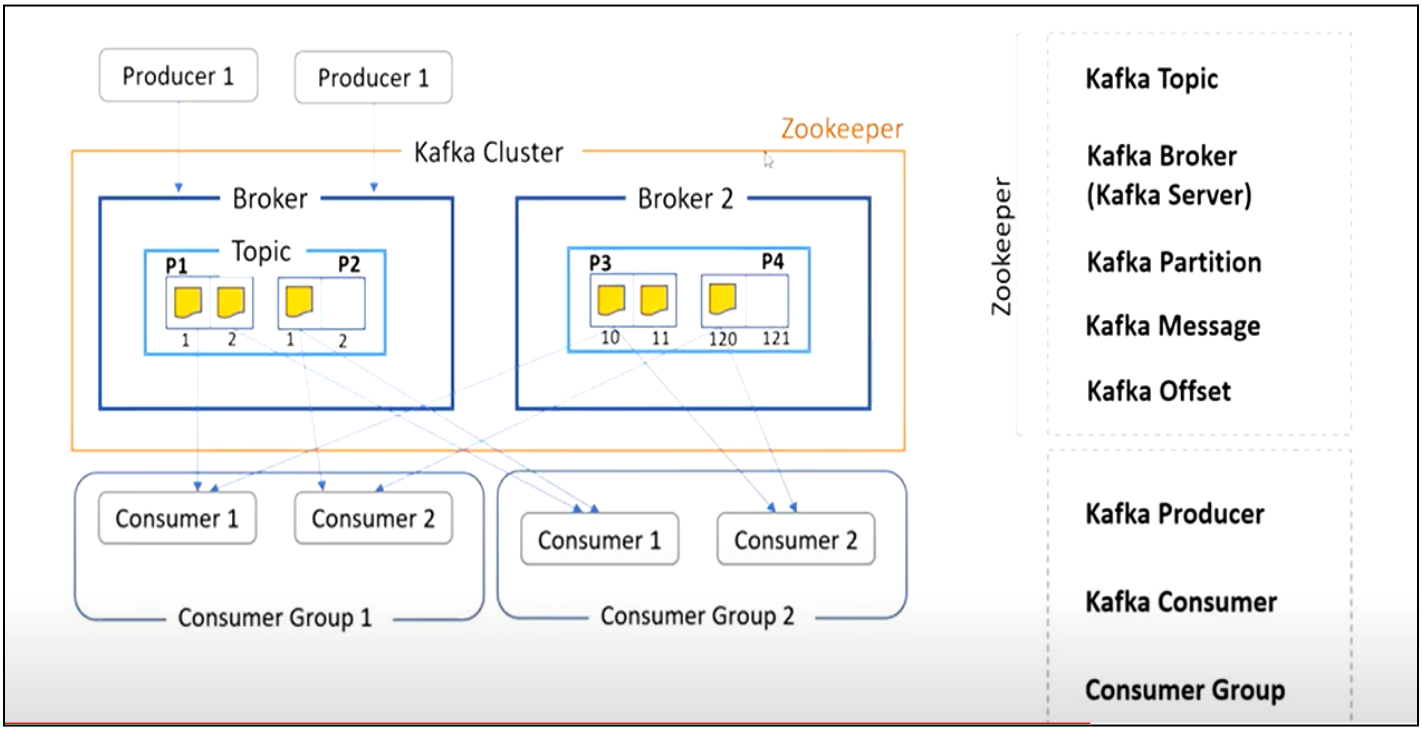
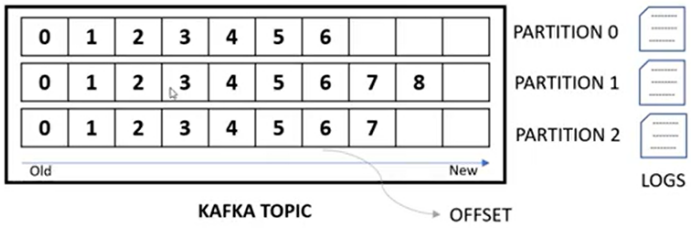
Kafka

Kafka Architecture:



KAFKA TOPICS – PARTITIONS AND OFFSETS

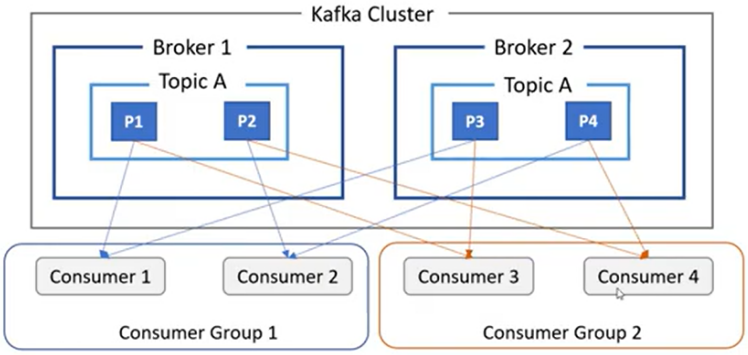


- A Kafka topic is a logical grouping of one or more partitions.
- A partition is the actual unit of storage in Kafka and that is what is stored on the disk.
- Each partition has a separate commit log on the disk.
- A partition contains a sequence of ordered, immutable records, that are continuously appended to.
- Every record in the partition is assigned a sequence number, called the offset, that uniquely identifies each record within each partition.
- Messages can only be appended in a partition, but cannot be modified or deleted.

KAFKA MESSAGES

- **Message Format**
  - Kafka messages are arrays of bytes of variable-size
  - Common formats are String, JSON, and Avro
- **Message Size**
  - Kafka messages do not have an explicit limit on message size
  - Recommended size for high performance is a few KBs
  - Recommended max size is 1 MB
- **Retention Period**
  - Kafka retains messages for a defined period of time
  - Time period can be set on a topic level or on a global level
  - Messages will be retained even after they are read
  - Messages will be discarded automatically after the retention period

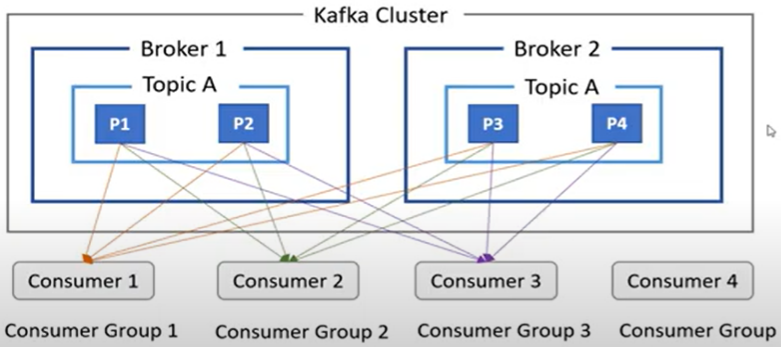
KAFKA CONSUMERS



Kafka provides the guarantee that a topic-partition is assigned to only one consumer within a group.

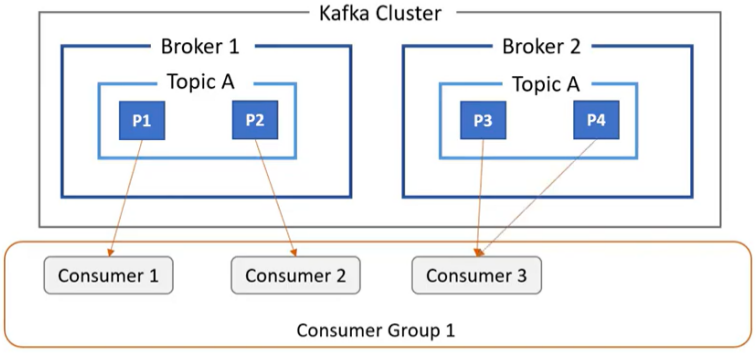
Apache Kafka - What are Consumers, Consumer Groups, and Rebalancing?

KAFKA CONSUMERS



Kafka provides the guarantee that each consumer group-listening on a topic – will receive data from all the partitions of that topic.

KAFKA CONSUMERS



Kafka re-distributes the load among consumers of a consumer group if a consumer is taken out or added – this is called rebalancing.

KAFKA CONSUMERS - SUMMARY

- Kafka consumers are client application or programs – that use Kafka consumer API to consume messages from kafka topics.
- Kafka consumers logically belong to a consumer group. If you do not specify a consumer group for the consumer, then a unique consumer group is assigned to the consumer by the Kafka platform.
- Two consumers from a consumer group cannot listen to the same partition at the same time.
- Kafka Rebalances the consumers when a consumer is either removed from or added to a consumer group.

