Kafka Basic Notes

By

Naveen Kumar M

https://www.linkedin.com/in/naveen-kumar-m-5a6960119/

Table of Contents

Starting zookeeper

zookeeper-server-start config/zookeeper.properties

Starting kafka server

kafka-server-start config/server.properties

Introduction to Apache Kafka

https://www.gentlydownthe.stream/

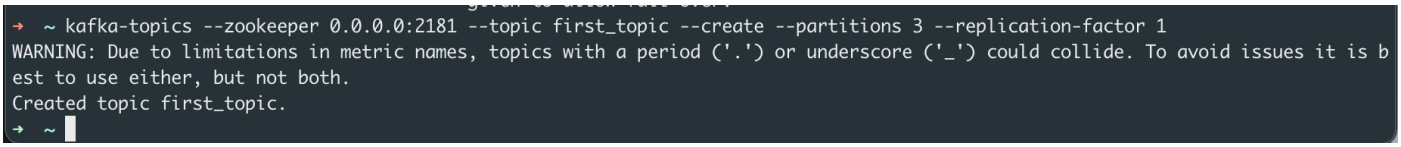
Zero Copy Princinple

https://iamonkar.dev/zero-copy/

Topic

Creating a new topic

kafka-topics --zookeeper 0.0.0.0:2181 --topic first_topic --create --partitions 3 --replication-factor 1



Listing Topics

kafka-topics --zookeeper 0.0.0.0:2181 --list

Describing Topic

kafka-topics --zookeeper 0.0.0.0:2181 --topic first_topic --describe

```
first_topic
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic first_topic --describe
Topic: first_topic      TopicId: ogRgIlPDRmSmdx3gdAAgFA PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: first_topic      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: first_topic      Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: first_topic      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
➜  ~
```

Delete the topic

kafka-topics --zookeeper 0.0.0.0:2181 --topic second_topic –delete

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic second_topic --delete
Topic second_topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Producers

Creating a producer

kafka-console-producer --broker-list 192.168.1.100:9092 --topic first_topic

```
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic first_topic
>First Topic message 1
>Fist topic messgage 2
>Fist topic messgage 3
>Fist topic messgage 4
>^C%
```

Producers with property

kafka-console-producer --broker-list 192.168.1.100:9092 --topic first_topic --producer-property acks=all

```
~/kafka_2.12-2.0.0 kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic --producer-property acks=all
>some message that is acked
>just for fun
>fun learning!
>^C%
```

```
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic first_topic --producer-property acks=a
ll
>Fist topic messgage 5
>Fist topic messgage 6
>Fist topic messgage 7
>^C%
```

Creating new topic with producers

- It’s best practice to create a topic before producers.
- When creating a topic by producers, topics will be created without partition and replica.

kafka-console-producer --broker-list 192.168.1.100:9092 --topic new_topic

```
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic new_topic
>Fist topic messgage 8
[2021-09-01 18:18:50,237] WARN [Producer clientId=console-producer] Error while fetching metadata with corr
elation id 3 : {new_topic=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --list
first_topic
new_topic
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic new_topic --describe
Topic: new_topic        TopicId: k38No0dDSIKGZCJev_1_WA PartitionCount: 1       ReplicationFactor: 1    Con
figs:
        Topic: new_topic        Partition: 0    Leader: 0       Replicas: 0     Isr: 0
➜  ~
```

In the server.properties file, we can configure the default partition count as followes

```
############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=/Users/naveen.kumar1/Softwares/Kafka/kafka_2.13-2.8.0/data/kafka
# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=3

# The number of threads per data directory to be used for log recovery at startup and flushing at shutdown.
# This value is recommended to be increased for installations with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Internal Topic Settings  #############################
# The replication factor for the group metadata internal topics "__consumer_offsets" and "__transaction_state"
# For anything other than development testing, a value greater than 1 is recommended to ensure availability such as 3.
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
```

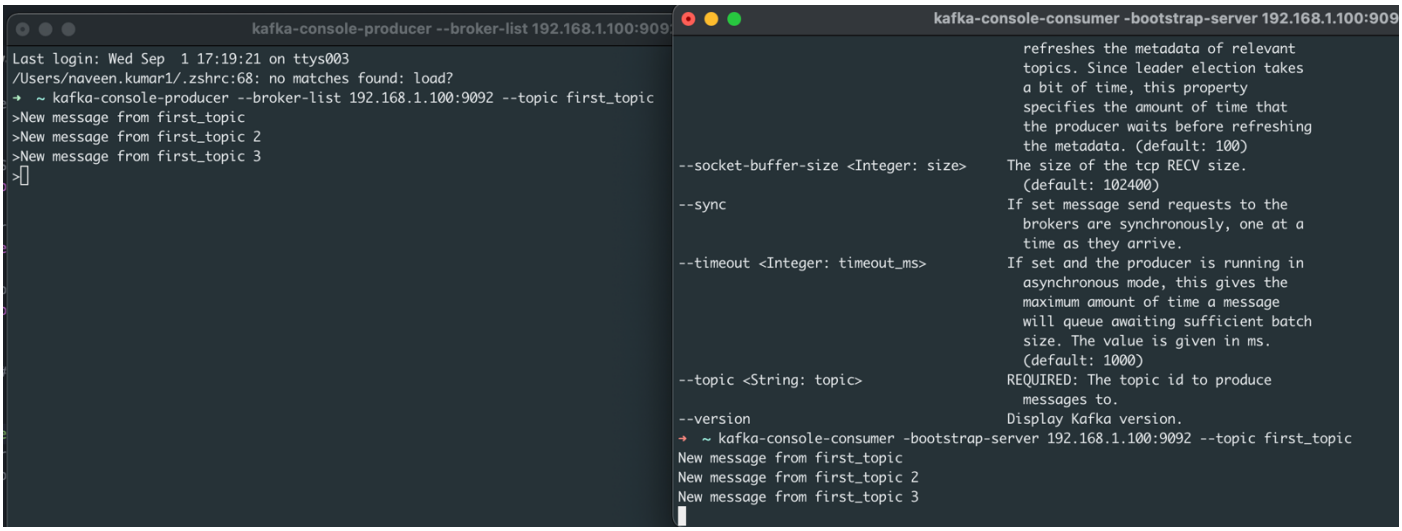
```
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic new_topic2
>new topic 2
[2021-09-01 18:36:57,464] WARN [Producer clientId=console-producer] Error while fetching metadata with corr
elation id 3 : {new_topic2=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2021-09-01 18:36:57,577] WARN [Producer clientId=console-producer] Error while fetching metadata with corr
elation id 4 : {new_topic2=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
>^C%
➜  ~
```

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic new_topic2 --describe
Topic: new_topic2       TopicId: GsQJkfCDRZGXloB3gK-X2A PartitionCount: 3       ReplicationFactor: 1    Con
figs:
        Topic: new_topic2       Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: new_topic2       Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: new_topic2       Partition: 2    Leader: 0       Replicas: 0     Isr: 0
➜  ~
```

Consumer

Listening to message for given topic from producer

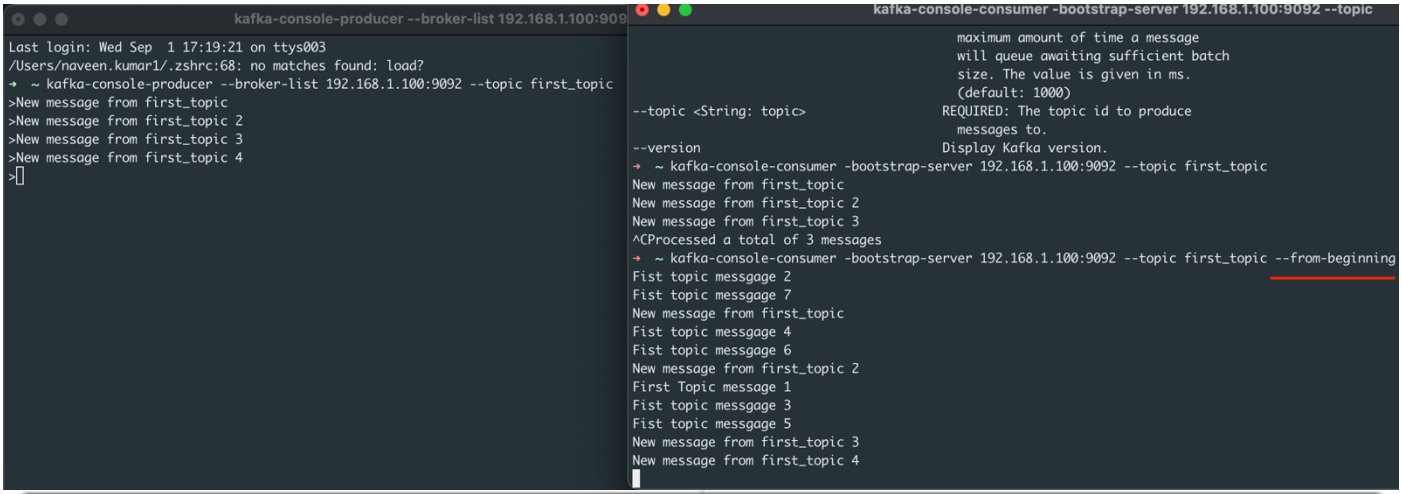
kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic first_topic



Listening to message for given topic from producer and retrieve all the messages from beginning

kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic first_topic --from-beginning

As we can see below, consumer brought all the messages passed from the beginning. Since we have three partitions, order within partition is maintained not across the partition.

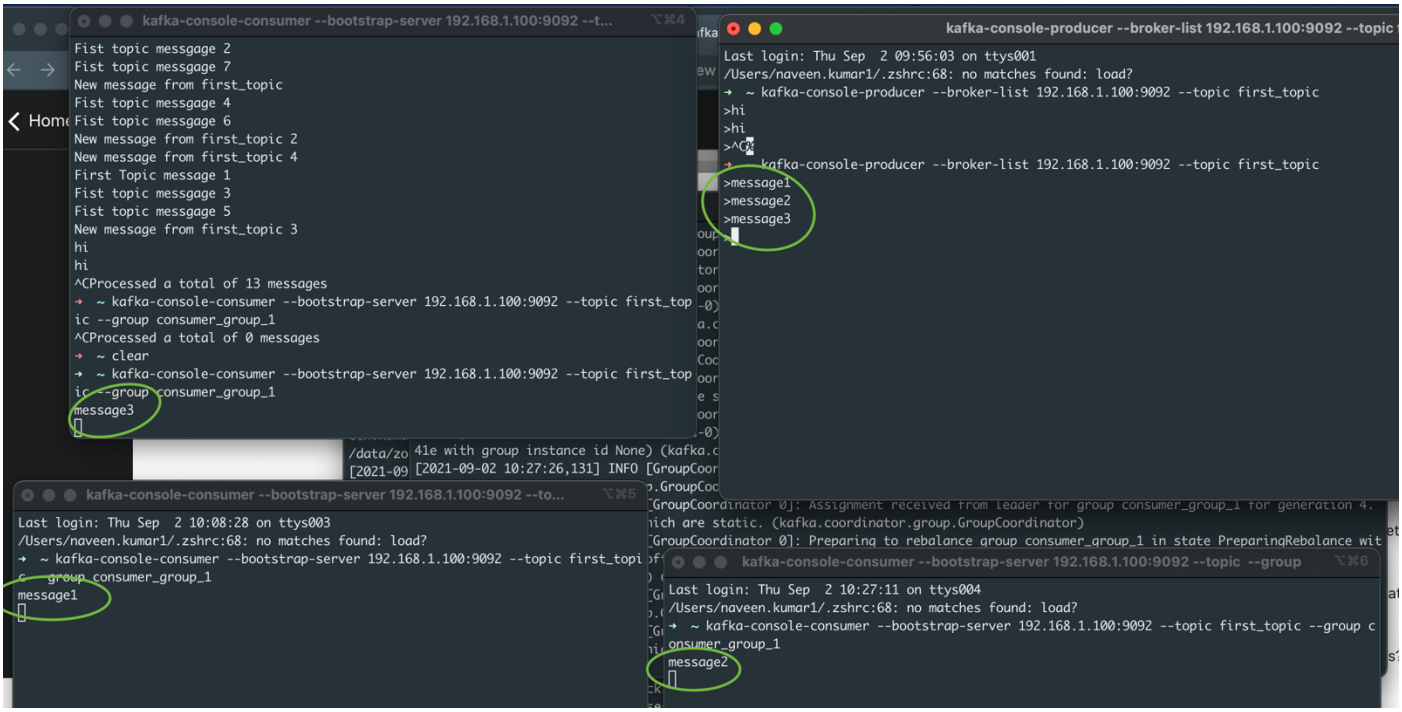


Important Note: as you can see, the order of the messages in this consumer is not "total", the order is per partition. Because "first_topic" was created with 3 partitions, we saw in the theory lectures that the order is only guaranteed at the partition level. If you try with a topic with 1 partition, you will see total ordering

Consumer group

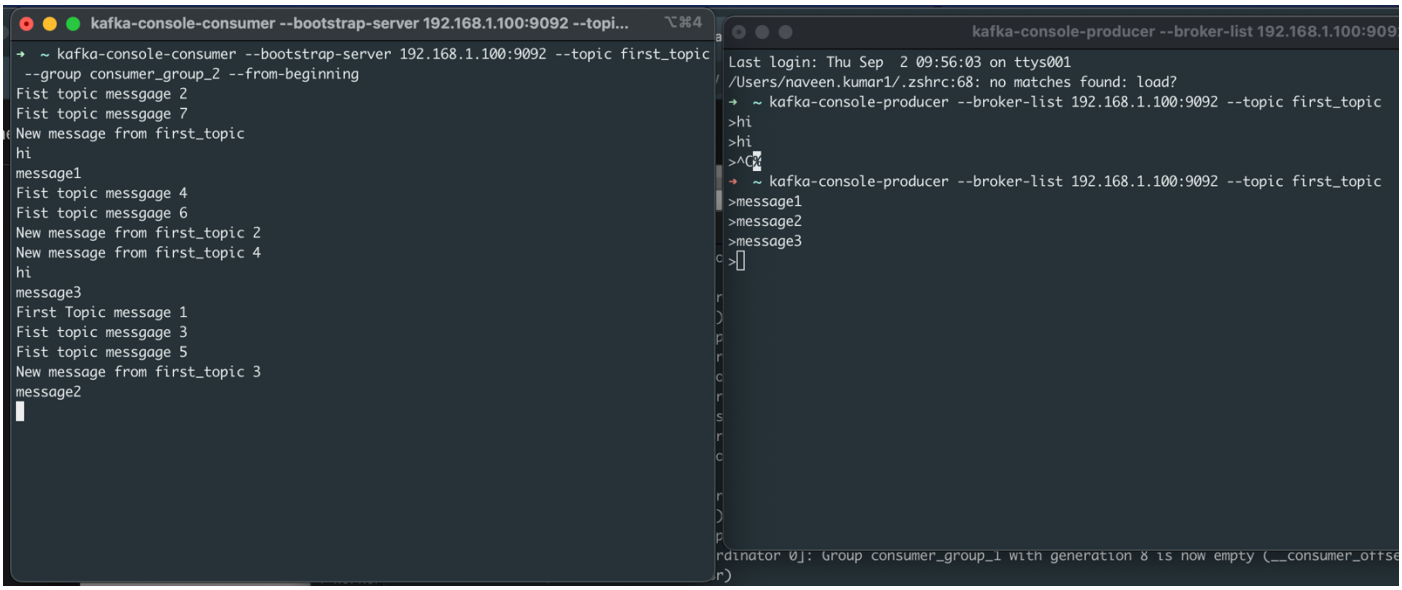
kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic first_topic --group consumer_group_1

In below example, we created a three consumer under the group called “consumer_group_1”. Since topic “first_topic” has a three partition, each consumer listening to different partition respectively.

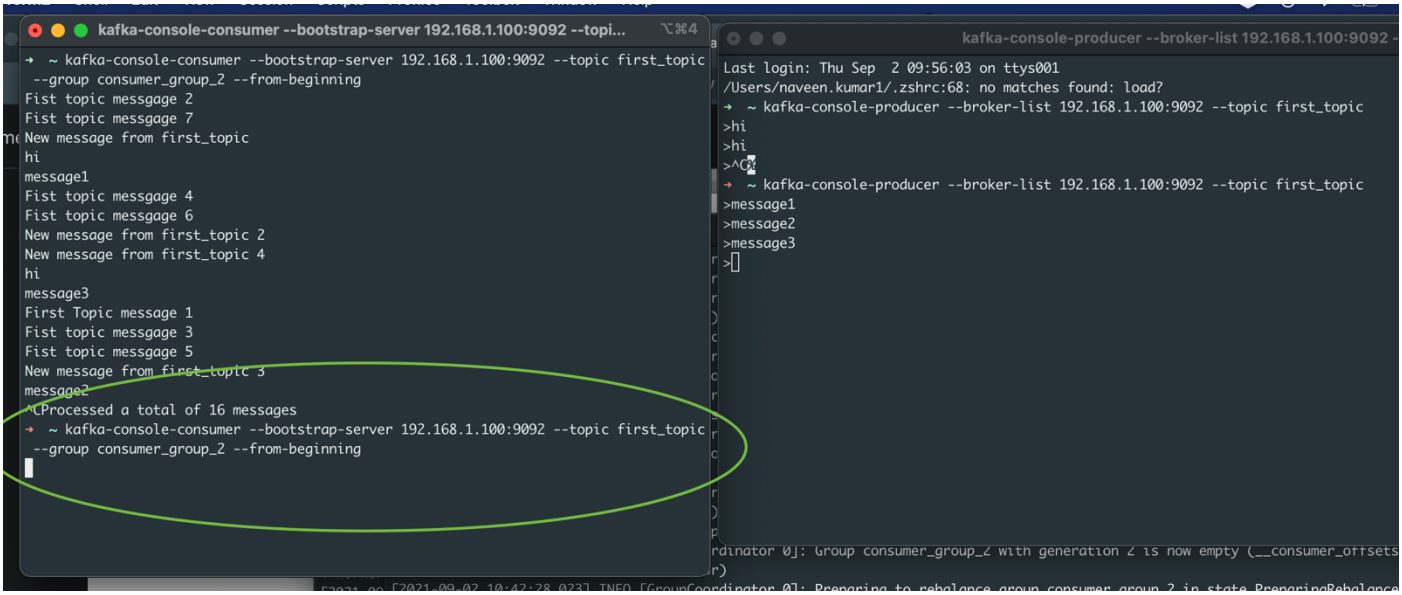


Consumer offset

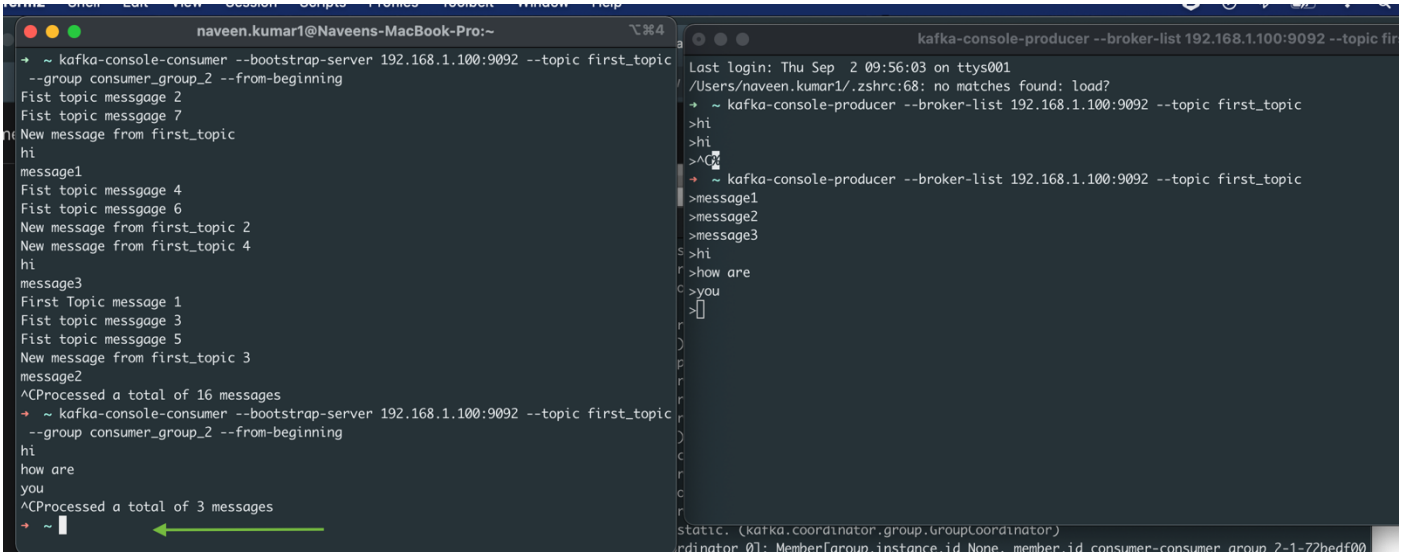
As per the below screen shot, when try to use the “--from-beginning”, we will be getting all the messages from a topic.

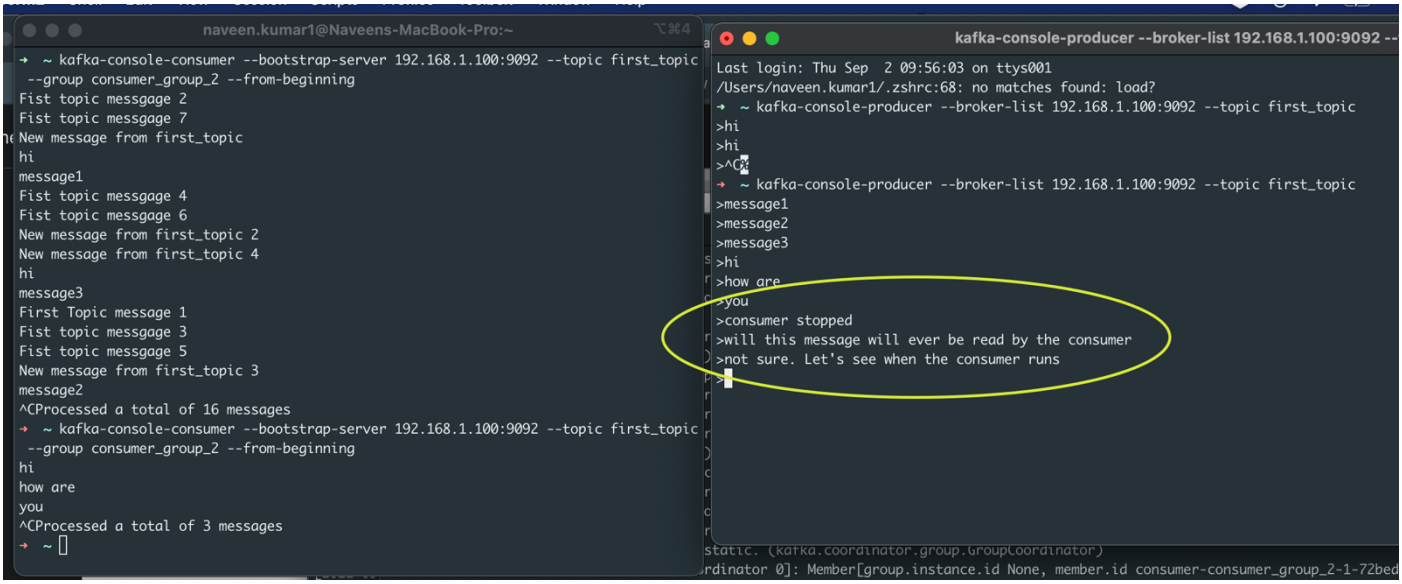


We try to run the command again; we won’t be getting all the messages as all the messages were read by the previous consumer. This is due to consumer offset.

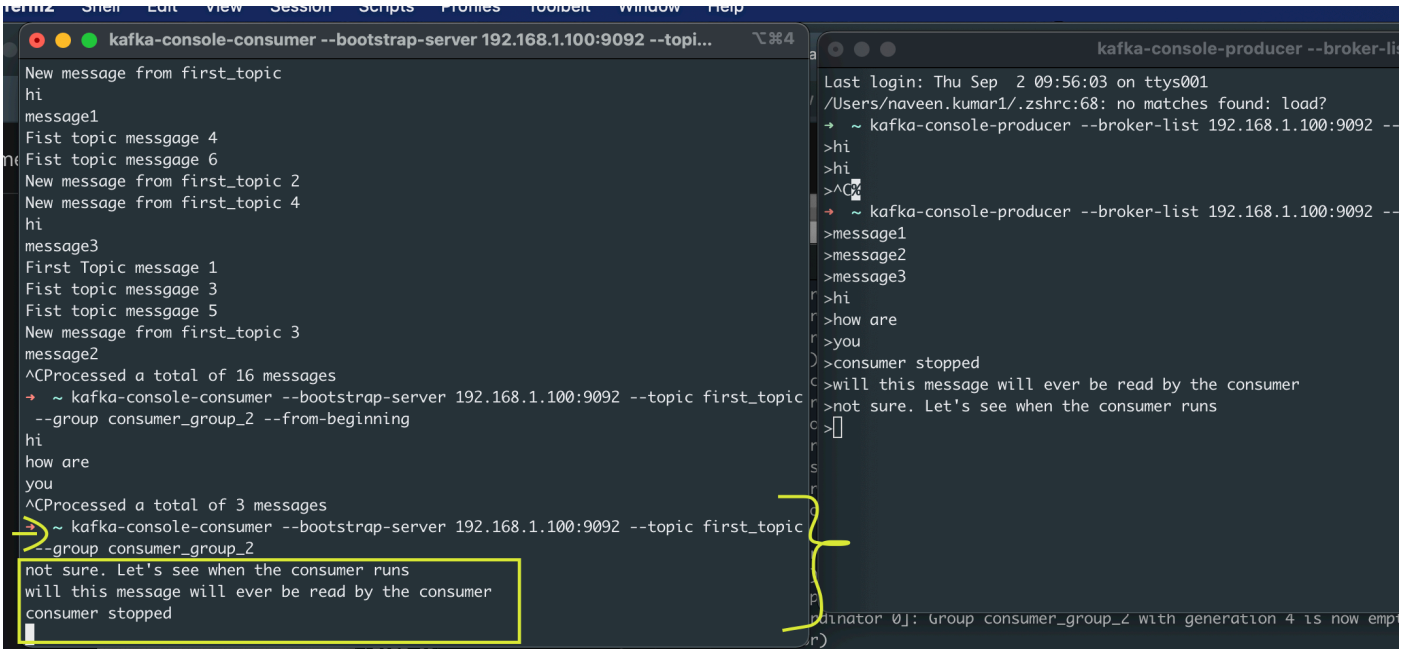


Let’s stop the consumer and produce some message using producer.





Now start the consumer and check whether produced messages are read by the consumer or not.



As per the abo screenshot, messages are read by the consumer.

Consumer group commands

Note: As of Kafka >= 2.1, the "console-consumer-" groups will not appear in this command anymore.

Listing all the consumer groups

kafka-consumer-groups --bootstrap-server localhost:9092 --list

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --list
consumer_group_2
consumer_group_1
➜  ~
```

