broker 2 fetches from leader
Broker 1 fetches from leader
ISR = [3, 2, 1]
Leader

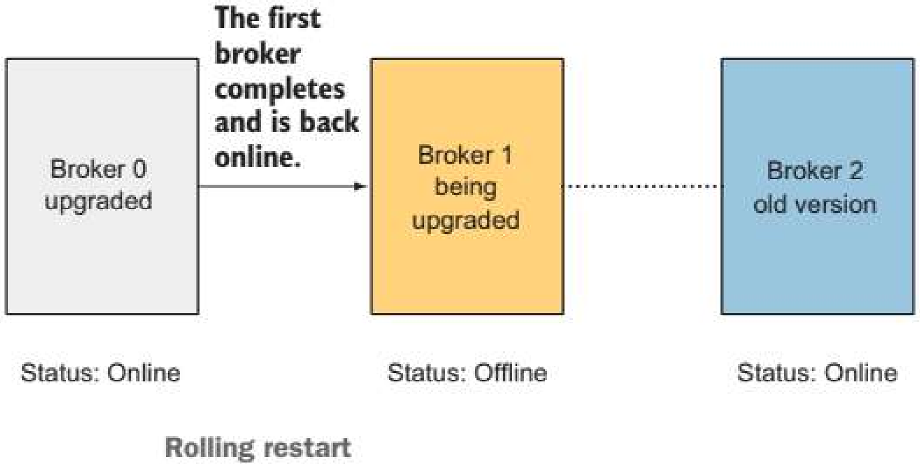

Broker 1 reads from broker 2
Broker 1
Follower
Partition 2
Broker 2
Follower New leader for kinaction_helloworld
Partition 2
Broker 3
Leader
Partition 2
Broker 2 fetches from leader
Broker 1 fetches from leader
ISR = [3, 2, 1]   [2, 1]

Broker 3 fails and broker 2 becomes the new leader.

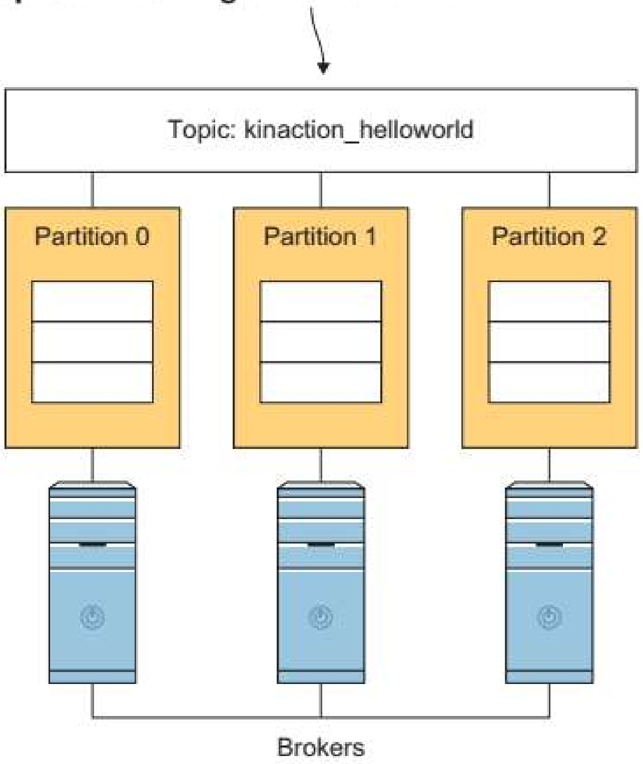
New leader elected



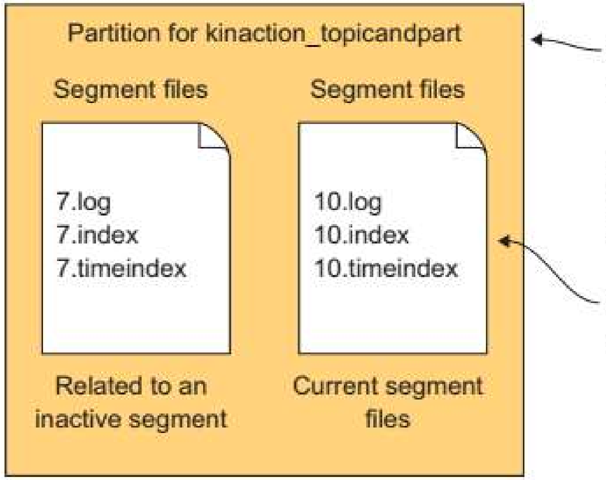
Unclean leader election


The first broker completes and is back online.
Broker 0 upgraded
Broker 1 being upgraded
Broker 2 old version
Status: Online
Status: Offline
Status: Online
Rolling restart

The topic kinaction_helloworld is made up of three partitions that will likely be spread out among different brokers.