1. Kafka initial steps

STEP 1: GET KAFKA

Download the latest Kafka release and extract it:

```
$ tar -xzf kafka_2.13-3.1.0.tgz
$ cd kafka_2.13-3.1.0
```

STEP 2: START THE KAFKA ENVIRONMENT

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

STEP 3: CREATE A TOPIC TO STORE YOUR EVENTS

```
$ bin/kafka-topics.sh --bootstrap-server localhost:9092  --create --topic quickstart-events
```

STEP 4: WRITE SOME EVENTS INTO THE TOPIC

```
$ bin/kafka-console-producer.sh --topic  kyc-lead --bootstrap-server localhost:9098
```

STEP 5: READ THE EVENTS

./bin/kafka-console-consumer.sh --topic kyc-lead --from-beginning --bootstrap-server localhost:9098
./bin/kafka-console-consumer.sh --topic lead-response --from-beginning --bootstrap-server localhost:9098
./bin/kafka-console-consumer.sh --topic ft-lead --from-beginning --bootstrap-server localhost:9098

Cmd to run kafka as daemon:

./bin/kafka-server-start.sh -daemon ./config/server-2.properties

2.Topics cmd

1.Creating the Kafka topic first_topic with 3 partitions and a replication factor of 1 when my Kafka broker is running at localhost:9092

kafka-topics.sh --bootstrap-server localhost:9095 --topic test_partition --create --partitions 1 --replication-factor 1

2.Listing topics when my Kafka broker is running at localhost:9092

./bin/kafka-topics.sh --bootstrap-server localhost:9098 --list

3.Describe topic:

./bin/kafka-topics.sh --bootstrap-server localhost:9098 --topic kyc-lead --describe

4.Altering the Kafka topic first_topic to have 5 partitions when my Kafka broker is running at localhost:9092

kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic kyc-lead --partitions 4

5.Deleting the topic first_topic when my Kafka broker is running at localhost:9092

kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic first_topic

3.Consumer cmd

1.show both the key and value and timestamp using the Kafka Console Consumer CLI

kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic kyc-lead --formatter kafka.tools.DefaultMessageFormatter --property print.timestamp=true --property print.key=true --property print.value=true --from-beginning

4.Consumer groups

1.Describe all consumer groups and state :

./bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --all-groups --state

list all Kafka consumers in a consumer groups:
./bin/kafka-consumer-groups.sh --bootstrap-server localhost:9098 --describe --group leads

5.zookeeper

1.check zookeeper status:

bin/zkServer.sh status

2.start zookeeper

./bin/zkServer.sh start

3. This command will give you the list of the active brokers between brackets:

(https://stackoverflow.com/questions/40146921/how-to-list-all-available-kafka-brokers-in-a-cluster)
./bin/zookeeper-shell.sh localhost:2181 or bin/zkCli.sh -server localhost:2181

>> $ ls /brokers/ids # Gives the list of active brokers
>> $ ls /brokers/topics #Gives the list of topics
>> $ get /brokers/ids/1 #Gives more detailed information of the broker id '1'

6.Set up multi-broker Kafka cluster:

Step 1: Setting up a multi-broker cluster:

Duplicate the existing properties file for each new broker:

cp config/server.properties config/server-1.properties
cp config/server.properties config/server-2.properties
Now edit these new files and set the following properties:

Modify new properties file to make the broker unique in the cluster.
//config/server-1.properties:
broker.id=1
listeners=PLAINTEXT://:9093
log.dir=/tmp/kafka-logs-1

//config/server-2.properties:
broker.id=2
listeners=PLAINTEXT://:9094
log.dir=/tmp/kafka-logs-2

//config/server-1.properties:
broker.id=1
listeners=PLAINTEXT://:9093
log.dir=/tmp/kafka-logs-1

//config/server-2.properties:
broker.id=2
listeners=PLAINTEXT://:9094

log.dir=/tmp/kafka-logs-2

7.Replicas

To increase the number of replicas for a given topic you have to:
(https://kafka.apache.org/documentation/#basic_ops_increase_replication_factor)

1. Specify the extra replicas in a custom reassignment json file

For example, you could create increase-replication-factor.json and put this content in it:

```
cat >> increase-replication-factor.json  <<EOF
{"version":1,
  "partitions":[
    {"topic":"kyc-lead","partition":0,"replicas":[0,1,2]},
    {"topic":"kyc-lead","partition":1,"replicas":[0,1,2]},
    {"topic":"kyc-lead","partition":2,"replicas":[0,1,2]},
    {"topic":"kyc-lead","partition":3,"replicas":[0,1,2]}
]}
EOF
```

//////////////////////////Note :"replicas":[0,1,2] ********0,1,2 are broker ids

2. Use the file with the --execute option of the kafka-reassign-partitions tool

[or kafka-reassign-partitions.sh - depending on the kafka package]

For example:

```
$ bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 --reassignment-json-file
        increase-replication-factor.json --execute
```

