kafka session 1
==========

Socket approach..

create real time streams

process streaming data

Apache Kafka is highly scalable and distributed platform for creating and processing streams in real-time.

Pub-Sub messaging system

producer -> message broker -> consumer

Kafka as a Data integration platform

1.server software - broker

2.client API - Java Library

Producer API

Consumer API

==========

Producers and consumers are decoupled.

there can be one producer and multiple consumers.

==========

producer
consumer
broker
cluster
topic
partitions
partition offset
consumer groups

kafka producer

sends data / message / record..

for kafka this is simple array of bytes..

==========

consumer are the recepeints..

application which read data from kakfa server is called consumer.

Twitter -> spark structured streaming

==========

broker

it is nothing but kafka server..

producer and consumer do not interact directly.

==========

cluster

group of computers each run one instance of kafka broker.

==========

topic

its a unique name for a data stream

is like a table

==========

partitions (is the smallest unit sitting on single machine)

number of partitions in a topic is design decision.

==========

partition offset.

unique sequence id of a message in partition.

this sequence id is assigned to every message by broker. these are immutable. (based on arrival)

to locate a message we need

topic name -> partition number -> offset

==========

consumer groups

retail example (scalability)

500 paritions - 100 consumers - 500 consumers

Kafka session - 3
==========

B1 - p1, p2, p3

B2 - p2, p1, p3

B3 - p3, p1, p2

Topic1 - p1 , p2 , p3

we can set the replication factor - 3

Scalability - partitions

fault tolerance - replication factor

==========

3 ways...

1. open source - Apache Kafka

2. commercial distribution - confluent.io

3. fully managed service - confluent, amazon aws (kinesis)

zookeeper already installed..

helps to cordinate the various distributed services..

many brokers...

3 brokers..

kafka brokers will store shared information in zookeeper.

This is used to cordinate various brokers...

Kakfa community is planning to get rid of zookeeper in coming versions..

have a one node kafka cluster..

zookeeper should be already running...

java version 1.7

we want to start a single broker.

2 pre-requisites
==========

1. java should be installed
2. zookeeper should be running (this is done)

https://drive.google.com/file/d/18t2TBE4aHs_AEAhucgvrmLSWY__FpbOW/view?usp=sharing

export JAVA_HOME=/home/cloudera/Downloads/jdk1.8.0_271
export PATH=$JAVA_HOME/bin:$PATH

in kafka folder

inside bin we have the runnable scripts..

config folder has configuration files which needs to be passed to the runnable scripts...

Kafka session - 4
==========

how to create a topic

./kafka-topics.sh --create --topic mynewtopic2 --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

how to list down the topics

./kafka-topics.sh --list --bootstrap-server localhost:9092

command line producer for producing the data

./kafka-console-producer.sh --broker-list localhost:9092 --topic mynewtopic

command line consumer to consume

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic mynewtopic --from-beginning

Kafka session - 5
==========

how to create a multi node kafka cluster.

start 3 brokers...

./kakfa-server-start.sh server.properies

ideally kafka broker run on port 9092

each broker should have a different IP

=========

broker.id
listeners=PLAINTEXT://:9092

3 brokers

machine1 - broker1
broker.id = 0
ip1:9092
/tmp/kafka-logs

machine2 - broker2
broker.id = 1
ip2:9092
/tmp/kafka-logs

machine3 - broker3
broker.id = 2
ip3:9092
/tmp/kafka-logs

3 brokers

server.properties
machine1 - broker1
broker.id = 0
ip1:9092
/tmp/kafka-logs

server1.properties
machine1 - broker2
broker.id = 1
ip1:9093
/tmp/kafka-logs1

server2.properties
machine1 - broker3
broker.id = 2
ip1:9094
/tmp/kafka-logs2

I have started 3 brokers...

but on same machine...

I want to create a new topic with 3 partitions...

./kafka-topics.sh --create --topic topic1 --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

command line producer for producing the data

./kafka-console-producer.sh --broker-list localhost:9092 --topic topic1

command line consumer to consume

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topic1 --from-beginning

topic1 with 3 partitions...

we have 3 brokers

topic1-0
topic1-1
topic1-2

broker 1 - partition 3
broker 2 - partition 2
broker 3 - partition 1

how to create a 3 node kafka cluster..

how to create a topic with 3 partitions...

we then created a producer and produced the data

we then started a consumer to consume the data...

we saw the log dir

/tmp/kafka-logs

.log (kafka-dump-log.sh)

/home/cloudera/Downloads/kafka_2.12-2.6.0/bin/kafka-dump-log.sh --files *.log

==========

Kafka session - 6
==========

3 brokers...

we created a topic with 3 partitions...

single consumer...

consumer group
==========

a group of consumers to make it a more scalable process..

we already have 3 brokers running...

we will create a new topic - customers

3 partitions...

replication 1...