Describing consumer group

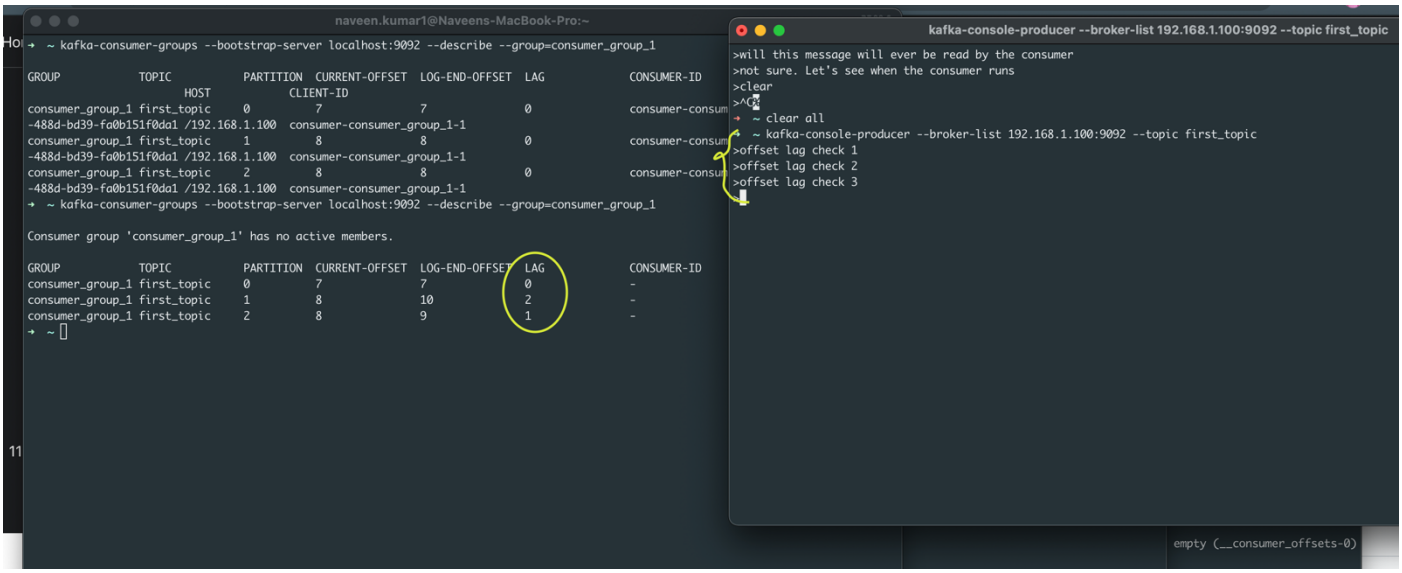
kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_2

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_2

Consumer group 'consumer_group_2' has no active members.

GROUP            TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
consumer_group_2 first_topic     0          7               7               0               -               -               -
consumer_group_2 first_topic     1          8               8               0               -               -               -
consumer_group_2 first_topic     2          7               7               0               -               -               -
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_1

Consumer group 'consumer_group_1' has no active members.

GROUP            TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
consumer_group_1 first_topic     0          5               7               2               -               -               -
consumer_group_1 first_topic     1          6               8               2               -               -               -
consumer_group_1 first_topic     2          5               7               2               -               -               -
➜  ~
```

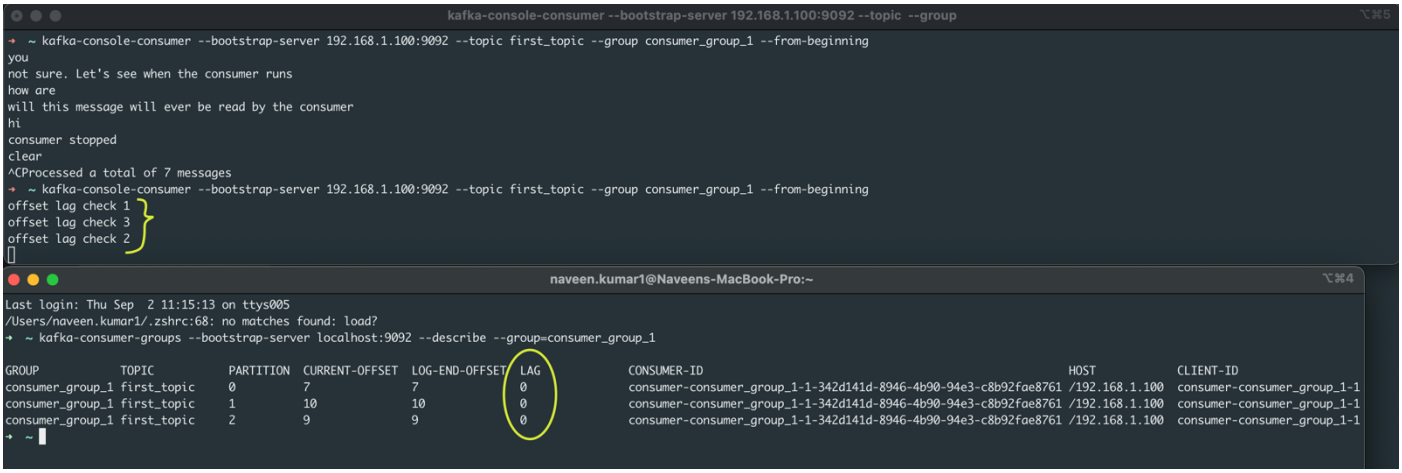
As per the above screenshot, “consumer_group_1” has an offset lag. It means that some messages are not read by the consumer.

Now let’s make the offset lag 0 by running the consumer.

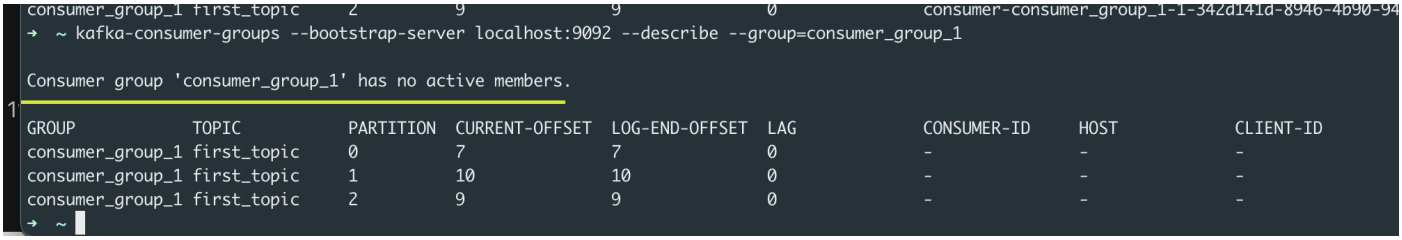
For an experiment, we turned off the consumer and produced and some messages and checked the offset lag count of the consumer. The screenshot of the same is given below.



Let’s check the lag count after running the consumer.

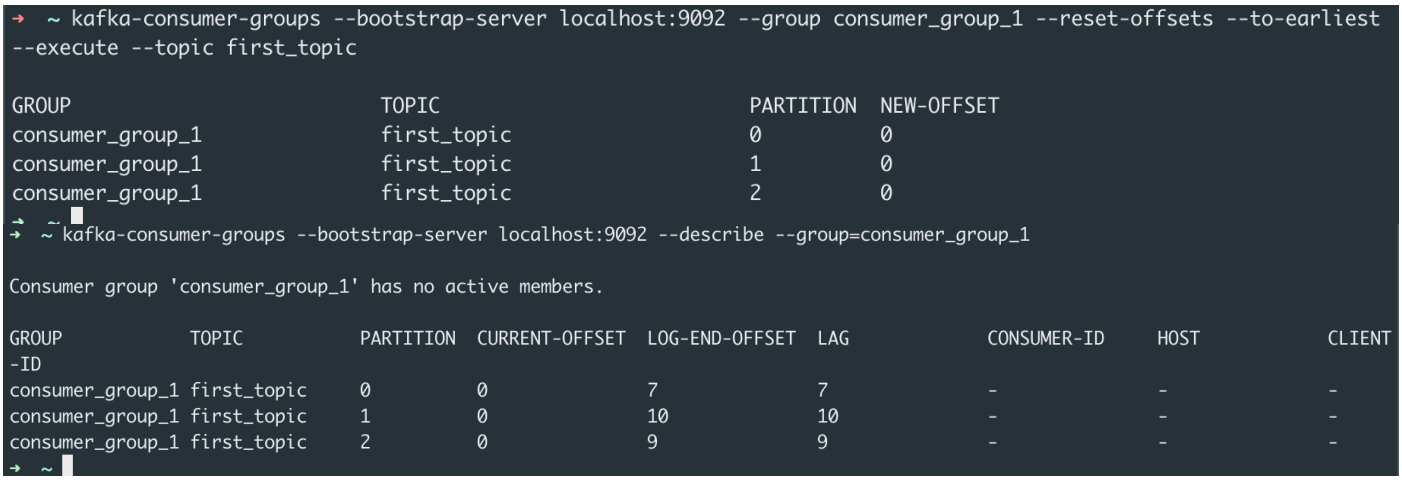


We will be getting this below message when no consumer is running.



Resetting offsets – to earliest

kafka-consumer-groups --bootstrap-server localhost:9092 --group consumer_group_1 --reset-offsets --to-earliest --execute --topic first_topic



Reading the data using consumer

```
➜  ~ kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic first_topic --group consumer_group_1
Fist topic messgage 2
Fist topic messgage 7
New message from first_topic
hi
message1
you
not sure. Let's see when the consumer runs
Fist topic messgage 4
Fist topic messgage 6
New message from first_topic 2
New message from first_topic 4
hi
message3
how are
will this message will ever be read by the consumer
```

Checking the offset lag

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_1

GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
                     HOST            CLIENT-ID
consumer_group_1 first_topic     0          7               7               0               consumer-consumer_group_1-1-e4f30cf8-9
ce9-4a18-b700-0fd11646b94f /192.168.1.100  consumer-consumer_group_1-1
consumer_group_1 first_topic     1          10              10              0               consumer-consumer_group_1-1-e4f30cf8-9
ce9-4a18-b700-0fd11646b94f /192.168.1.100  consumer-consumer_group_1-1
consumer_group_1 first_topic     2          9               9               0               consumer-consumer_group_1-1-e4f30cf8-9
ce9-4a18-b700-0fd11646b94f /192.168.1.100  consumer-consumer_group_1-1
➜  ~
```

Resetting offsets – shift by

kafka-consumer-groups --bootstrap-server localhost:9092 --group consumer_group_1 --reset-offsets --shift-by 2 --execute --topic first_topic

Shift by positive number

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --group consumer_group_1 --reset-offsets --shift-by 2 -
-execute --topic first_topic
[2021-09-02 13:04:32,926] WARN New offset (9) is higher than latest offset for topic partition first_topic-0. Value
will be set to 7 (kafka.admin.ConsumerGroupCommand$)
[2021-09-02 13:04:32,926] WARN New offset (12) is higher than latest offset for topic partition first_topic-1. Value
 will be set to 10 (kafka.admin.ConsumerGroupCommand$)
[2021-09-02 13:04:32,926] WARN New offset (11) is higher than latest offset for topic partition first_topic-2. Value
 will be set to 9 (kafka.admin.ConsumerGroupCommand$)

GROUP                          TOPIC                          PARTITION  NEW-OFFSET
consumer_group_1               first_topic                    0          7
consumer_group_1               first_topic                    1          10
consumer_group_1               first_topic                    2          9
```

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_1

Consumer group 'consumer_group_1' has no active members.

GROUP           TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST           CLIENT
-ID
consumer_group_1 first_topic    0          7               7               0               -               -              -
consumer_group_1 first_topic    1          10              10              0               -               -              -
consumer_group_1 first_topic    2          9               9               0               -               -              -
➜  ~
```

Shift by negative number

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --group consumer_group_1 --reset-offsets --shift-by -2
--execute --topic first_topic

GROUP                          TOPIC                          PARTITION  NEW-OFFSET
consumer_group_1               first_topic                    0          5
consumer_group_1               first_topic                    1          8
consumer_group_1               first_topic                    2          7
➜  ~
```

```
➜  ~ kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group=consumer_group_1

Consumer group 'consumer_group_1' has no active members.