Example topic with partitions

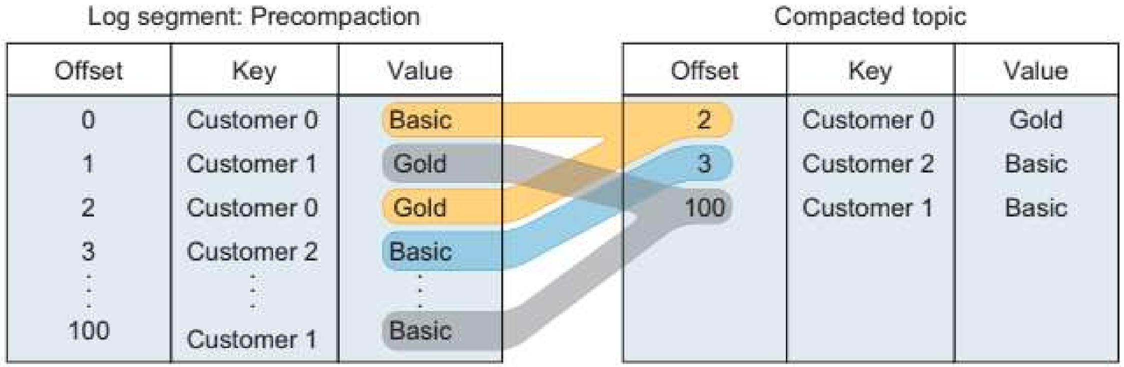


Partition made up of one to many segments

kinaction_topicandpart filename lengths are shortened for this example.

Each segment has multiple similarly named files.

Segments make up a partition.



Log segment: Precompaction

Offset	Key	Value
0	Customer 0	Basic
1	Customer 1	Gold
2	Customer 0	Gold
3	Customer 2	Basic
⋮	⋮	⋮
100	Customer 1	Basic

Compacted topic

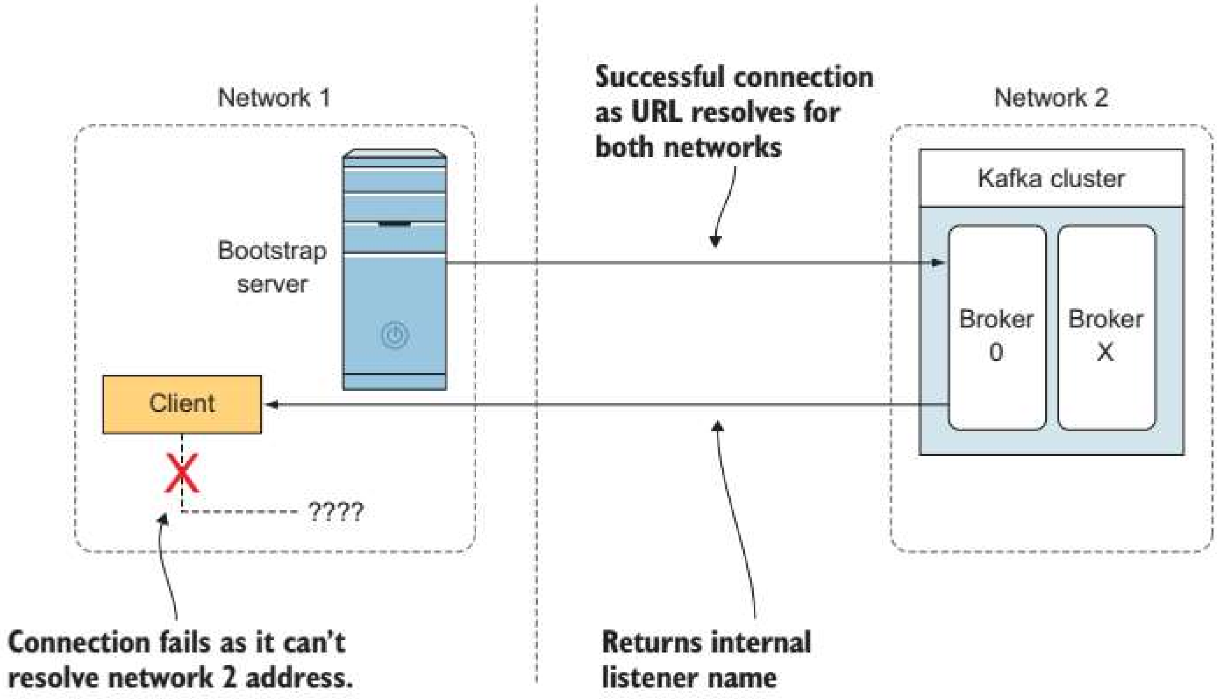
Offset	Key	Value
2	Customer 0	Gold
3	Customer 2	Basic
100	Customer 1	Basic