./kafka-topics.sh --create --topic customers --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

12434 are number of records in the file..

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic customers --from-beginning --group customer-group

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic customers --from-beginning --group customer-group

./kafka-console-producer.sh --broker-list localhost:9092 --topic customers < ../../../Desktop/customers.csv

Processed a total of 8432 messages - consumer1 - customers-0 & customers1 (partition 1 & 2)
Processed a total of 4002 messages - consumer2 - customers-2 (partition3)

/tmp

broker0 | kafka-logs customers-2 - 4002

broker1 | kafka-logs1 customers-1 - 4004

broker2 | kafka-logs2 customers-0 - 4428

customers-0

customers-1

customers-2

/home/cloudera/Downloads/kafka

3 partitions and you have 5 consumers...

Kafka session - 7
==========

3 partitions and replication factor of 3

we will try to understand the storage architecture..

./kafka-topics.sh --create --topic customers-new --bootstrap-server localhost:9092 --partitions 3 --replication-factor 3

customers-new-0
customers-new-1
customers-new-2

1 GB or 1 week of data (whichever is smaller)

Topic -> Log files -> partitions -> replicated -> segments...

3 replicas..
==========

1. leader partition customers-new-0 broker-0
2. follower partition customers-new-0 broker-1, customers-new-0 broker-2

./kafka-topics.sh --describe --topic customers-new --bootstrap-server localhost:9092,localhost:9093

segment offset

if we want to uniquely identify a message...

we would need 3 things..

1. topic name

2. partition number

3. offset number

offset is not unique for a topic..

but its unique for a partition...

00000000000034578.log

Kafka session - 8
==========

what is a kafka cluster...

a group of brokers...

is a kafka cluster a master-slave architecture?

kafka cluster follows a masterless architecture.

if a kafka cluster do not have a master..

then who takes care of maintaining list of active brokers..

and monitoring the brokers..

zookeeper - maintains the list of active brokers..

b0,b1,b2

zookeeper acts like a database..

whenever a broker starts, it creates an ephemeral node in zookeeper.

/brokers/ids

Downloads/Kafka/bin

zookeeper-shell.sh localhost:2181

I stopped the broker 0

then we queried again to see the list of active nodes in zookeeper.

ls /brokers/ids

delete the directory..

rm -rf /tmp/kafka-logs

unblock the port 9092

sudo fuser -k 9092/tcp

monitoring of active brokers.. reassigning work when active broker leaves the cluster..

controller
==========

one of the broker from kafka cluster is elected as the controller.

in a kafka cluster there can be just one controller.

if we have single node kafka cluster, it will server both as a broker and a controller.

a controller monitors the list of active brokers..

whenever the controller notices that a broker left the cluster then it will reassign the work to other brokers.

which broker is elected as the controller?

the first broker that starts in the cluster becomes the controller.

how zookeeper tracks it...

whenever we start the first broker in the cluster..

it will create an ephemeral node

/controller

the first broker that starts in cluster becomes the controller.

when other brokers start they will also try to create the /controller ephemeral node. But they will get an exception.

both the other brokers 1,2 will be trying continuously to create the emphemeral node /controller.

other brokers will keep on watching the /controller ephemeral node.

and when the controller is dead then all the other brokers will try to create the ephemeral node but only one broker will be able to do that.

Kafka session - 9
==========

how the partitions of topics... are distributed across various brokers..

also we will see how to handle fault tolerance..

we have a cluster with 6 brokers..

2 racks..

consider we have a topic with 10 partitions..

lets say we have a replication factor of 3...

10 * 3 = 30 replicas..

how are these 30 partitions assigned to 6 brokers.

1. even distribution (achieving workload balance)

2. fault tolerance (replicas should be stored on different machines)

to distribute the partitions across brokers..

P0,P1,P2,P3,P4,P5,P6,P7,P8,P9

followers - P0,P1,P2,P3,P4,P5,P6,P7,P8,P9

followers - P0,P1,P2,P3,P4,P5,P6,P7,P8,P9

1. it creates an ordered list of available brokers..

R1-B0 P0 P6 P5 P4
R2-B3 P1 P7 P0 P6 P5
R1-B1 P2 P8 P1 P7 P0 P6
R2-B4 P3 P9 P2 P8 P1 P7
R1-B2 P4 P3 P9 P2 P8
R2-B5 P5 P4 P3 P9

Leader partition
follower partitions.