GROUP           TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST           CLIENT
-ID
consumer_group_1 first_topic    0          5               7               2               -               -              -
consumer_group_1 first_topic    1          8               10              2               -               -              -
consumer_group_1 first_topic    2          7               9               2               -               -              -
➜  ~
```

Consumer and Producers with Keys

kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic --property parse.key=true --property key.separator=,

kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --from-beginning --property print.key=true --property key.separator=,

```
➜  ~ kafka-console-producer --broker-list 127.0.0.1:9092 --topic first_topic --property parse.key=true --property key.separator=,
a,>b
>a,b
>b,c
>c,d
>
```

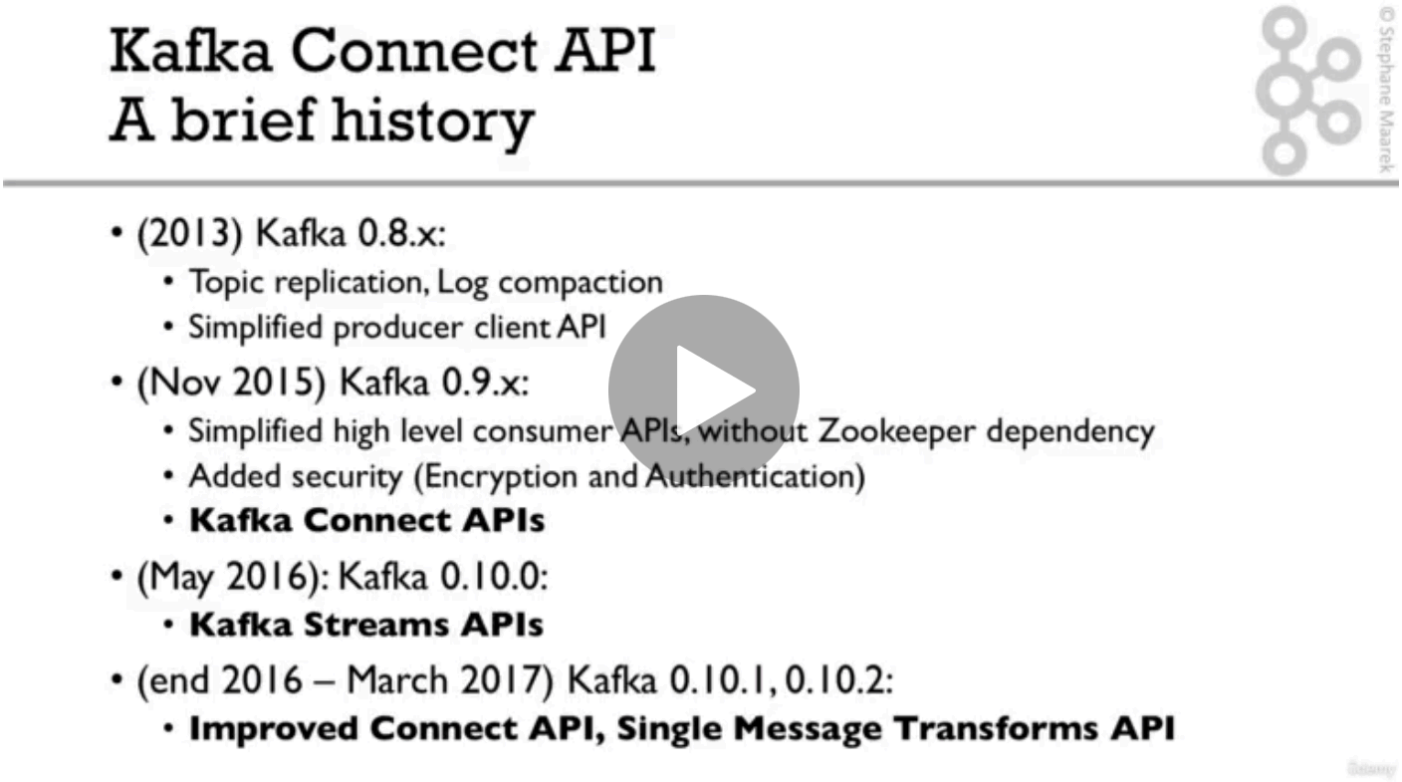
```
➜  ~ kafka-console-consumer --bootstrap-server 127.0.0.1:9092 --topic first_topic --from-beginning --property print.key=true --pro
perty key.separator=,
null,Fist topic messgage 2
null,Fist topic messgage 7
null,New message from first_topic
null,hi
null,message1
null,you
null,not sure. Let's see when the consumer runs
null,Fist topic messgage 4
null,Fist topic messgage 6
null,New message from first_topic 2
```

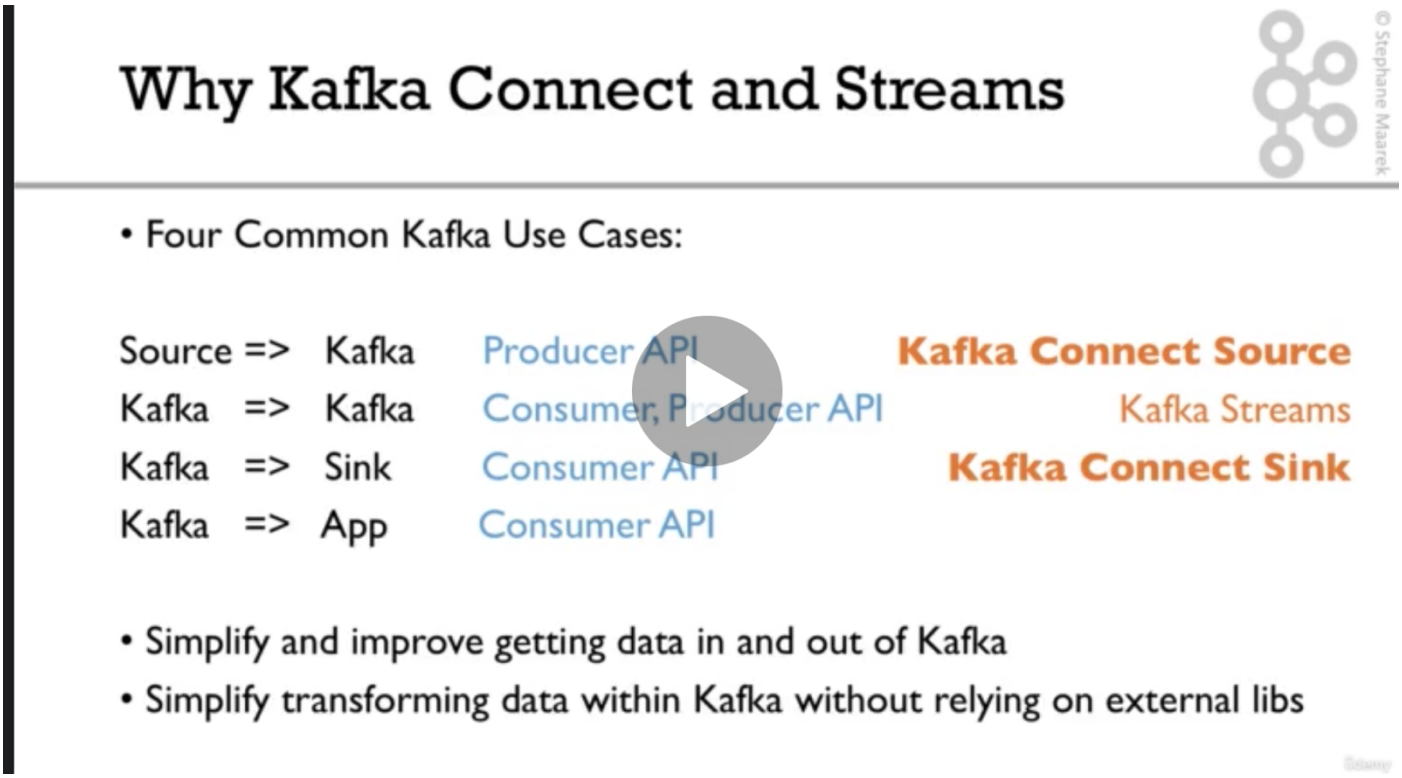
Producer Configuration
https://kafka.apache.org/documentation/#producerconfigs

Consumer Configuration
https://kafka.apache.org/documentation/#consumerconfigs

Kafka Connect

API History



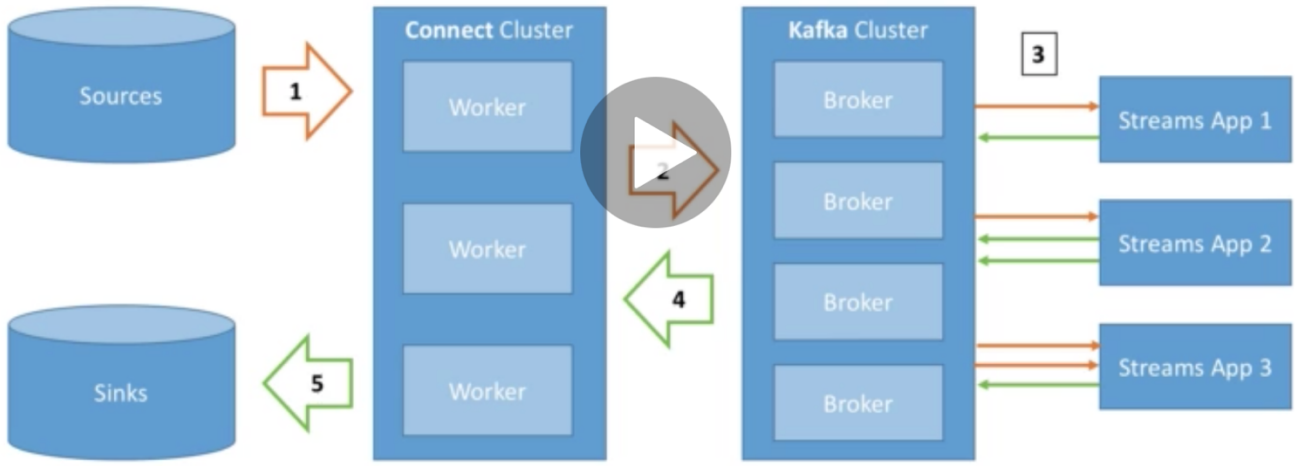


Why Kafka Connect

Why Kafka Connect

- Programmers always want to import data from the same sources:
  Databases, JDBC, Couchbase, GoldenGate, SAP HANA, Blockchain, Cassandra, DynamoDB, FTP, IOT, MongoDB, MQTT, RethinkDB, Salesforce, Solr, SQS, Twitter, etc…
- Programmers always want to store data in the same sinks:
  - S3, ElasticSearch, HDFS, JDBC, SAP HANA, DocumentDB, Cassandra, DynamoDB, HBase, MongoDB, Redis, Solr, Splunk, Twitter
- It is tough to achieve Fault Tolerance, Idempotence, Distribution, Ordering
- Other programmers may already have done a very good job!

Kafka Connect and Streams Architecture Design



- Here **Connect cluster** holds the Kafka Producer API on 1st step and Consumer API on 5th step.

Kafka Connect – High level

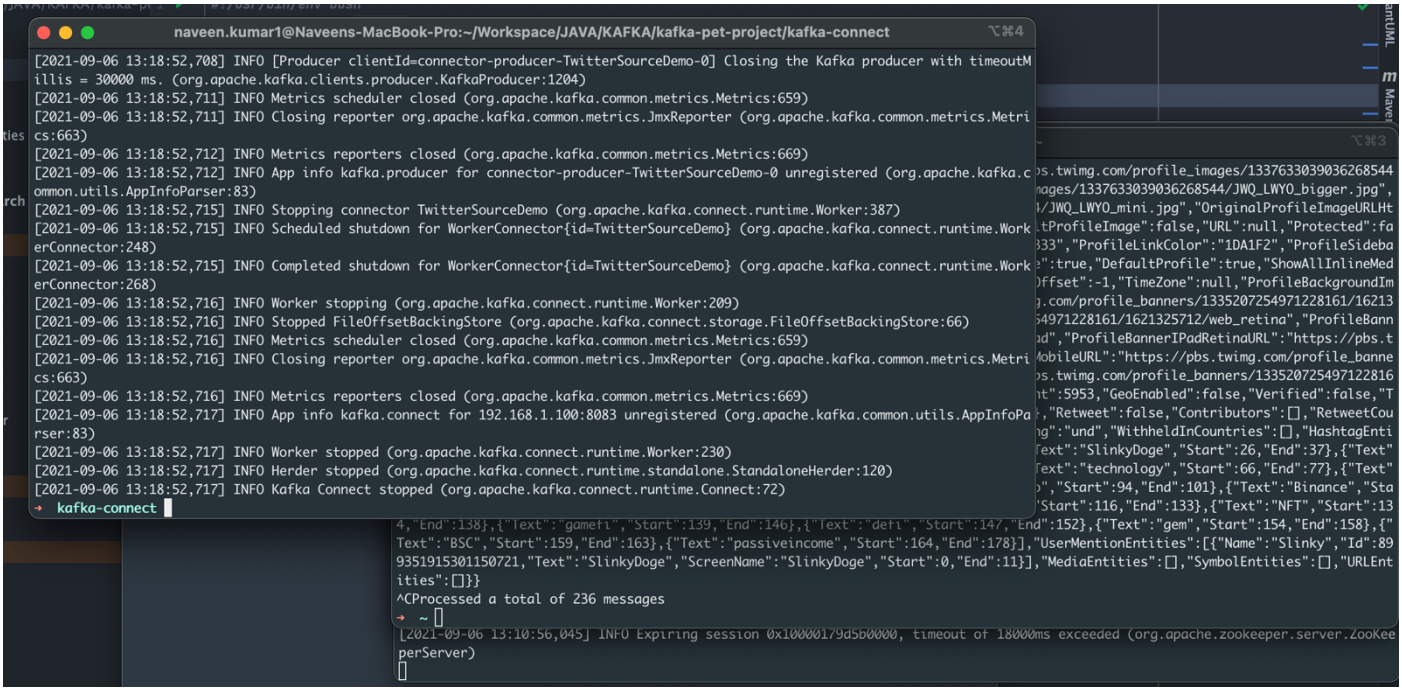
- Source Connectors to get data from Common Data Sources
- Sink Connectors to publish that data in Common Data Stores
- Make it easy for non-experienced dev to quickly get their data reliably into Kafka
- Part of your ETL pipeline
- Scaling made easy from small pipelines to company-wide pipelines
- Re-usable code!

- Easy for ETL.
- Reusable code.
- Non experienced dev can do it easily.
- Readymade code for connecting transferring data between source and target system in seamless way.

Kafka Connector Example

connect-standalone connect-standalone.properties twitter.properties

kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic twitter_status_connect --from-beginning

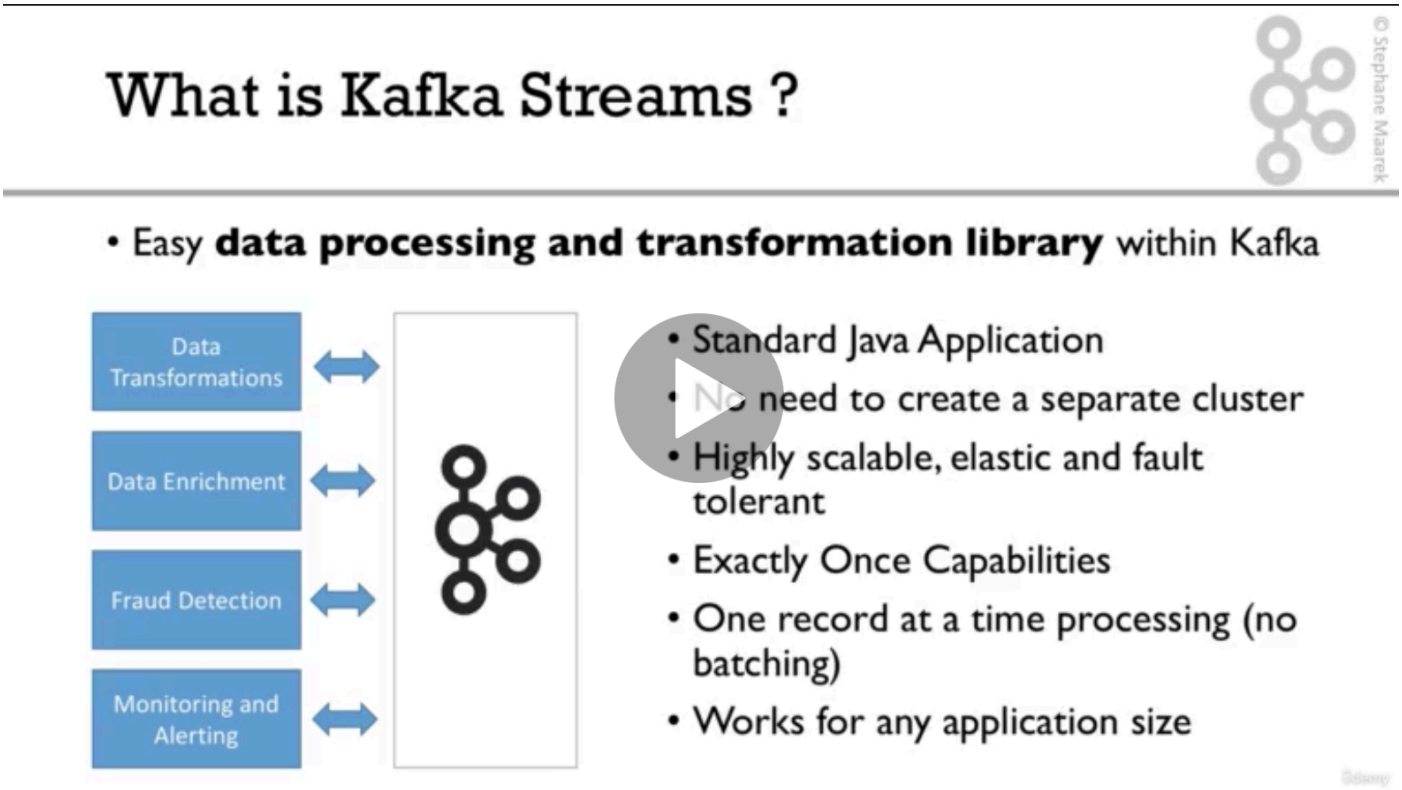


Kafka Streams

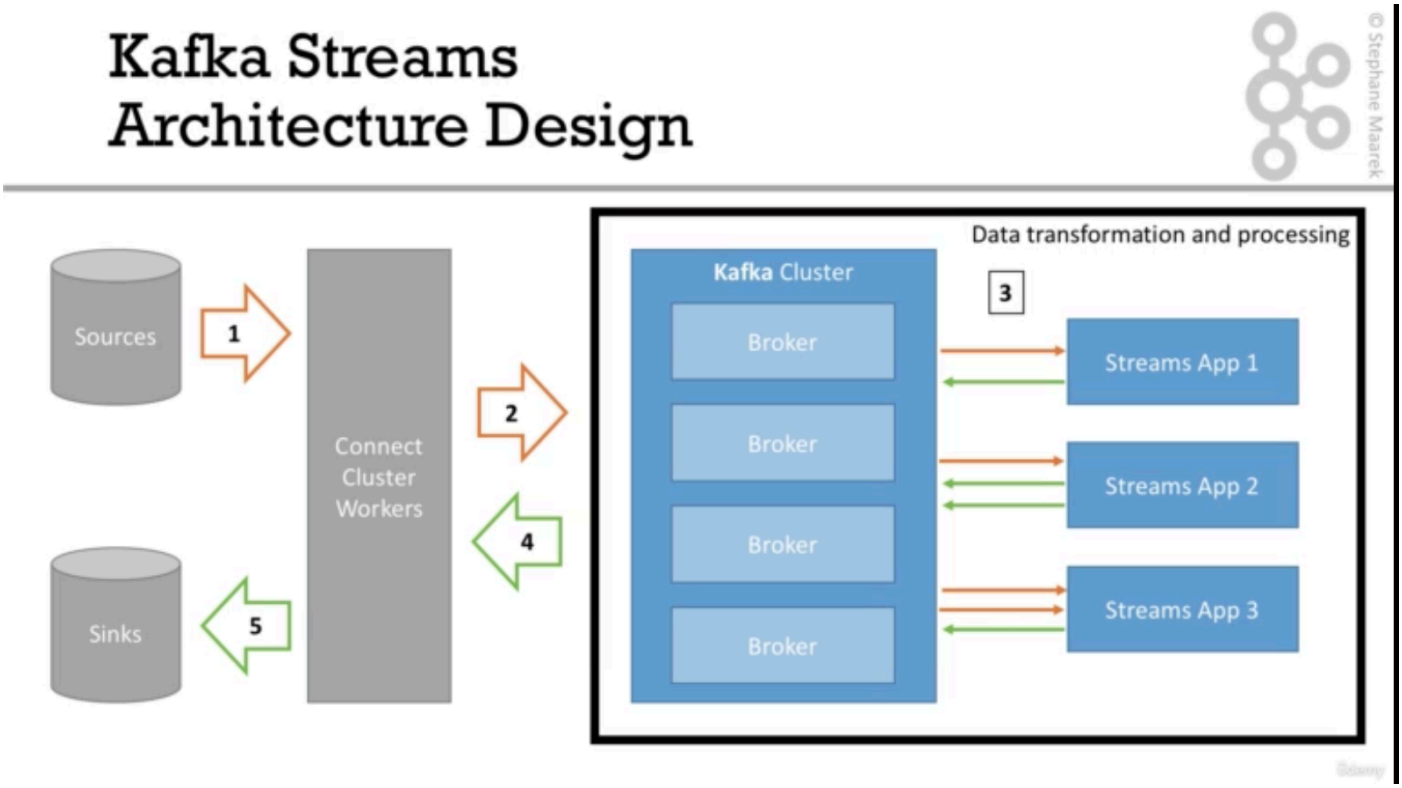
Why Kafka Stream?

- You want to do the following from the twitter_tweets topic:
  - Filter only tweets that have over 10 likes or replies
  - Count the number of tweets received for each hashtag every 1 minute
- Or combine the two to get trending topics and hashtags in real time!
- With Kafka Producer and Consumer, you can achieve that but it's very low level and not developer friendly

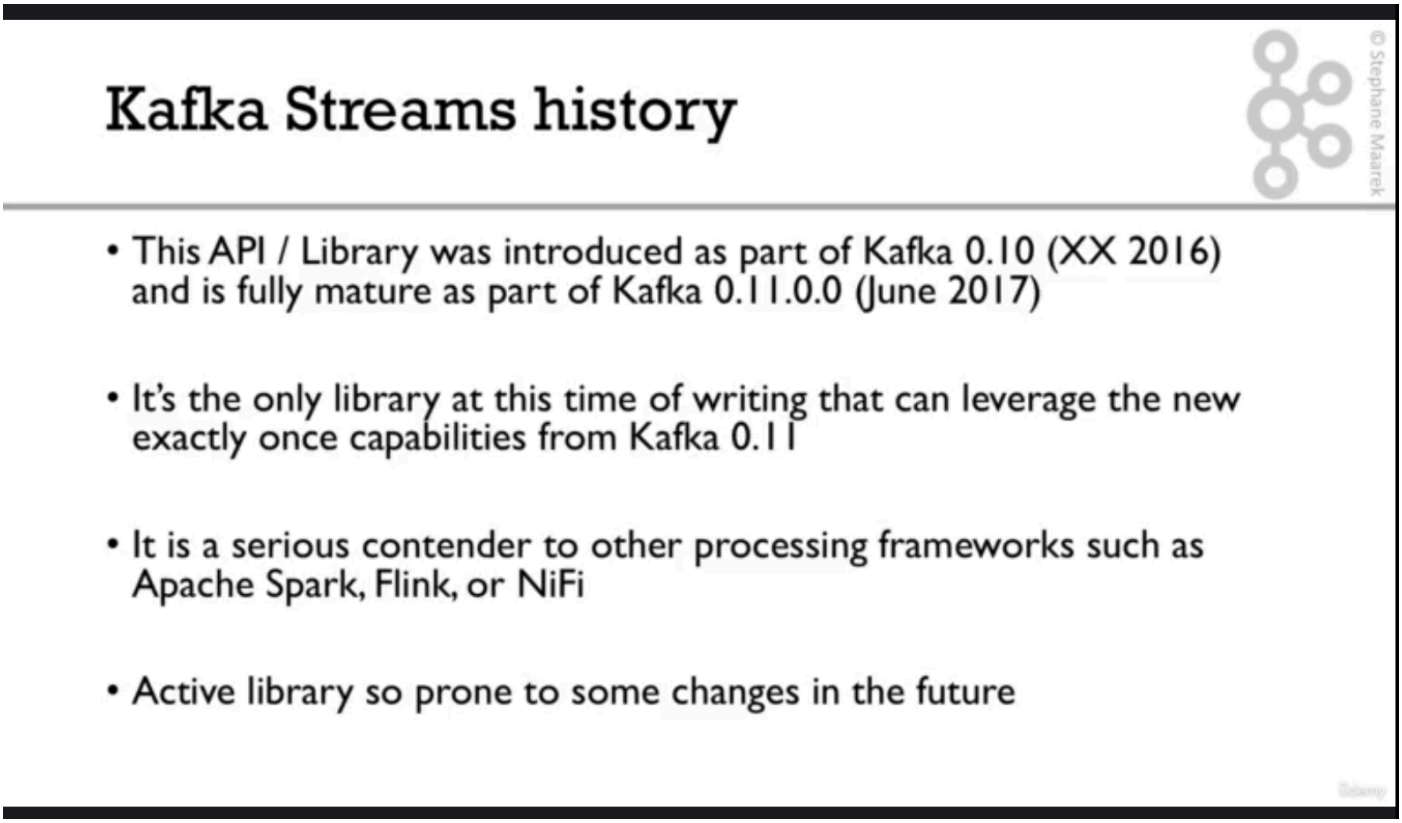
What is Kafka Stream?



Kafka Stream Architecture

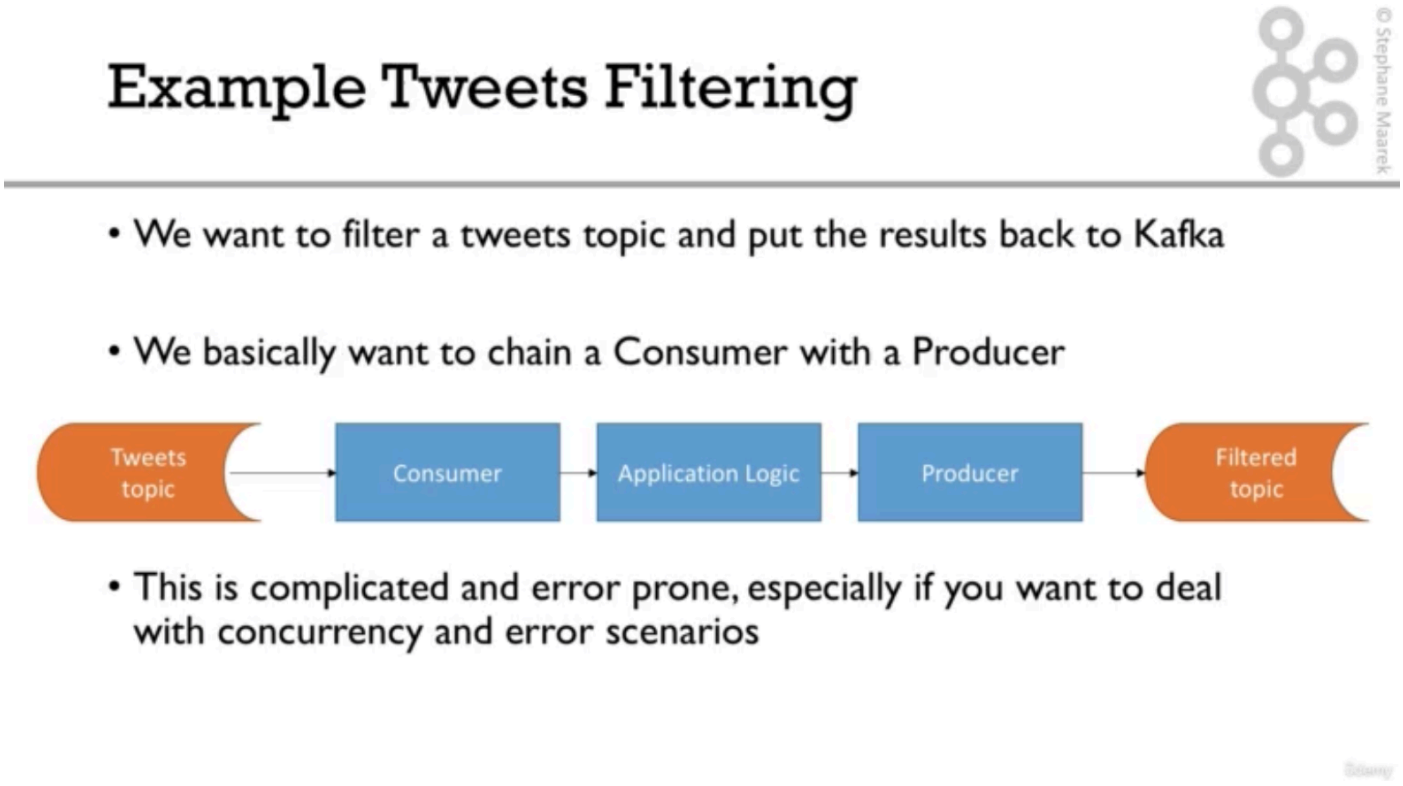


Kafka Stream History



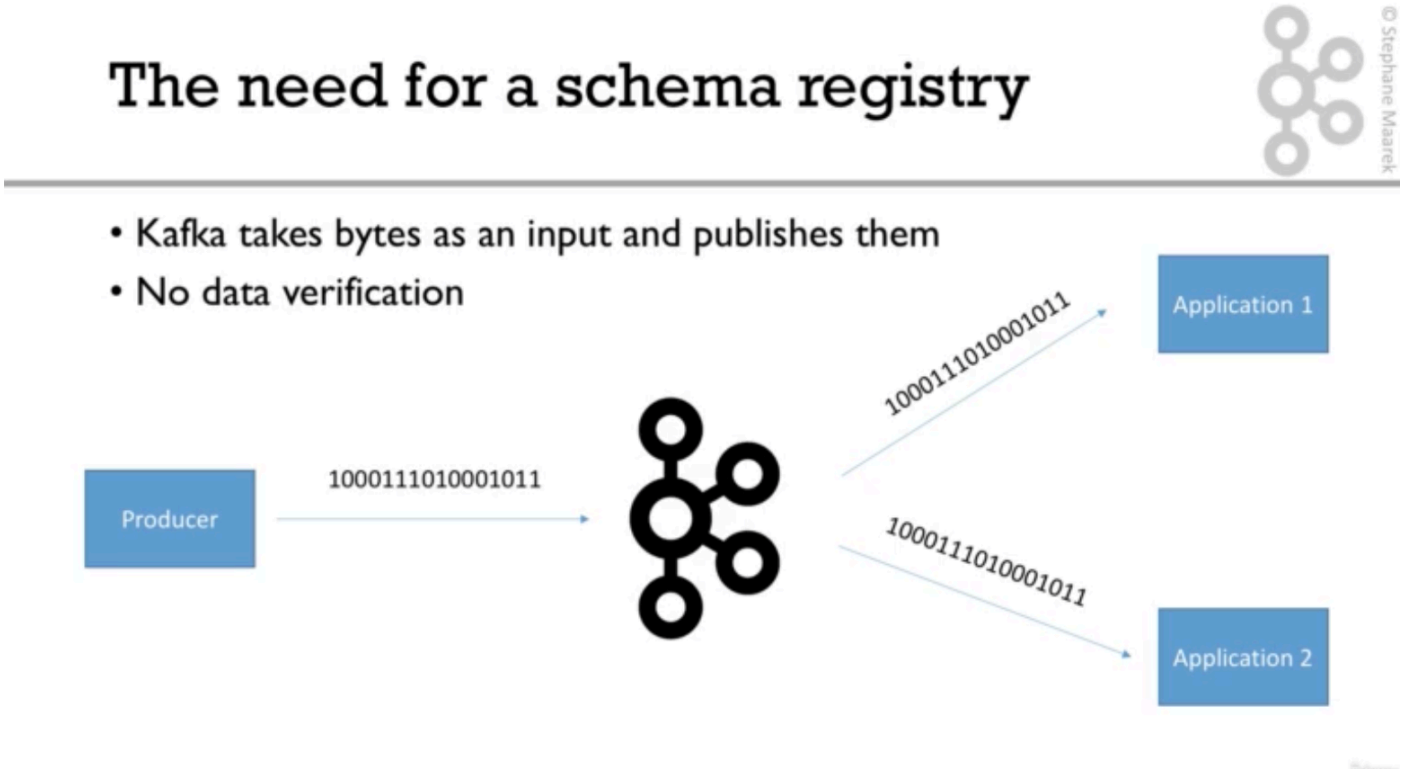
If no Kafka Stream

Example Tweets Filtering



Schema registry

The Need for schema registry



The need for a schema registry

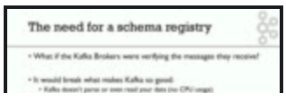
- What if the producer sends bad data?
- What if a field gets renamed?
- What if the data format changes from one day to another?

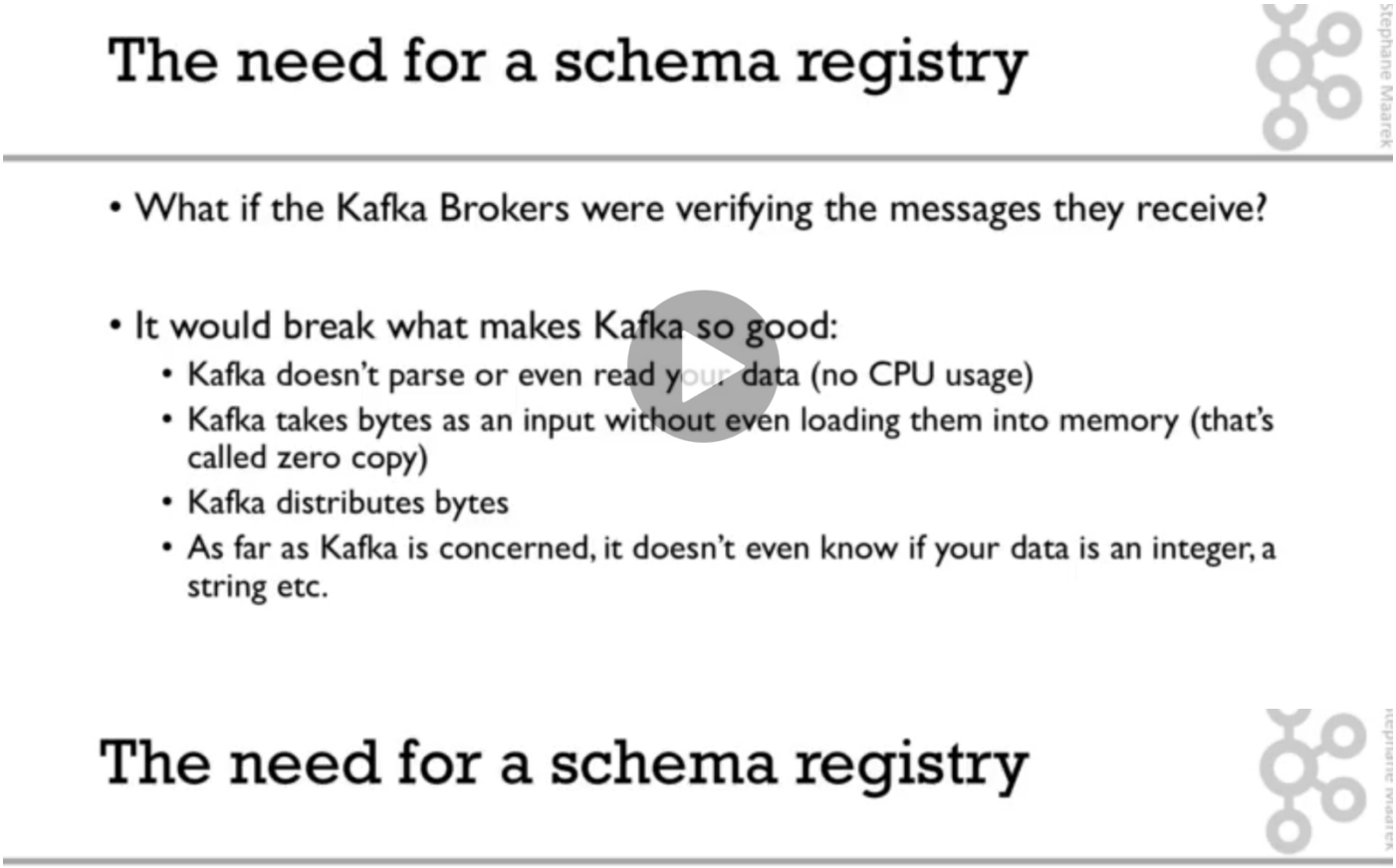
- **The Consumers Break!!!**

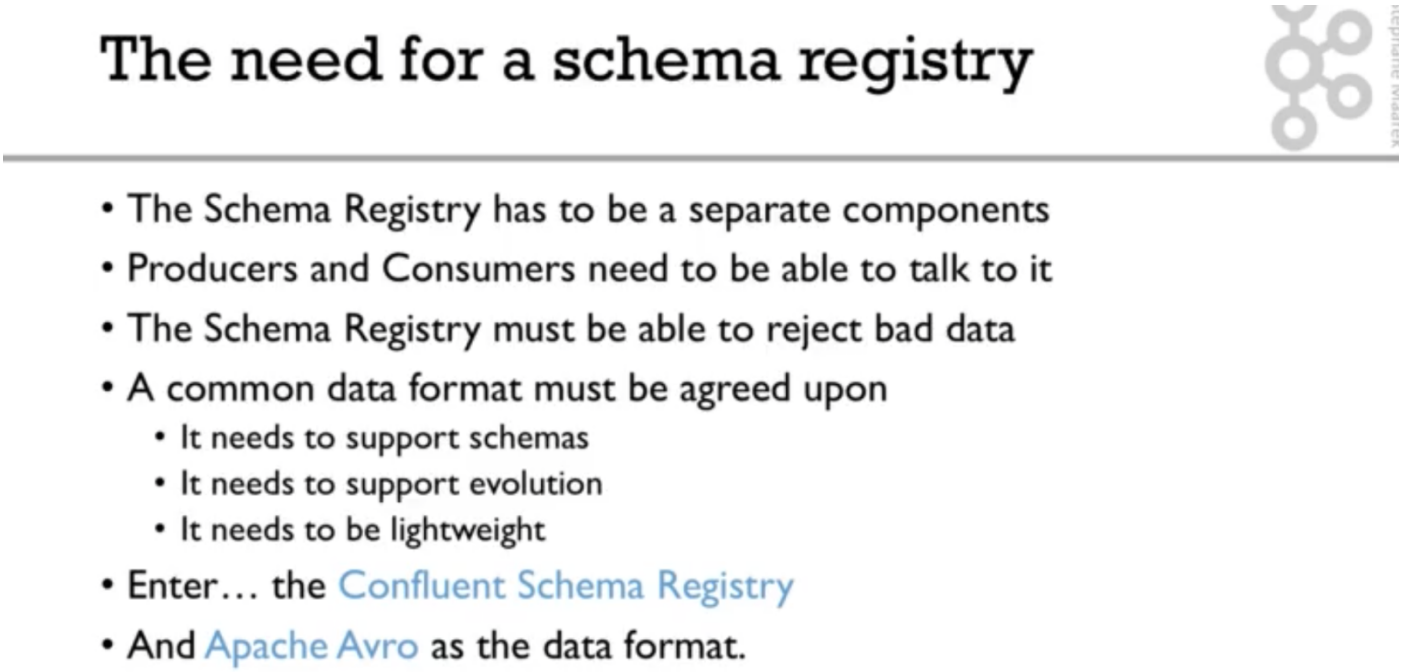
The need for a schema registry

- We need data to be self describable
- We need to be able to evolve data without breaking downstream consumers.

- We need schemas… and a schema registry!







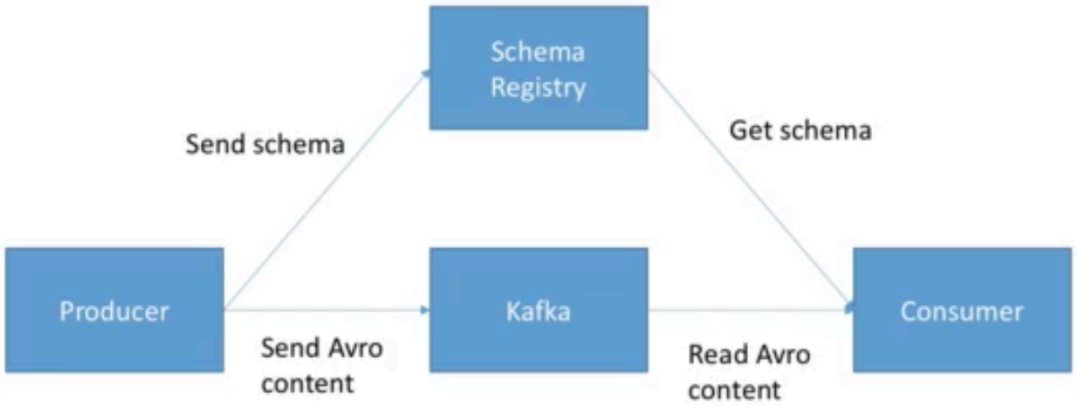
Pipeline without schema registry

Pipeline without Schema Registry

Confluent Schema Registry Purpose

Confluent Schema Registry Purpose

- Store and retrieve schemas for Producers / Consumers
- Enforce Backward / Forward / Full compatibility on topics
- Decrease the size of the payload of data sent to Kafka



Schema Registry Takeaway

Schema Registry: gotchas

- Utilizing a schema registry has a lot of benefits
- BUT it implies you need to
  - Set it up well
  - Make sure it's highly available
  - Partially change the producer and consumer code
- Apache Avro as a format is awesome but has a learning curve
- The schema registry is free and open sourced, created by Confluent (creators of Kafka)
- As it takes time to setup, we won't cover the usage in this course

Which API to Use

https://medium.com/@stephane.maarek/the-kafka-api-battle-producer-vs-consumer-vs-kafka-connect-vs-kafka-streams-vs-ksql-ef584274c1e

Question 3:

To continuously export data from Kafka into a target database, I should use

- ○ Kafka Connect Source
- ○ Kafka Streams
- ○ Kafka Consumer
- ◉ Kafka Connect Sink

Good job!

Kafka does not verify or parse incoming data

Question 4:

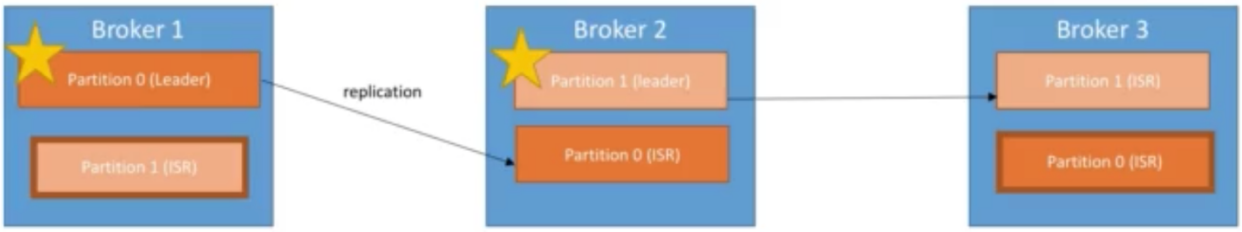
Kafka can verify our incoming data's schema and ensure it's correct

- ○ yes
- ◉ no

Partition Count and Replication Factor

Partitions Count, Replication Factor

- The two most important parameters when creating a topic.
- They impact performance and durability of the system overall



- It is best to get the parameters right the first time!
  - If the Partitions Count increases during a topic lifecycle, you will break your keys ordering guarantees
  - If the Replication Factor increases during a topic lifecycle, you put more pressure on your cluster, which can lead to unexpected performance decrease

- Guess the right count

Partition Count

Choosing partition count guidelines.

Partitions Count

- Each partition can handle a throughput of a few MB/s (measure it for your setup!)
- More partitions implies:
  - Better parallelism, better throughput
  - Ability to run more consumers in a group to scale
  - Ability to leverage more brokers if you have a large cluster
  - BUT more elections to perform for Zookeeper
  - BUT more files opened on Kafka

- Guidelines:
  - **Partitions per topic = MILLION DOLLAR QUESTION**
    - (Intuition) Small cluster (< 6 brokers): 2 x # brokers
    - (Intuition) Big cluster (> 12 brokers): 1 x # of brokers