Compaction in general

Broker retention configuration

Key	Purpose
`log.retention.bytes`	The largest size threshold in bytes for deleting a log.
`log.retention.ms`	The length in milliseconds a log will be maintained before being deleted.
`log.retention.minutes`	Length before deletion in minutes. `log.retention.ms` is used as well if both are set.
`log.retention.hours`	Length before deletion in hours. `log.retention.ms` and `log.retention.minutes` would be used before this value if either of those are set.

Scenario 1: no advertised listeners. Producer client starts and requests metadata from bootstrap server.

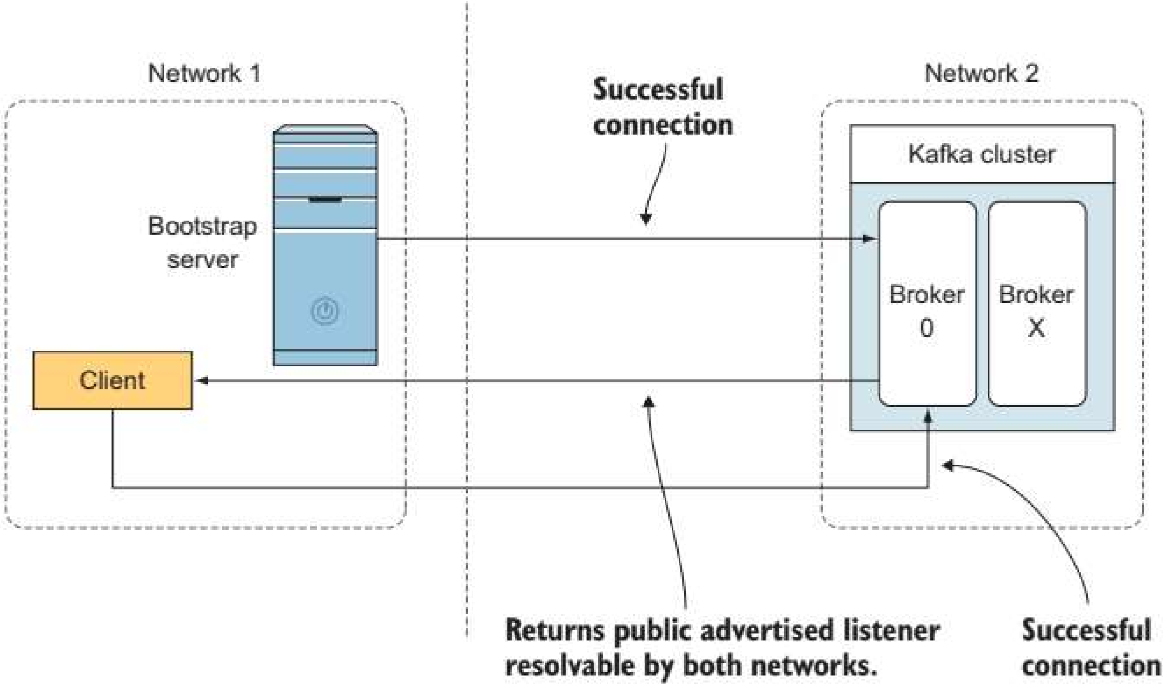


Successful connection as URL resolves for both networks

Connection fails as it can't resolve network 2 address.

Returns internal listener name

Scenario 2: advertised listeners with URL resolved by both networks. Producer client requests metadata.



Thanks