3. Verify the replication factor with the kafka-topics tool

[or kafka-topics.sh - depending on the kafka package]

```
 $ ./bin/kafka-topics.sh --bootstrap-server localhost:9092  --topic kyc-lead --describe
```

8. Safe kafka producer config:

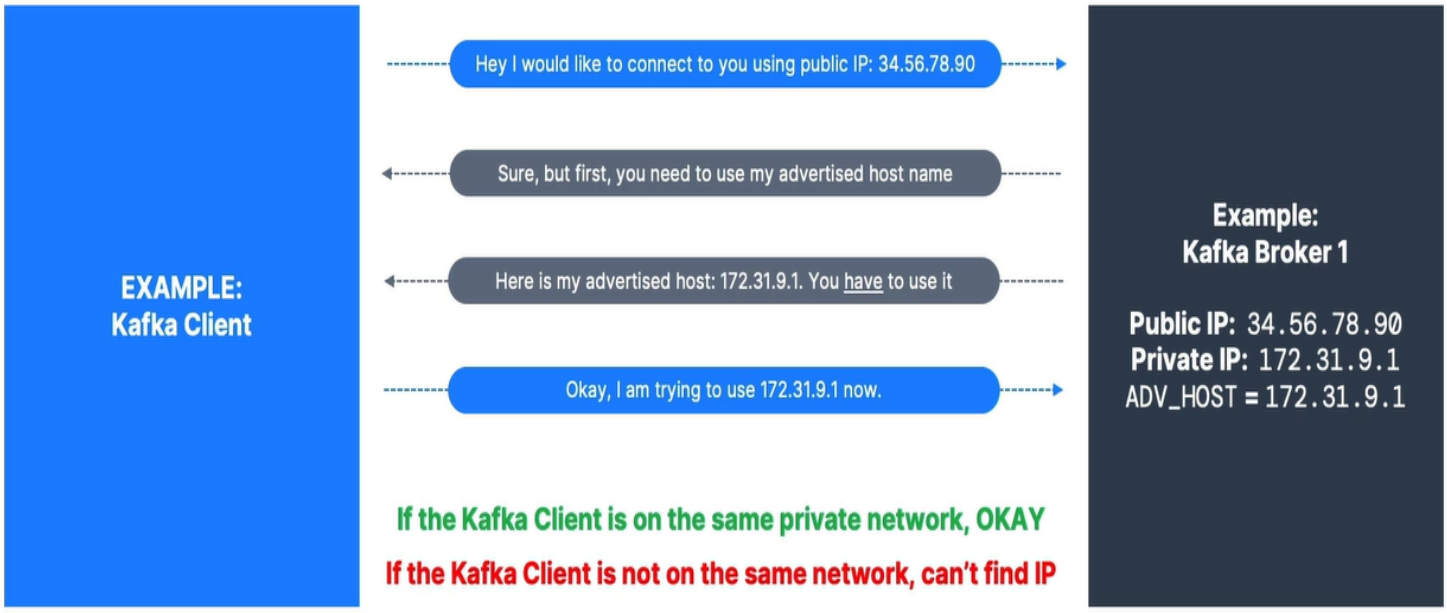
Imp note:

- **batch.size**: The maximum amount of data that can be sent in a single request. If batch.size is (32*1024) that means 32 KB can be sent out in a single request.
- **buffer.memory**: if Kafka Producer is not able to send messages(batches) to Kafka broker (Say broker is down). It starts accumulating the message batches in the buffer memory (default 32 MB). Once the buffer is full, It will wait for "**max.block.ms**" (default 60,000ms) so that buffer can be cleared out. Then it's throw exception.
- **linger.ms** refers to the time to wait before sending messages out to Kafka. It defaults to 0, which the system interprets as ‘send messages as soon as they are ready to be sent’.
- **batch.size** refers to the maximum amount of data to be collected before sending the batch.
- ***Kafka producers will send out the next batch of messages whenever linger.ms or batch.size is met first.***

Advertising listener :

- 1st client connect to broker via public ip
- The broker shares the adv host ip to client
- Then rest of the communication b/w client and host takes via adv host

- Advertised listeners is the most important setting of Kafka



- ***If you are setting public ip as adv host the we need to change adv host as and when public ip changes***
- ***If we are using local host the both client and broker must be on same network***

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 hostname pointing to the public IP
- Your clients must be able to resolve the public hostname

Connector
Streams
KSQL

Refer:

- https://www.conduktor.io/kafka
- https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#what-is-kafka
- https://docs.confluent.io/platform/current/kafka/monitoring.html
- https://docs.confluent.io/platform/current/multi-dc-deployments/replicator/replicator-tuning.html#improving-network-utilization-of-a-connect-task