    - Adjust for number of consumers you need to run in parallel at peak throughput
    - Adjust for producer throughput (increase if super-high throughput or projected increase in the next 2 years)
  - **TEST!** Every Kafka cluster will have different performance.
  - Don't create a topic with 1000 partitions!

Replication Factor

Replication Factor

- Should be at least 2, usually 3, maximum 4
- The higher the replication factor (N):
  - Better resilience of your system (N-1 brokers can fail)
  - BUT more replication (higher latency if acks=all)
  - BUT more disk space on your system (50% more if RF is 3 instead of 2)

- Guidelines:
  - **Set it to 3 to get started** (you must have at least 3 brokers for that)
  - If replication performance is an issue, get a better broker instead of less RF
  - **Never set it to 1 in production**

Clusters guidelines

- It is pretty much accepted that a broker should not hold more than 2000 to 4000 partitions (across all topics of that broker).
- Additionally, a Kafka cluster should have a maximum of 20,000 partitions across all brokers.
- The reason is that in case of brokers going down, Zookeeper needs to perform a lot of leader elections

- If you need more partitions in your cluster, add brokers instead
- If you need more than 20,000 partitions in your cluster (it will take time to get there!), follow the Netflix model and create more Kafka clusters.

- Overall, you don't need a topic with 1000 partitions to achieve high throughput. Start at a reasonable number and test the performance

Kafka Topic Naming Convention

https://cnr.sh/essays/how-paint-bike-shed-kafka-topic-naming-conventions

Case Study

MovieFlix

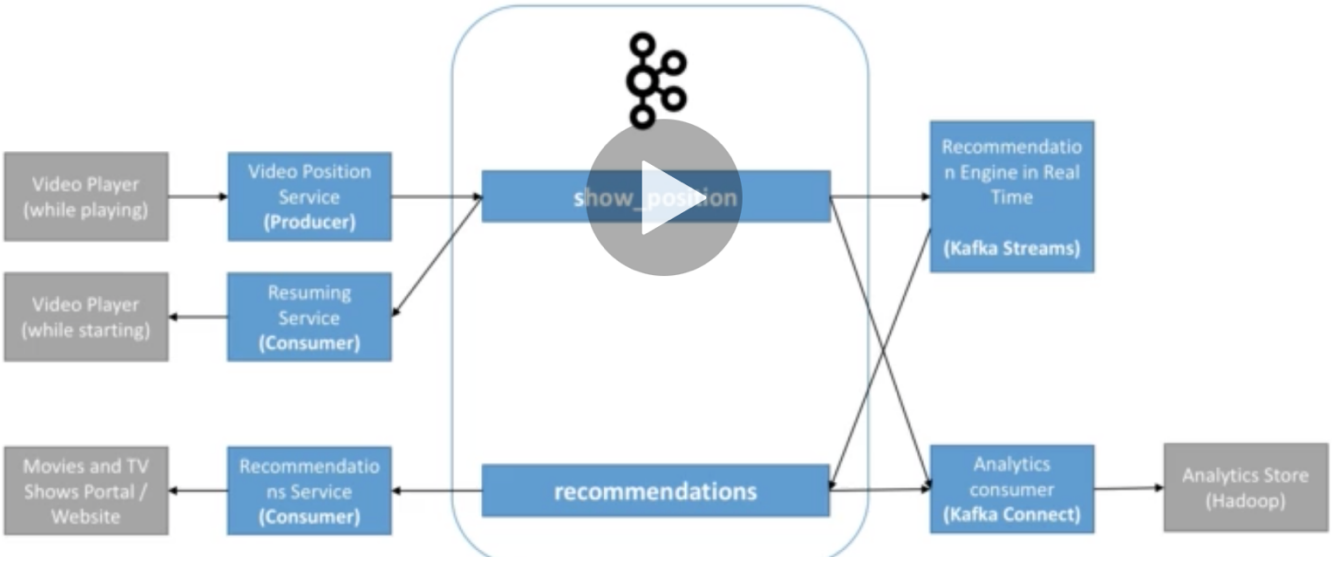
Video Analytics - MovieFlix

- *MovieFlix is a company that allows you to watch TV Shows and Movies on demand. The business wants the following capabilities:*

- Make sure the user can resume the video where they left it off
- Build a user profile in real time
- Recommend the next show to the user in real time
- Store all the data in analytics store

- How would you implement this using Kafka?

Video Analytics – MovieFlix Architecture



Video Analytics – MovieFlix Comments

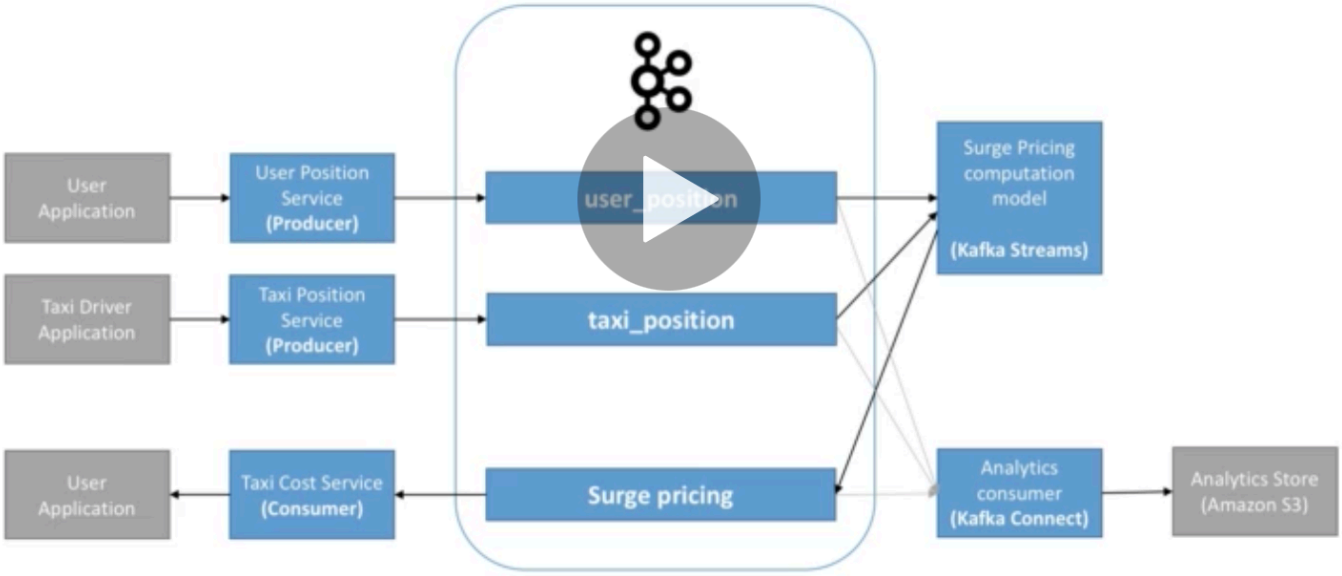
- show_position topic:
  - is a topic that can have multiple producers
  - Should be highly distributed if high volume > 30 partitions
  - If I were to choose a key, I would choose "user_id"
- recommendations topic:
  - The kafka streams recommendation engine may source data from the analytical store for historical training
  - May be a low volume topic
  - If I were to choose a key, I would choose "user_id"

GetTaxi

IOT Example - GetTaxi

- *GetTaxi is a company that allows people to match with taxi drivers on demand, right-away. The business wants the following capabilities:*
- The user should match with a close by driver
- The pricing should "surge" if the number of drivers are low or the number of users is high
- All the position data before and during the ride should be stored in an analytics store so that the cost can be computed accurately
- How would you implement this using Kafka?

IOT Example - GetTaxi Architecture



IOT Example - GetTaxi Comments

- taxi_position, user_position topic:s
  - Are topics that can have multiple producers
  - Should be highly distributed if high volume > 30 partitions
  - If I were to choose a key, I would choose "user_id","taxi_id"
  - Data is ephemeral and probably doesn't need to be kept for a long time
- surge_pricing topic:
  - The computation of Surge pricing comes from the Kafka Streams application
  - Surge pricing may be regional and therefore that topic may be high volume
  - Other topics such as "weather" or "events" etc can be included in the Kafka Streams application

Social Media

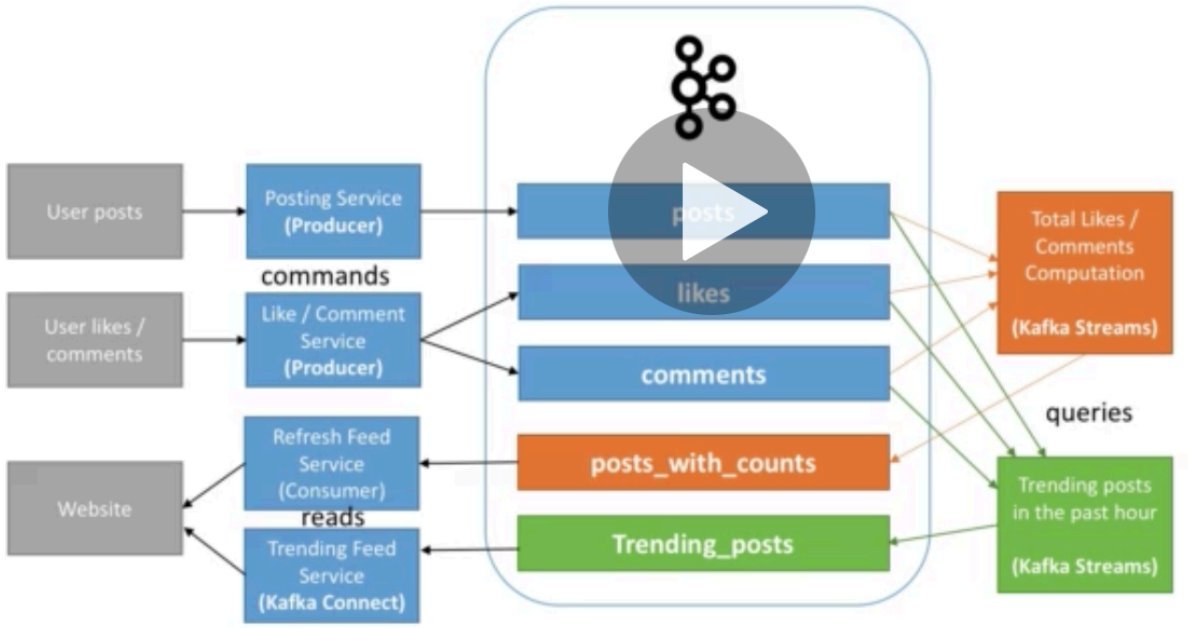
CQRS – MySocialMedia

- *MySocialMedia is a company that allows you people to post images and others to react by using "likes" and "comments". The business wants the following capabilities:*

- Users should be able to post, like and comment
- Users should see the total number of likes and comments per post in real time
- High volume of data is expected on the first day of launch
- Users should be able to see "trending" posts

- How would you implement this using Kafka?

CQRS – MySocialMedia Architecture



CQRS – MySocialMedia Comments

- Responsibilities are "segregated" hence we can call the model CQRS (Command Query Responsibility Segregation)
- Posts
  - Are topics that can have multiple producers
  - Should be highly distributed if high volume > 30 partitions
  - If I were to choose a key, I would choose "user_id"
  - We probably want a high retention period of data for this topic
- Likes, Comments
  - Are topics with multiple producers
  - Should be highly distributed as the volume of data is expected to be much greater
  - If I were to choose a key, I would choose "post_id"
- The data itself in Kafka should be formatted as "events":
  - User_123 created a post_id 456 at 2 pm
  - User_234 liked post_id 456 at 3 pm
  - User_123 deleted a post_id 456 at 6 pm

My Bank

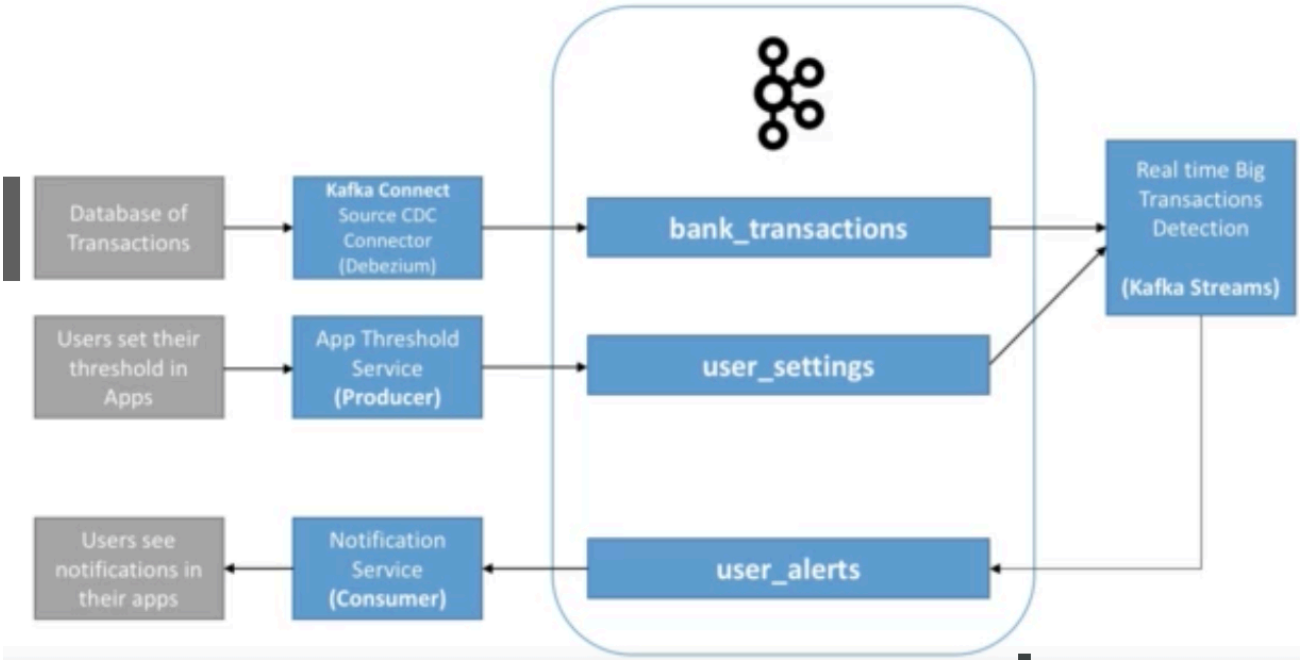
Finance application - MyBank

- *MyBank is a company that allows real-time banking for its users. It wants to deploy a brand new capability to alert users in case of large transactions*

- The transaction data already exists in a database
- Thresholds can be defined by the users
- Alerts must be sent in real time to the users

- How would you implement this using Kafka?

Finance application – MyBank Architecture



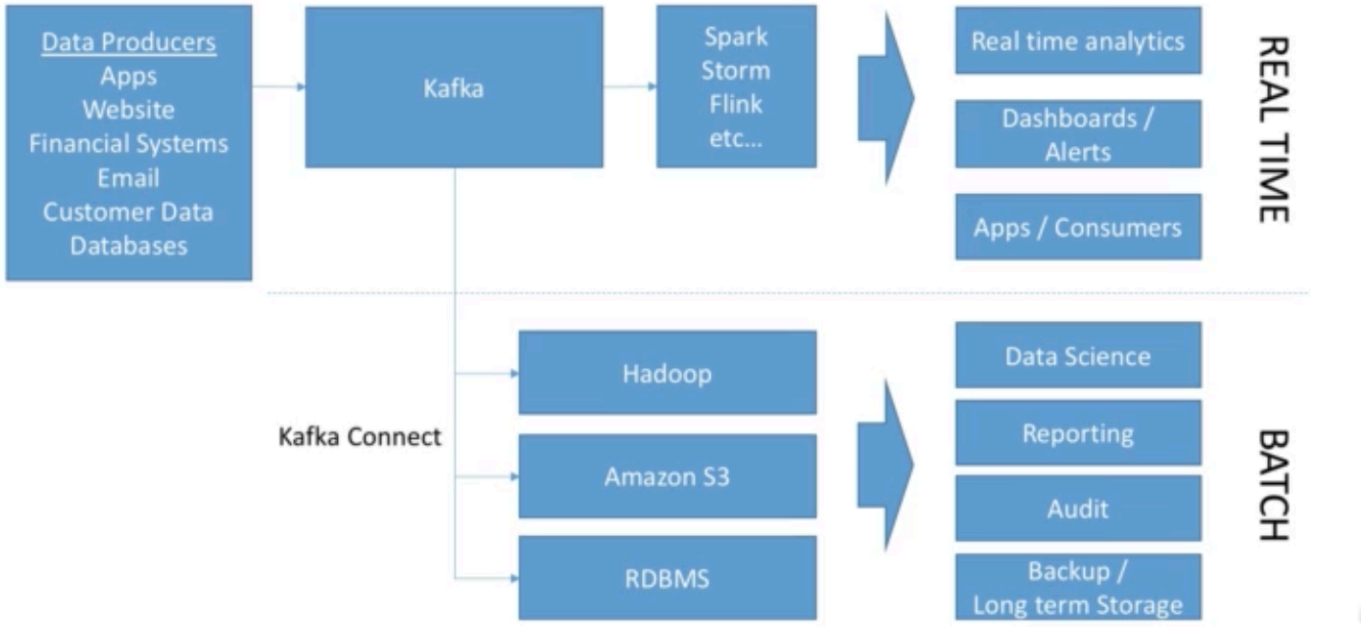
Finance application – MyBank Comments

- Bank Transactions topics:
  - Kafka Connect Source is a great way to expose data from existing databases!
  - There are tons of CDC (change data capture) connectors for technologies such as PostgreSQL, Oracle, MySQL, SQLServer, MongoDB etc…
- Kafka Streams application:
  - When a user changes their settings, alerts won't be triggered for past transactions
- User thresholds topics:
  - It is better to send events to the topic (User 123 enabled threshold at $1000 at 12 pm on July 12th 2018)
  - Than sending the state of the user: (User 123: threshold $1000)

Big Data Ingestion

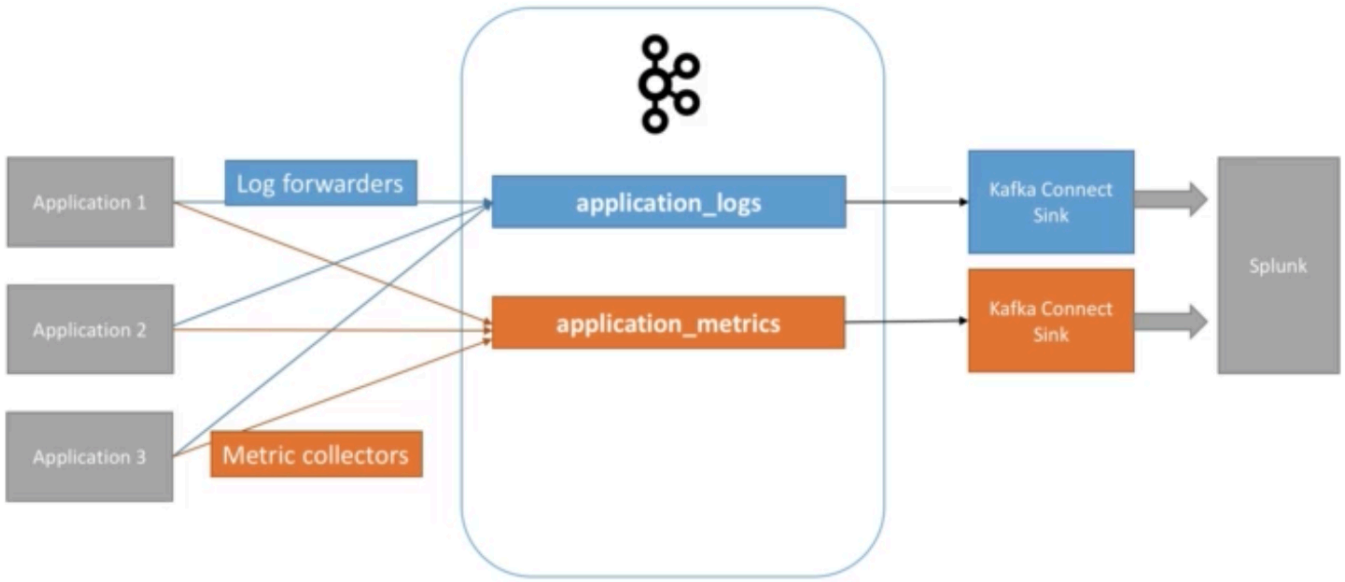
- It is common to have "generic" connectors or solutions to offload data from Kafka to HDFS, Amazon S3, and ElasticSearch for example
- It is also very common to have Kafka serve a "speed layer" for real time applications, while having a "slow layer" which helps with data ingestions into stores for later analytics
- Kafka as a front to Big Data Ingestion is a common pattern in Big Data to provide an "ingestion buffer" in front of some stores

Big Data Ingestion



Logging and Metrics Aggregation

Logging & Metrics Aggregation



Kafka Cluster Setup

Kafka Cluster Setup
High Level Architecture