A broker can perform 2 kinds of activities..
1. leader activity - leader partitions
2. follower activity - follower partitions

A leader is responsible for all the requests from producers and consumers..

if a producer wants to produce message to kafka topic..

the producer will request one of the broker in the kafka cluster and get the topic metadata.

This metadata will contain a list of all leader partitions of that topic..

p0 - ip1:9092
p1 - ip2: 9093
p2 - ip3:9094

so producer will directly send message to leader partitions..

once message is produced.. the leader broker will send an acknowledgment.

consumer also reads from leaders partitions..

so the producer and consumer always interact with the leader.

Kafka session - 10
==========

Followers do not serve the producer and consumer request.

Followers copy the message from the leader and try to stay upto date with the messages..

a follower always wants to be in sync and upto date.

the aim of a follower is to be elected as a leader when current leader fails or dies...

so followers always wants to be in sync upto date with the leader. if they lag behind then they cant be elected as the leader.

Followers request for messages from leader and leadder sends the messages. this is a continous cycle.

Followers might lag behind because of many factors..

1. Network congestion..

2. broker failure.. (follower broker crash/restart)

ISR (in sync replicas)

Zookeeper maintains list of ISR. This list is maintained by leader broker..

if the follower is not too far.. the leader will add the follower to the ISR list..

how do we define he not too far.. ?

default value is 10 seconds..

you have 3 replicas..

1 leader and 2 follower..

both the followers 12 seconds behind..

ISR list will be empty in this case..

now if leader goes down who will be the next leader?

if we select any of the follower we miss the latesgt 12 seconds message..

1. committed vs un-committed messages

2. minimum in sync replicas

if the message is successfuly copied across all followers than the message will be marked as committed.

leader partition - committed , un-committed

if the leader fails, the committed messages are already taken care.

uncommitted messages will be lost..

uncommitted messages can be resent by the producer.

producers can choose to receive an acknowledgement of sent messages only after message is fully committed..

producer waits for acknowledgment.

if producer do not get an acknowledgement till the timeout interval then it will resend the message.

we have 3 partitions..

and 2 follower partitions go down..

single leader partition has the data and is in sync..

kafka protects this by setting minimum number of in-sync replicas of a topic..

min.insync.replicas=2

if we do not have atleast 2 replicas insync.. the broker will not accept any message..

it will say not enough replicas exception..

in such case the leader becomes a read only partition..
Class comments

Kafka session - 11
==========

it is a 4 step process

Step 1. setting some properties.

step 2. create object of kafka producer.

step 3. calling the send method. this takes a producer record as argument

step 4. closing the producer object.

Kafka session - 12
==========

Producer record object takes

2 mandatory arguments
==========

Topic name
Message value

Optional arguments
==========

Message key - is quite important (groupings, join)

Target partition

Message timestamp

once the producer record is created it's handed over to the producer using the send method.

Kafka producer does not transmit the records immediately to the kafka cluster.

Every record goes through serialization, partitioning and then kept in the buffer.

Serialization
==========
kafka does not know how to serialize the key and value.

for this reason specifiying a key serializer and value serializer is mandatory.

Paritioner
==========

Topic1 - 4 partitions p0, p1, p2, p3

1. Default mechanism to decide the partition..

hash key partitioner - 17%4 = 3
round robin paritioner - whenever key is null (whenever there is no key)

2. Custom partitioner class

we can write our own custom partitioning logic.

Kafka session 13
==========

Message timestamp
==========

every message by default has a timestamp even if we do not specify that.

2 types of timestamp
==========

1. create time (time at which message was produced) 10 am
2. log append time (time when broker receives the message) 10:01 am

default is create time.

message.timestamp.type = 0 (create time)

message.timestamp.type = 1 (log append time)

by default the value is 0.

serialization + parititioning

after this the message is put in the buffer.

producer consists of partitionwise buffer space.

By default the size of buffer is 32 mb.

Buffering helps you to give network optimization.

if the buffer is full, then the send method has to wait and after a certain time it might give an exception.

buffer.memory

Kafka session 14
==========

At least once vs at most once vs exactly once
==========

IO Thread took the messages from buffer and sent it to the kafka cluster(broker)

The message is stored successfully in the cluster.

But The IO thread didnt receive an acknowledment..

IO thread will again try to send the message..

if using the producer configuration we set the number of retries= 0 then we get at most once semantics..

the default behavious is at least once.

sometimes there iss a need to have exactly once..

enable.idempotence = true

1. each producer will contact the leader broker.. it will request for a unique producer id.

Broker will dynamically assign a unique id to each producer.

2. Message sequence number..starts from 0 and increase.. per partition..

each message will be uniquely identified by producer id and sequence number.

this will help the broker to find out duplicates and missing records..

Note: it only guarantees for producer retries.. its not valid for duplicates sent by application itself.