- You want multiple brokers in different data centers (racks) to distribute your load. You also want a cluster of at least 3 zookeeper
- In AWS:

Kafka Cluster Setup Gotchas

- It's not easy to setup a cluster
- You want to isolate each Zookeeper & Broker on separate servers
- Monitoring needs to be implemented
- Operations have to be mastered
- You need a really good Kafka Admin

- Alternative: many different "Kafka as a Service" offerings on the web
- No operational burdens (updates, monitoring, setup, etc...)

Kafka Monitoring and Operations

Kafka Monitoring and Operations

- Kafka exposes metrics through JMX.
- These metrics are highly important for monitoring Kafka, and ensuring the systems are behaving correctly under load.
- Common places to host the Kafka metrics:
  - ELK (ElasticSearch + Kibana)
  - Datadog
  - NewRelic
  - Confluent Control Centre
  - Promotheus
  - Many others…!

Kafka Monitoring and Operations

- Some of the most important metrics are:
- Under Replicated Partitions: Number of partitions are have problems with the ISR (in-sync replicas). May indicate a high load on the system
- Request Handlers: utilization of threads for IO, network, etc… overall utilization of an Apache Kafka broker.
- Request timing: how long it takes to reply to requests. Lower is better, as latency will be improved.

https://kafka.apache.org/documentation/#monitoring

https://docs.confluent.io/platform/current/kafka/monitoring.html

https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/

Kafka Monitoring and Operations

- Kafka Operations team must be able to perform the following tasks:
  - Rolling Restart of Brokers
  - Updating Configurations
  - Rebalancing Partitions
  - Increasing replication factor
  - Adding a Broker
  - Replacing a Broker
  - Removing a Broker
  - Upgrading a Kafka Cluster with zero downtime

Kafka Security

The need for encryption, authentication & authorization in Kafka

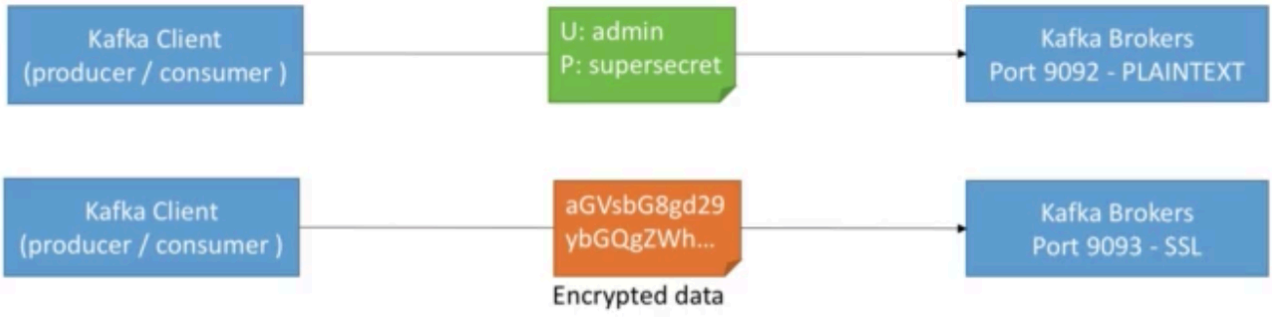
- Currently, any client can access your Kafka cluster (authentication)
- The clients can publish / consume any topic data (authorisation)
- All the data being sent is fully visible on the network (encryption)

- Someone could intercept data being sent
- Someone could publish bad data / steal data
- Someone could delete topics

- All these reasons push for more security and an authentication model

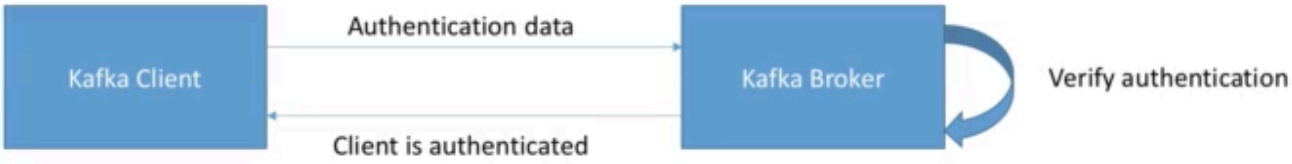
Encryption in Kafka

- Encryption in Kafka ensures that the data exchanged between clients and brokers is secret to routers on the way
- This is similar concept to an https website



Authentication in Kafka

- Authentication in Kafka ensures that only clients that can prove their identity can connect to our Kafka Cluster
- This is similar concept to a login (username / password)



Authentication in Kafka

- Authentication in Kafka can take a few forms
- SSL Authentication: clients authenticate to Kafka using SSL certificates
- SASL Authentication:
  - PLAIN: clients authenticate using username / password (weak – easy to setup)
  - Kerberos: such as Microsoft Active Directory (strong – hard to setup)
  - SCRAM: username / password (strong – medium to setup)

Authorisation in Kafka

- Once a client is authenticated, Kafka can verify its identity
- It still needs to be combined with authorisation, so that Kafka knows that
  - "User alice can view topic finance"
  - "User bob cannot view topic trucks"
- ACL (Access Control Lists) have to be maintained by administration and onboard new users

Putting it all together

- You can mix
  - Encryption
  - Authentication
  - Authorisation
- This allows your Kafka clients to:
  - Communicate securely to Kafka
  - Clients would authenticate against Kafka
  - Kafka can authorise clients to read / write to topics

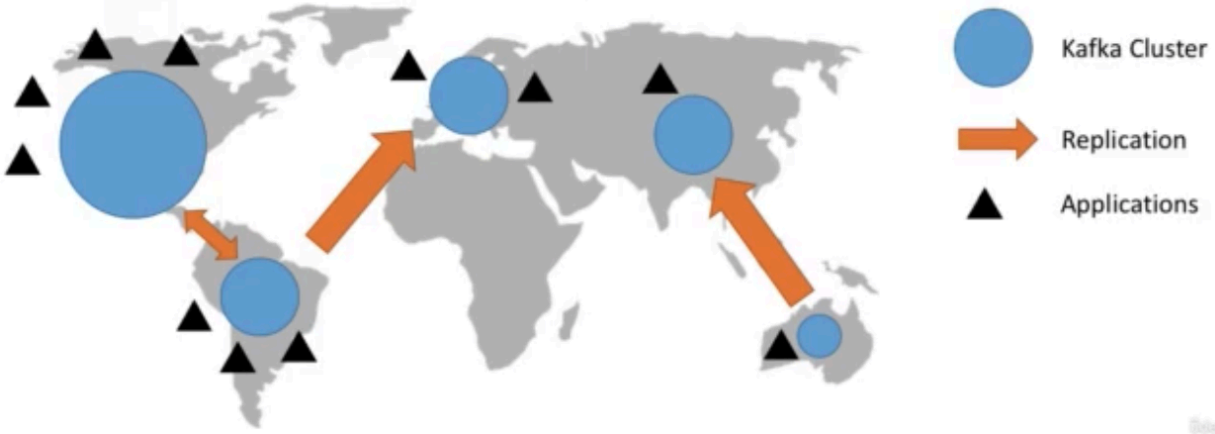
State of the art of Kafka Security

- Kafka Security is fairly new (0.10)
- Kafka Security improves over time and becomes more flexible / easier to setup as time goes.
- Currently, it is **hard** to setup Kafka Security.
- Best support for Kafka Security for applications is with **Java**.

Kafka Multi Cluster + Replication

Kafka Multi Cluster + Replication

- Kafka can only operate well in a single region
- Therefore, it is very common for enterprises to have Kafka clusters across the world, with some level of replication between them

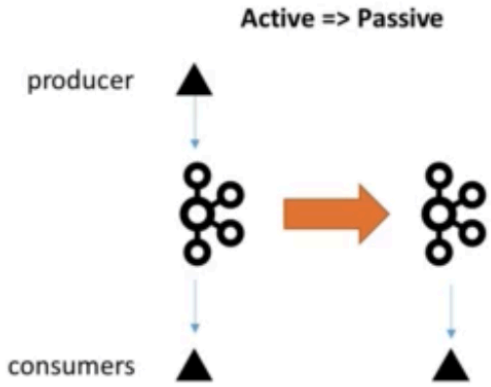


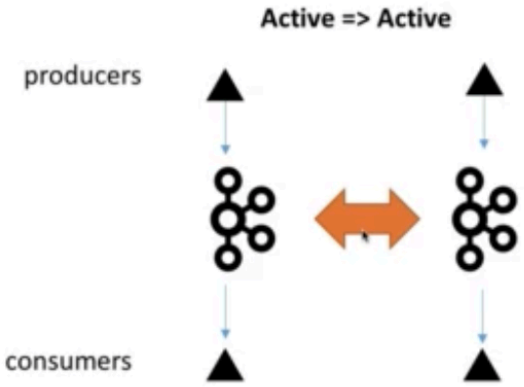
Kafka Multi Cluster + Replication

- A replication application at its core is just a consumer + a producer
- There are different tools to perform it:
  - Mirror Maker - open source tool that ships with Kafka
  - Netflix uses Flink – they wrote their own application
  - Uber uses uReplicator – addresses performance and operations issues with MM
  - Comcast has their own open source Kafka Connect Source
  - Confluent has their own Kafka Connect Source (paid)
- Overall, try these and see if it works for your use case before writing your own

Kafka Multi Cluster + Replication

- There are two designs for cluster replication:





Kafka Multi Cluster + Replication

- Active => Active:
  - You have a global application
  - You have a global dataset
- Active => Passive:
  - You want to have an aggregation cluster (for example for analytics)
  - You want to create some form of disaster recovery strategy (*it's hard)
  - Cloud Migration (from on-premise cluster to Cloud cluster)
- **Replicating doesn't preserve offsets, just data!**

https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=27846330

https://engineering.salesforce.com/mirrormaker-performance-tuning-63afaed12c21

https://docs.confluent.io/platform/current/multi-dc-deployments/replicator/replicator-tuning.html#improving-network-utilization-of-a-connect-task

https://community.cloudera.com/t5/Community-Articles/Kafka-Mirror-Maker-Best-Practices/ta-p/249269

https://www.confluent.io/kafka-summit-sf17/multitenant-multicluster-and-hieracrchical-kafka-messaging-service/

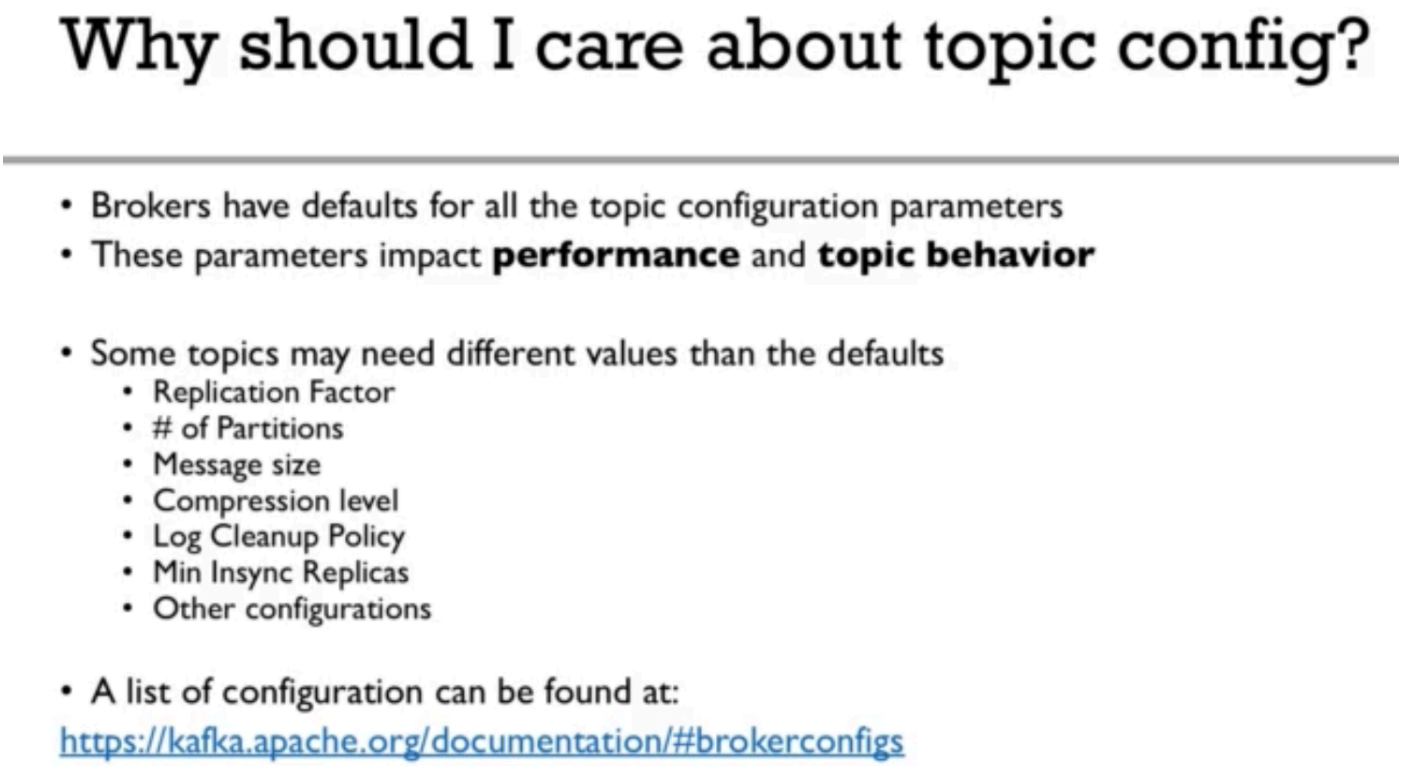
https://eng.uber.com/ureplicator-apache-kafka-replicator/

https://www.altoros.com/blog/multi-cluster-deployment-options-for-apache-kafka-pros-and-cons/

https://github.com/Comcast/MirrorTool-for-Kafka-Connect

Advanced Kafka

Topic Configuration



Created a new topic “configured-topic” for configuration.

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic configured_topic --create --partitions 3 --replication-factor 1
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is b
est to use either, but not both.
Created topic configured_topic.
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic configured_topic --describe
Topic: configured_topic TopicId: 2EPF4Eb4QnyOI_z5V5o1_A PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: configured_topic Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 2    Leader: 0       Replicas: 0     Isr: 0
➜  ~
```

Describing topic configuration

kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic –describe

```
➜  ~ kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --describe
Warning: --zookeeper is deprecated and will be removed in a future version of Kafka.
Use --bootstrap-server instead to specify a broker to connect to.
Configs for topic 'configured_topic' are
➜  ~
```

Adding Topic Configuration

kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --add-config min.insync.replicas=2 --alter

```
→  ~ kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --add-config min.insync.replicas=2
 --alter
Warning: --zookeeper is deprecated and will be removed in a future version of Kafka.
Use --bootstrap-server instead to specify a broker to connect to.
Completed updating config for entity: topic 'configured_topic'.
```

```
→  ~ kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --describe
Warning: --zookeeper is deprecated and will be removed in a future version of Kafka.
Use --bootstrap-server instead to specify a broker to connect to.
Configs for topic 'configured_topic' are min.insync.replicas=2
```

```
→  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic configured_topic --describe
Topic: configured_topic TopicId: 2EPF4Eb4QnyOI_z5V5o1_A PartitionCount: 3       ReplicationFactor: 1    Configs: min.insync.replicas=2
        Topic: configured_topic Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 2    Leader: 0       Replicas: 0     Isr: 0
→  ~
```

Deleting Topic Configuration

kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --delete-config min.insync.replicas --alter

```
→  ~ kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --delete-config min.insync.replicas --alter
Warning: --zookeeper is deprecated and will be removed in a future version of Kafka.
Use --bootstrap-server instead to specify a broker to connect to.
Completed updating config for entity: topic 'configured_topic'.
→  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic configured_topic --describe
Topic: configured_topic TopicId: 2EPF4Eb4QnyOI_z5V5o1_A PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: configured_topic Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: configured_topic Partition: 2    Leader: 0       Replicas: 0     Isr: 0
→  ~
```

Segment and Indexes

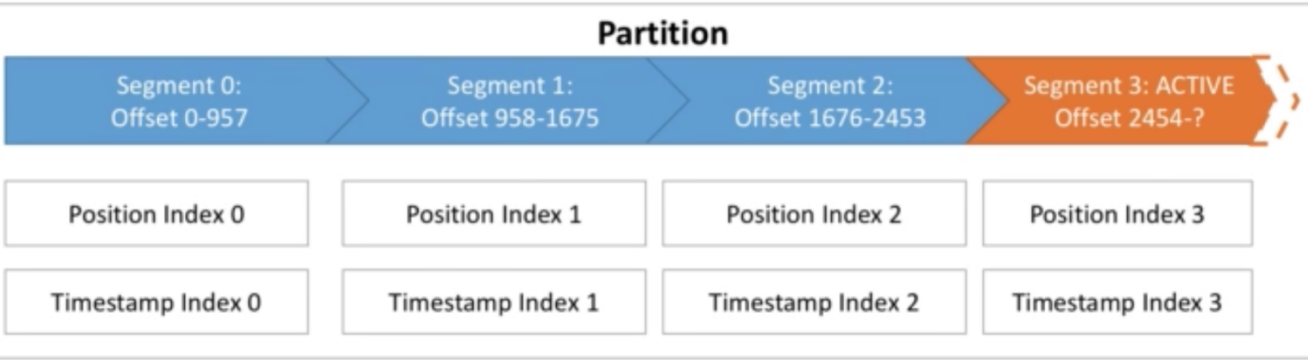
Partitions and Segments

- Topics are made of partitions (we already know that)
- Partitions are made of… segments (files)!

- Only one segment is ACTIVE (the one data is being written to)
- Two segment settings:
  - log.segment.bytes: the max size of a single segment in bytes
  - log.segment.ms: the time Kafka will wait before committing the segment if not full

Segments and Indexes

- Segments come with two indexes (files):
  - An offset to position index: allows Kafka where to read to find a message
  - A timestamp to offset index: allows Kafka to find messages with a timestamp
- Therefore, Kafka knows where to find data in a constant time!



Segments: Why should I care?

- A smaller **log.segment.bytes** (size, default: 1GB) means:
  - More segments per partitions
  - Log Compaction happens more often
  - BUT Kafka has to keep more files opened (*Error: Too many open files*)
- Ask yourself: how fast will I have new segments based on throughput?

- A smaller **log.segment.ms** (time, default 1 week) means:
  - You set a max frequency for log compaction (more frequent triggers)
  - Maybe you want daily compaction instead of weekly?
- Ask yourself: how often do I need log compaction to happen?

Log Cleanup Policies

Log Cleanup Policies

- Many Kafka clusters make data expire, according to a policy
- That concept is called log cleanup.

Policy1: log.cleanup.policy=delete (Kafka default for all user topics)
- Delete based on age of data (default is a week)
- Delete based on max size of log (default is -1 == infinite)

Policy 2: log.cleanup.policy=compact (Kafka default for topic *__consumer_offsets*)
- Delete based on keys of your messages
- Will delete old duplicate keys **after** the active segment is committed
- Infinite time and space retention

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic __consumer_offsets --describe
Topic: __consumer_offsets       TopicId: cTdobg5QTeieGTLamAbssg PartitionCount: 50      ReplicationFactor: 1    Configs: compression
.type=producer,cleanup.policy=compact,segment.bytes=104857600
        Topic: __consumer_offsets       Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 7    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 8    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 9    Leader: 0       Replicas: 0     Isr: 0
        Topic: __consumer_offsets       Partition: 10   Leader: 0       Replicas: 0     Isr: 0
```

Log Cleanup: Why and When?

- Deleting data from Kafka allows you to:
  - Control the size of the data on the disk, delete obsolete data
  - Overall: Limit maintenance work on the Kafka Cluster
- How often does log cleanup happen?
  - Log cleanup happens on your partition segments!
  - Smaller / More segments means that log cleanup will happen more often!
  - Log cleanup shouldn't happen too often => takes CPU and RAM resources
  - The cleaner checks for work every 15 seconds (log.cleaner.backoff.ms)

Delete

Log Cleanup Policy: Delete

- log.retention.hours:
  - number of hours to keep data for (default is 168 – one week)
  - Higher number means more disk space
  - Lower number means that less data is retained (if your consumers are down for too long, they can miss data)
- log.retention.bytes:
  - Max size in Bytes for each partition (default is -1 – infinite)
  - Useful to keep the size of a log under a threshold

Log Cleanup Policy: Delete

Old segment will be deleted based on time or space rules

New data is written to the active segment

Use cases - two common pair of options:

- One week of retention:
  - log.retention.hours=168 and log.retention.bytes=-1
- Infinite time retention bounded by 500MB:
  - log.retention.hours=17520 and log.retention.bytes=524288000

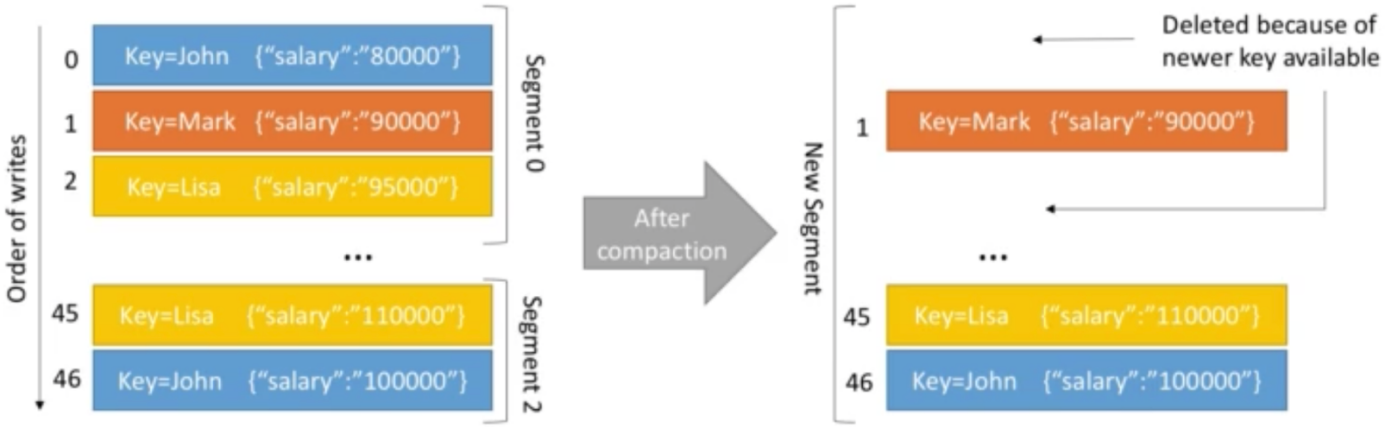
Compaction

Log Cleanup Policy: Compact

- Log compaction ensures that your log contains at least the last known value for a specific key within a partition
- Very useful if we just require a SNAPSHOT instead of full history (such as for a data table in a database)
- The idea is that we only keep the latest "update" for a key in our log

Log Compaction: Example

- Our topic is: employee-salary
- We want to keep the most recent salary for our employees



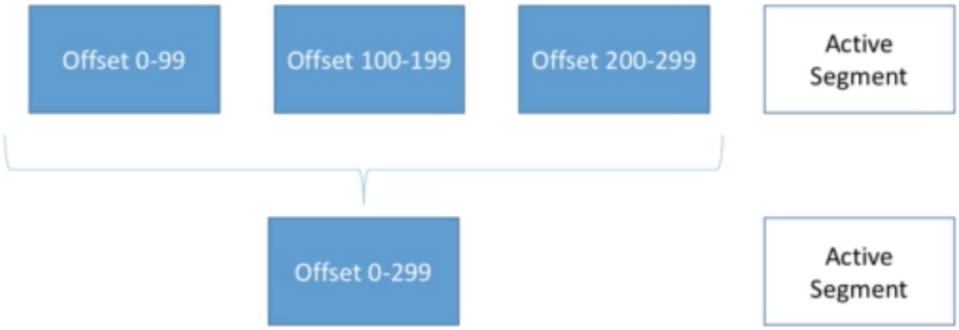
Log Compaction Guarantees

- **Any consumer that is reading from the tail of a log (most current data) will still see all the messages sent to the topic**
- Ordering of messages it kept, log compaction only removes some messages, but does not re-order them
- The offset of a message is immutable (it never changes). Offsets are just skipped if a message is missing
- Deleted records can still be seen by consumers for a period of delete.retention.ms (default is 24 hours).

Log Compaction Myth Busting

- It doesn't prevent you from pushing duplicate data to Kafka
  - De-duplication is done after a segment is committed
  - Your consumers will still read from tail as soon as the data arrives
- It doesn't prevent you from reading duplicate data from Kafka
  - Same points as above
- Log Compaction can fail from time to time
  - It is an optimization and it the compaction thread might crash
  - Make sure you assign enough memory to it and that it gets triggered
  - Restart Kafka if log compaction is broken (this is a bug and may get fixed in the future)
- You can't trigger Log Compaction using an API call (for now…)

Log Compaction



- Log compaction is configured by (log.cleanup.policy=compact):
  - Segment.ms (default 7 days): Max amount of time to wait to close active segment
  - Segment.bytes (default 1G): Max size of a segment
  - Min.compaction.lag.ms (default 0): how long to wait before a message can be compacted
  - Delete.retention.ms (default 24 hours): wait before deleting data marked for compaction
  - Min.Cleanable.dirty.ratio (default 0.5): higher => less, more efficient cleaning. Lower => opposite

Compaction Example

Step 1: Creating a topic named “employee salary” with following configuration.

1. cleanup.policy=compact
2. min.cleanable.dirty.ratio=0.001
3. segment.ms=5000

kafka-topics --zookeeper 0.0.0.0:2181 --topic employee-salary --create --partitions 1 --replication-factor 1 --config cleanup.policy=compact --config min.cleanable.dirty.ratio=0.001 --config segment.ms=5000

```
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic employee-salary --create --partitions 1 --replication-factor 1 --config cleanup.p
olicy=compact --config min.cleanable.dirty.ratio=0.001 --config segment.ms=5000
Created topic employee-salary.
➜  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic employee-salary --describe
Topic: employee-salary  TopicId: IWjVKG0ZQBmYhst-pMsKGA PartitionCount: 1       ReplicationFactor: 1    Configs: cleanup.policy=comp
act,segment.ms=5000,min.cleanable.dirty.ratio=0.001
        Topic: employee-salary  Partition: 0    Leader: 0       Replicas: 0     Isr: 0
➜  ~
```

In Kafka Log

```
[2021-09-07 14:43:56,920] INFO Created log for partition employee-salary-0 in /Users/naveen.kumar1/Softwares/Kafka/kafka_2.13-2.8.0/data/kafka/employee-salary-0 with properties {compression.type -> producer, min.insync.replicas -> 1, message.downconversion.enable -> true, segment.jitter.ms -> 0, cleanup.policy -> compact, flush.ms -> 9223372036854775807, retention.ms -> 604800000, segment.bytes -> 1073741824, flush.messages -> 9223372036854775807, message.format.version -> 2.8-IV1, max.compaction.lag.ms -> 9223372036854775807, file.delete.delay.ms -> 60000, max.message.bytes -> 1048588, min.compaction.lag.ms -> 0, message.timestamp.type -> CreateTime, preallocate -> false, index.interval.bytes -> 4096, min.cleanable.dirty.ratio -> 0.001, unclean.leader.election.enable -> false, retention.bytes -> -1, delete.retention.ms -> 86400000, segment.ms -> 5000, message.timestamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760}. (kafka.log.LogManager)
```

Step 2: Running the consumer and producer

kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic employee-salary --from-beginning --property print.key=true --property key.separator=,

kafka-console-producer --broker-list 192.168.1.100:9092 --topic employee-salary --property parse.key=true --property key.separator=,

```
➜  ~ kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic employee-salary --from-beginning --property print.key=true --property key.separator=,
A,1000
B,1000
C,1000
```

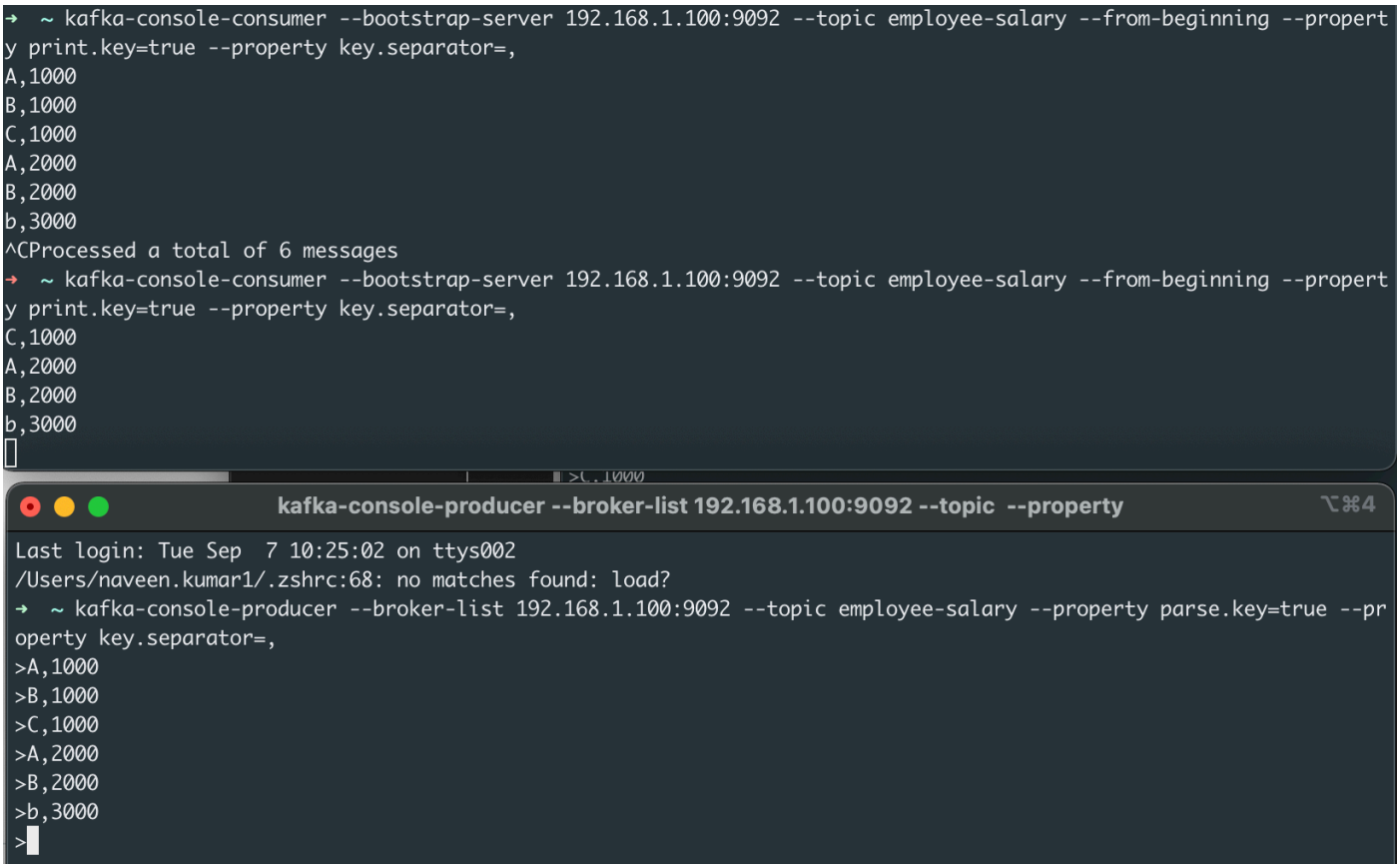
```
kafka-console-producer --broker-list 192.168.1.100:9092 --topic  --property
Last login: Tue Sep  7 10:25:02 on ttys002
/Users/naveen.kumar1/.zshrc:68: no matches found: load?
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic employee-salary --property parse.key=true --property key.separator=,
>A,1000
>B,1000
>C,1000
>
```

Step 3: Produce the new value for the key which already existing.

```
➜  ~ kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic employee-salary --from-beginning --property print.key=true --property key.separator=,
A,1000
B,1000
C,1000
A,2000
B,2000
b,3000
```

```
kafka-console-producer --broker-list 192.168.1.100:9092 --topic  --property
Last login: Tue Sep  7 10:25:02 on ttys002
/Users/naveen.kumar1/.zshrc:68: no matches found: load?
➜  ~ kafka-console-producer --broker-list 192.168.1.100:9092 --topic employee-salary --property parse.key=true --property key.separator=,
>A,1000
>B,1000
>C,1000
>A,2000
>B,2000
>b,3000
>
```

Step 4: Turning off and on consumer for compaction.



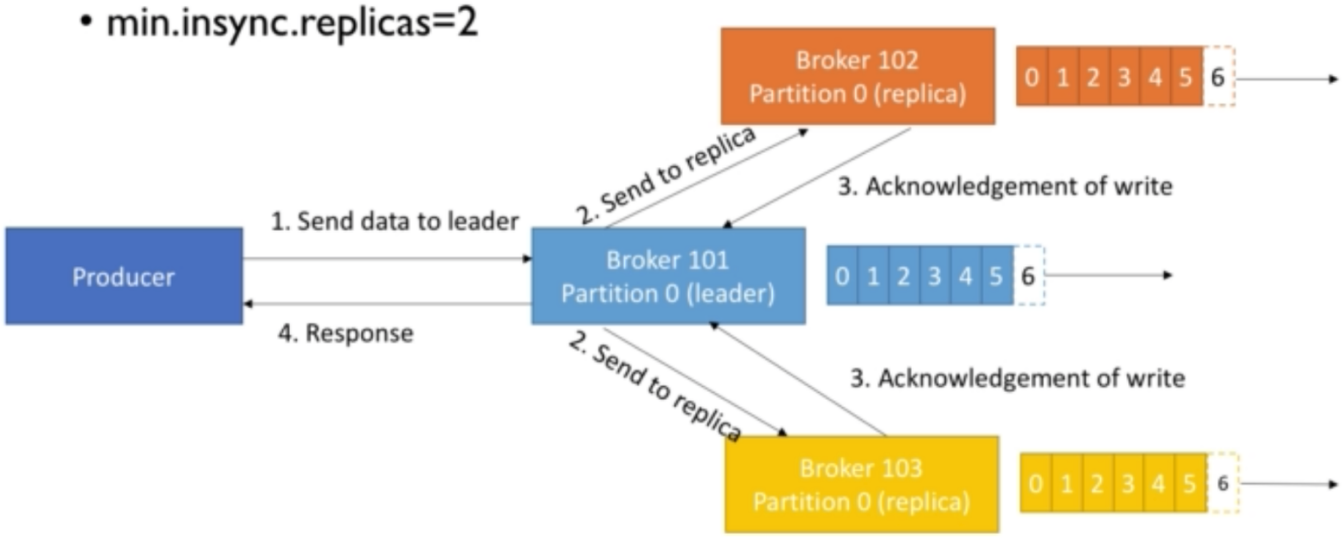
min.insync.replication

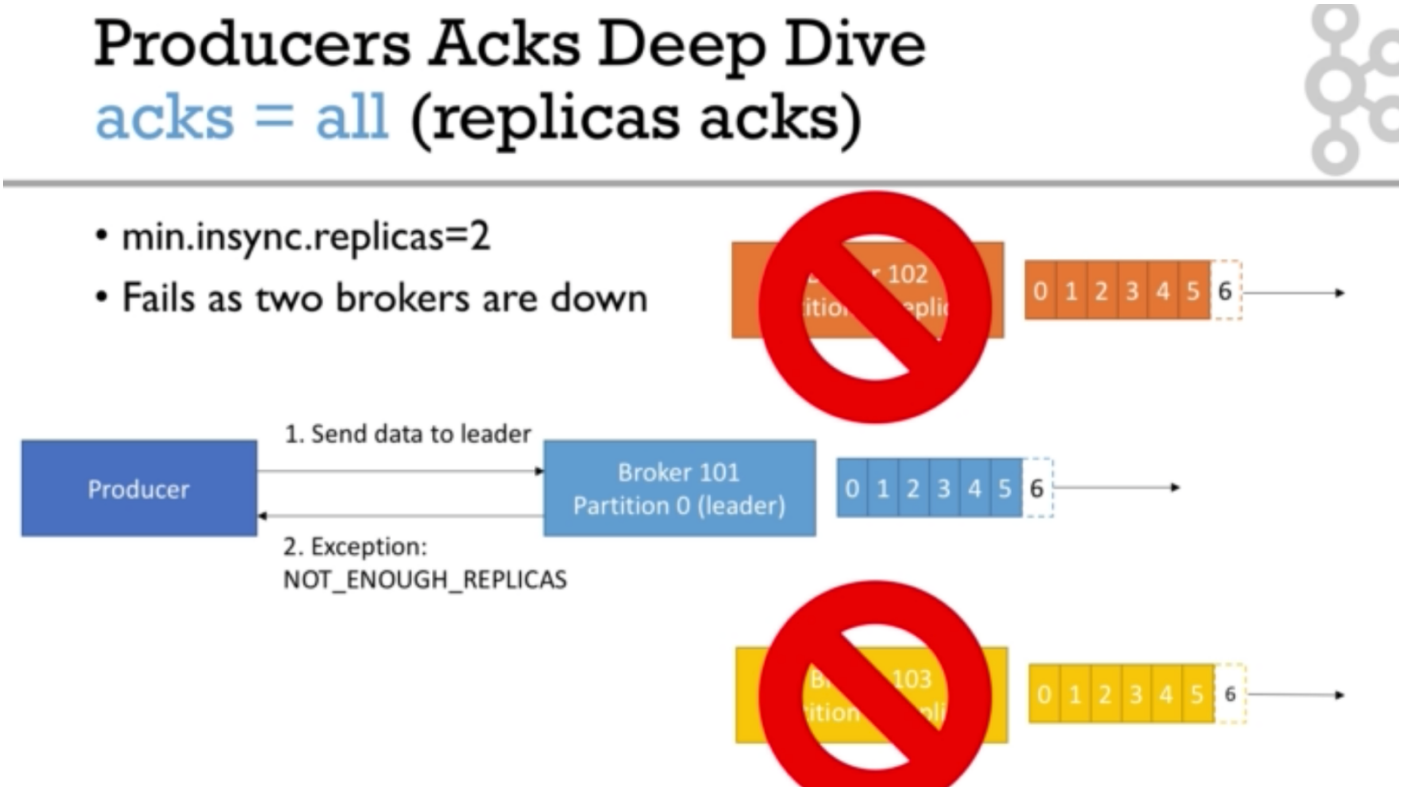
min.insync.replicas

- Acks=all must be used in conjunction with min.insync.replicas.
- min.insync.replicas can be set at the broker or topic level (override).
- min.insync.replicas=2 implies that at least 2 brokers that are ISR (including leader) must respond that they have the data.
- That means if you use replication.factor=3, min.insync=2, acks=all, you can only tolerate 1 broker going down, otherwise the producer will receive an exception on send.

min.insync.replicas

- min.insync.replicas=2





Add min.insync.replicas

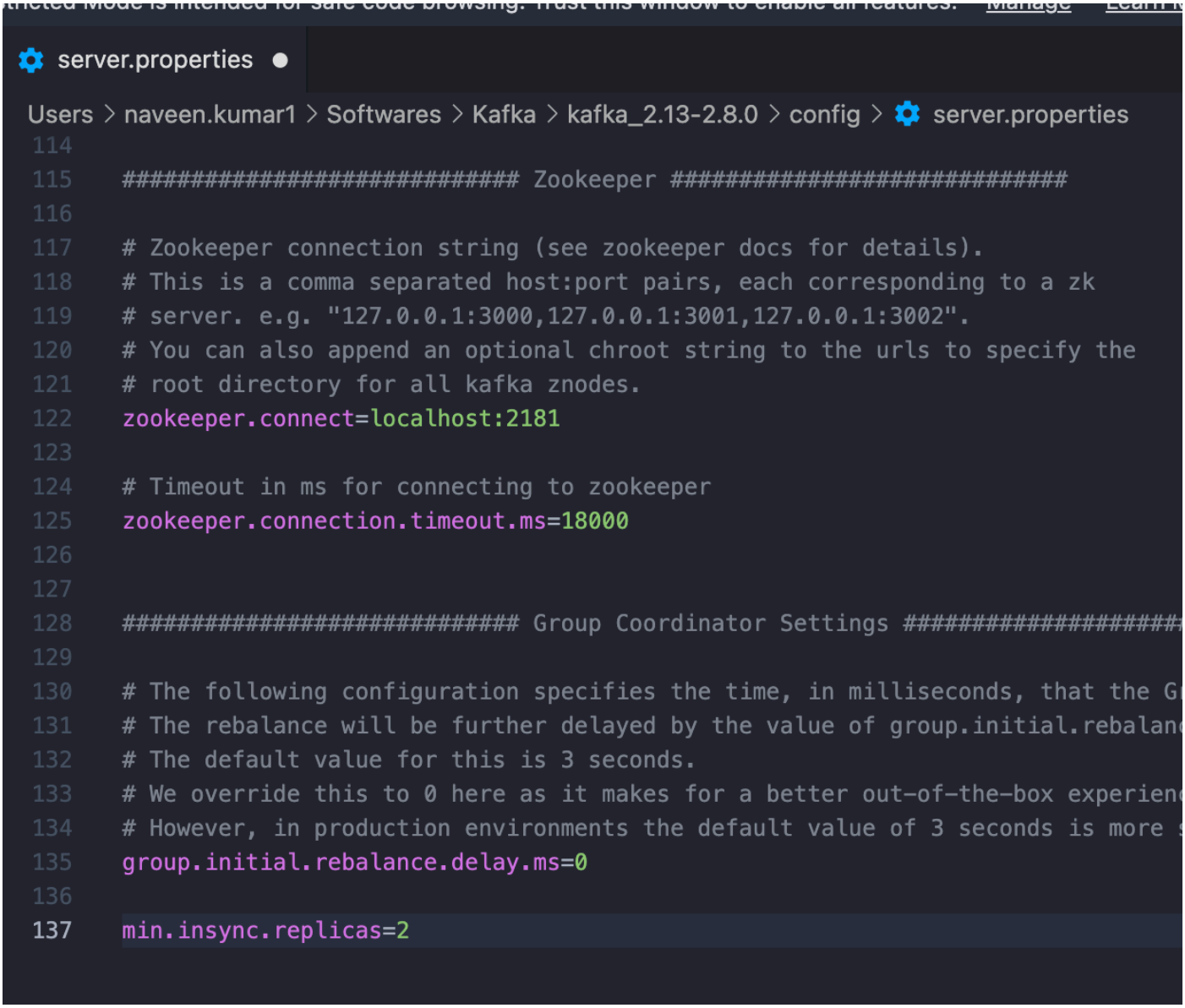
Method 1:

kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --add-config min.insync.replicas=2 --alter

```
→  ~ kafka-configs --zookeeper 0.0.0.0:2181 --entity-type topics --entity-name configured_topic --add-config min.insync.replicas=2
 --alter
Warning: --zookeeper is deprecated and will be removed in a future version of Kafka.
Use --bootstrap-server instead to specify a broker to connect to.
Completed updating config for entity: topic 'configured_topic'.
```

Method 2:

Through server.properties file.



Unclean Leader Election

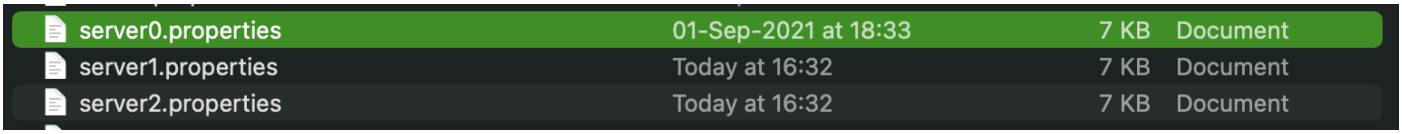
unclean.leader.election

- If all your In Sync Replicas die (but you still have out of sync replicas up), you have the following option:
  - Wait for an ISR to come back online (default)
  - Enable unclean.leader.election=true and start producing to non ISR partitions
- If you enable unclean.leader.election=true, you improve availability, but you will lose data because other messages on ISR will be discarded.
- Overall this is a very dangerous setting and its implication must be understood fully before enabling it.
- Use cases include: metrics collection, log collection, and other cases where data loss is somewhat acceptable, at the trade-off of availability.

Annexes

Kafka Cluster Hands On

Step 1: Configuring 3 different Property File for Kafka server.



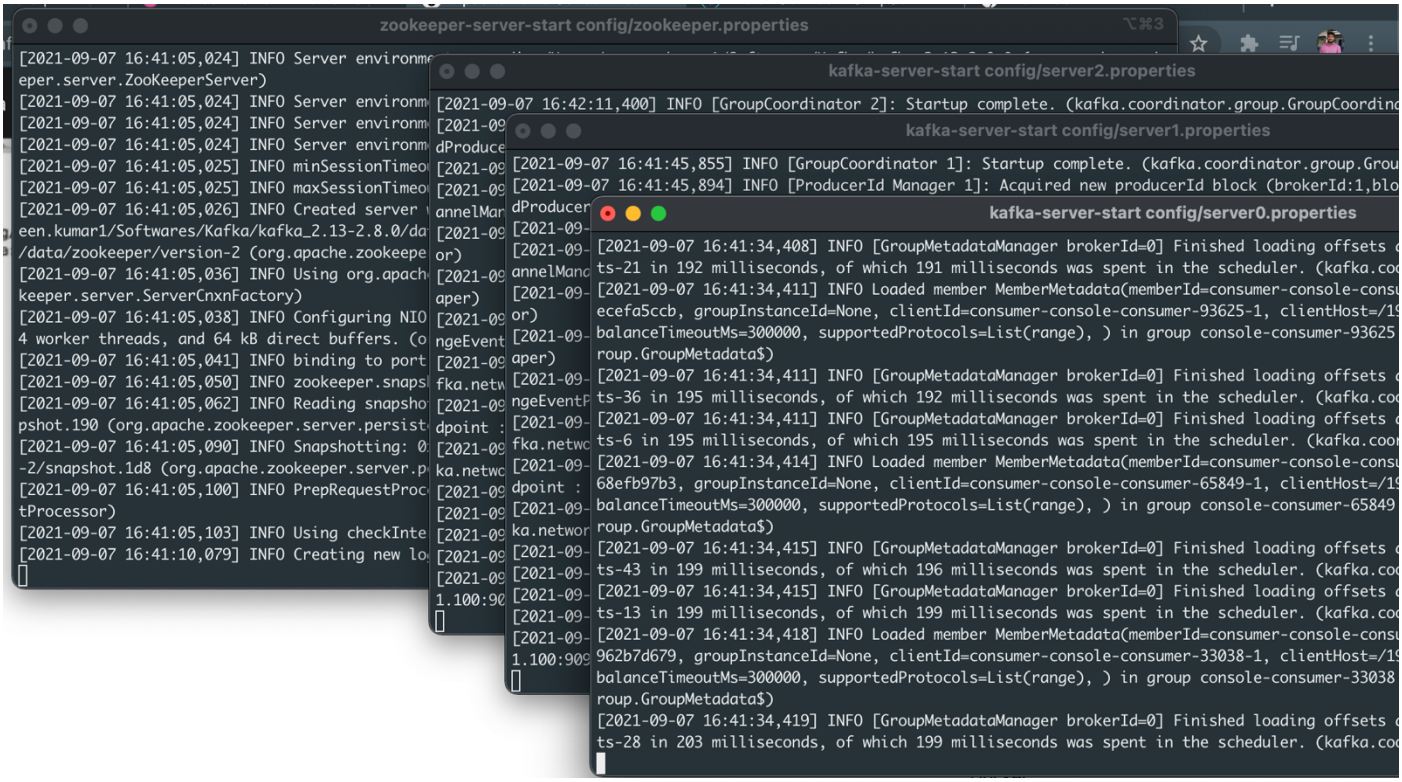
Changed the following properties

- broker.id=2
- listeners=PLAINTEXT://:9094
- log.dirs=/Users/naveen.kumar1/Softwares/Kafka/kafka_2.13-2.8.0/data/kafka2

And created the folder accordingly

Step 2: Starting the three different kafka servers using following commands.

kafka-server-start config/server0.properties
kafka-server-start config/server1.properties
kafka-server-start config/server2.properties



Step 3: Create a new topic with 6 partition and 3 replicas

kafka-topics --zookeeper 0.0.0.0:2181 --topic many_reps --create --partitions 6 --replication-factor 3

```
→  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic many_reps --create --partitions 6 --replication-factor 3
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid i
est to use either, but not both.
Created topic many_reps.
→  ~ kafka-topics --zookeeper 0.0.0.0:2181 --topic many_reps --describe
Topic: many_reps        TopicId: xAhzGkLMTkGIyoQqdRKuQg PartitionCount: 6       ReplicationFactor: 3    Configs:
        Topic: many_reps        Partition: 0    Leader: 0       Replicas: 0,2,1 Isr: 0,2,1
        Topic: many_reps        Partition: 1    Leader: 1       Replicas: 1,0,2 Isr: 1,0,2
        Topic: many_reps        Partition: 2    Leader: 2       Replicas: 2,1,0 Isr: 2,1,0
        Topic: many_reps        Partition: 3    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
        Topic: many_reps        Partition: 4    Leader: 1       Replicas: 1,2,0 Isr: 1,2,0
        Topic: many_reps        Partition: 5    Leader: 2       Replicas: 2,0,1 Isr: 2,0,1
→  ~
```

Step 4: Creating a producer with all the three brokers

kafka-console-producer --broker-list 192.168.1.100:9092, 192.168.1.100:9093, 192.168.1.100:9094 --topic many_reps

```
→  ~  kafka-console-producer --broker-list 192.168.1.100:9092,192.168.1.100:9093,192.168.1.100:9094 --topic many_reps
>All three brokers in single run
>This is really cool
>
```

Step 5: Disconnect and connect with single broker

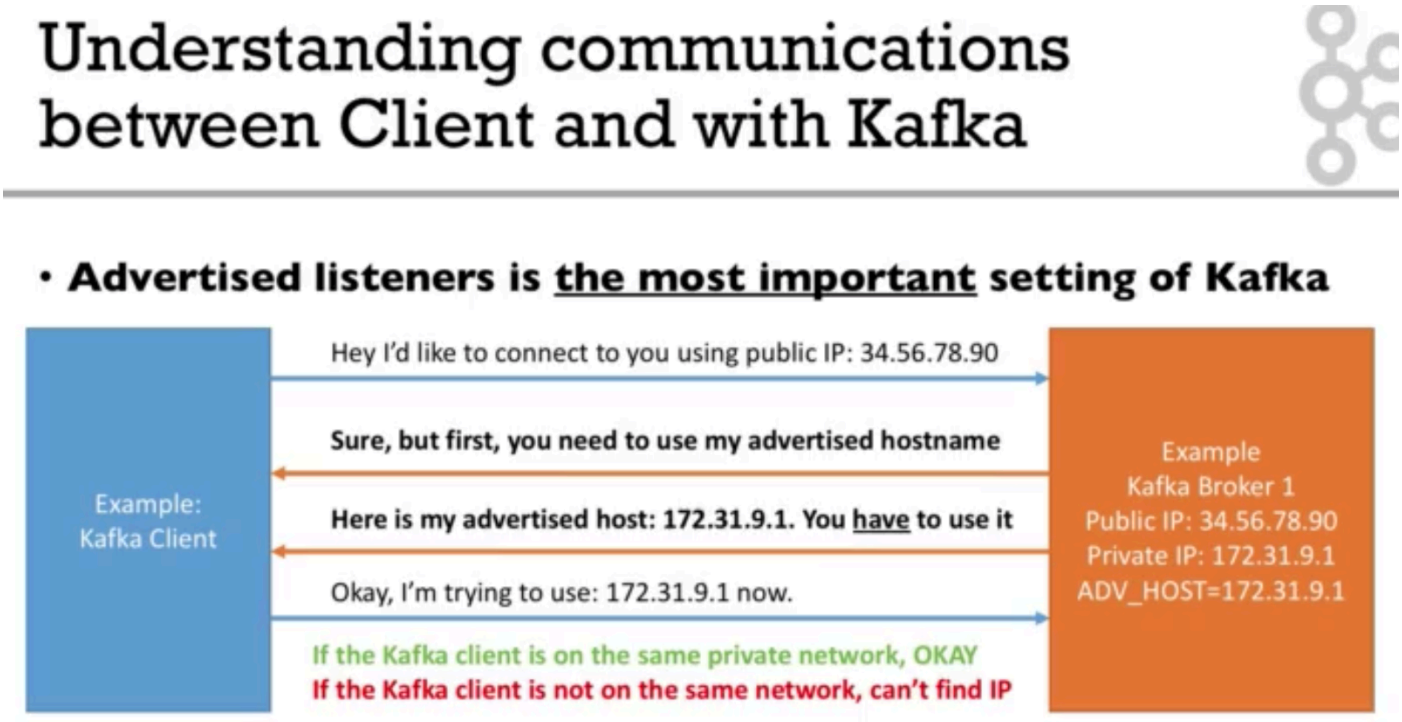
```
→  ~  kafka-console-producer --broker-list 192.168.1.100:9093 --topic many_reps
>This message is from broker which pointing to 9093
>
```

Step 6: Run consumer and check for all the messages

Even though we used single broker for producer, it will connect with all the brokers in cluster. When we use consumer, we could able to see all the messages.

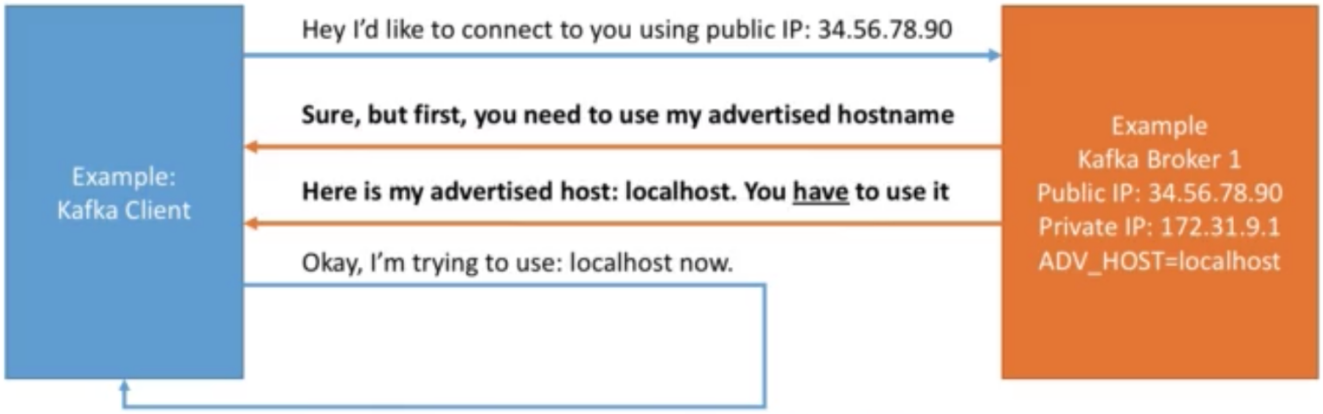
```
→  ~ kafka-console-consumer --bootstrap-server 192.168.1.100:9092 --topic many_reps --from-beginning

This is really cool
All three brokers in single run
This message is from broker which pointing to 9093
```

Kafka Advertised Host Setting



But what if I put localhost? It works on my machine!

- **Advertised hostname is <u>the most important</u> setting of Kafka**



If the Kafka client is on the machine and you have 1 broker, OKAY

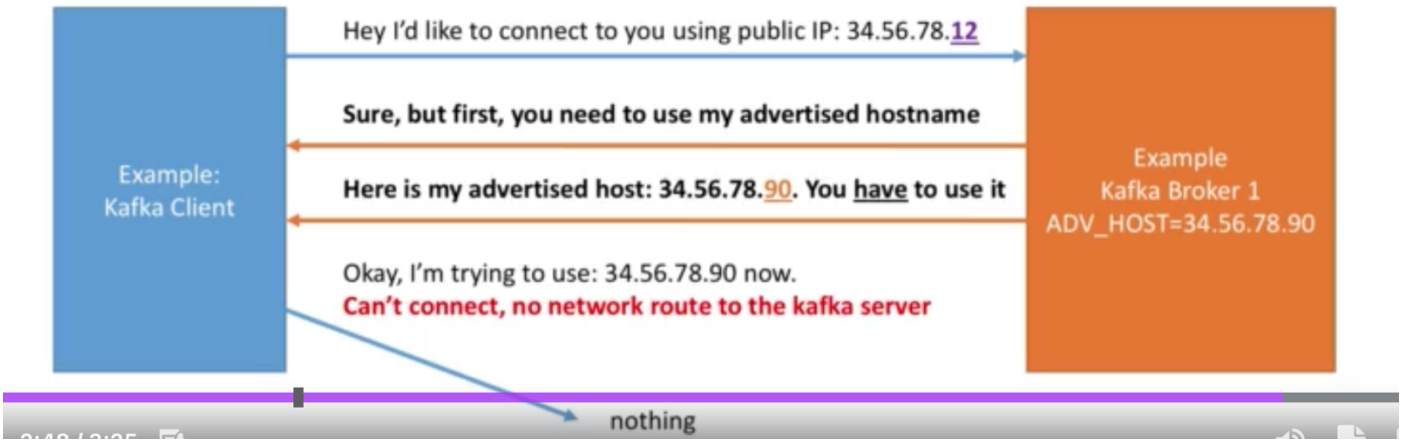
If the Kafka client is not on the same machine or you have 2 or more broker, NOT OKAY

What if I use the public IP?

- **Advertised hostname is <u>the most important</u> setting of Kafka**

What if I use the public IP…
But after a reboot Kafka public IP changed!

- Assume the IP of your server has changed:
  - FROM 34.56.78.90 => TO 34.56.78.12



So what do I set for advertised.listeners?

- If your clients are on your private network, set either:
  - the internal private IP
  - the internal private DNS hostname
- Your clients should be able to resolve the internal IP or hostname

- If your clients are on a public network, set either:
  - The external public IP
  - The external public DNS hostname pointing to the public IP
- Your clients must be able to resolve the public DNS

What's next?

- Go ahead and try to build your data pipelines and applications!
- Find out what you want to learn about next:
- Dev:
  - Kafka Connect
  - Kafka Streams
  - Confluent Schema Registry
- Admin:
  - Kafka Setup
  - Kafka Monitoring
  - Kafka Security