# **Apache Kafka**

Reference links

<https://www.conduktor.io/kafka/kafka-topics-internals-segments-and-indexes/>

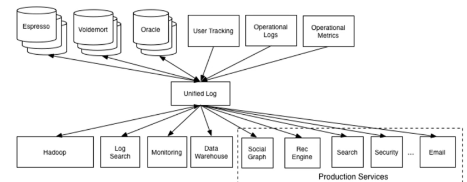
|  |  |
| --- | --- |
|  |  |
| To explore options | <https://www.conduktor.io/kafka/kafka-options-explorer/> |
|  |  |
|  |  |

Kafka was originated at linked in, to solve the data integration problem

Kafka was developed by Jay Kreps, Neha Narkhede, and Jun Rao

They built Kafka as realtime streaming platform

Only difference is hadoop enables only batch processing of data , whereas kafka enables real time processing of data



## What is apache Kafka?

Apache kafka is

* horizontally scalable (means we don’t need a single super kafka broker which have hundreds of gb’s of ram- like instead of having 1 super/master cook - we have 10 cooks in kitchen to share the load so here kafka is horizontally scalable which supports cluster of kafka brokers)
* fault tolerant – means -even if 1 broker in a cluster goes down as we have topic replication (our data is copied/distributed to many other brokers and kafka is durable) we can connect to other replication topic in another broker
* distributed event streaming platform (event indicate something has happened- like adding item to cart, deleting an ele from a cart) event =notification +state (immutable pieces of information because these events are already happened )
* in kafka all events are written in order inside a partition
* u can replicate data like anything
* kafka is Distributed Streaming platform (distributed commit log ), its more than MQ (Here distributed means data is copied to other brokers)
* in kafka data is stored in order (Because once we open a partition file we will continuously append the data to that file)

it can process 1 trillion messages per day

ex:- phone pe – 41 clusters, 450 brokers, 1 million messages per second

Use case:

- in phone pe they are using kafka for inter service communication , async processing like in 1DSTR application , they used kafka for async processing

Apache Kafka is highly scalable (each broker can be replicated easily like replication factor of 3.. and once ur connected to any broker in a cluster means you are connected to the entire cluster) and it’s a distributed platform for creating and processing streams

Its Iike a Blood circulatory systems which will pump blood to all organs similarly it will push data to all listeners. Messages can be sent from producers to consumers in milli seconds

Example of processing streams

* To know the lorries real time positions the machines kept in lorries will continuously send their latitude and longitude data to the kafka broker, this is called stream of data because these lorries will send continuous data/ data stream to the broker

Stream is nothing but continuous flow of data

How it evolved as streaming platform from data integration platform

When to use kafka

Assume if a system is providing some data, if many other systems wants same data then use Kafka

Let say we are sending some amazon order information to one topic, if other systems also wants same data like delivery service, Item less price detection service, notification service.. if all these applications also wants the same data then use Kafka

Components of Kafka

|  |
| --- |
| 1. Kafka Broker - |
| 1. Kafka Client |
| 1. Kafka connect – for integration between 2 components / multiple components solving above problem- like 1dstr appn is used for integration as we don’t get stream of data()because its not continuous flow of data |
| 1. Kafka Streams |
| 1. Kafka SQL/ Ksql – with this kafka wanted to become real time database. – all above 4 are open sourced whereas this is licensed |
|  |

|  |
| --- |
| PF ~~ 1 partition = 1 folder  5 partitions= kafka will create 5 folders ,1 folder means 1 partition |
| 1 folder will have many files ~~ 1 file == 1 segment  1 partition folder will have many segments (segment is a file for kafka)= 1 folder will have many files  Only one segment is ACTIVE at any point in time |

Analogies

In kafka we have lot of topics (in city we have lot of schools-Narayana, Sri chaitanya)

Same like movie theater

|  |  |  |
| --- | --- | --- |
| Topic | 1 partition = 1 folder | 1 folder will have many segment files and the offset number will be continued across all the file segments |
| Movie- topic | It have many partitions  Classes- gold, economy  Inside a partition offset number will continue  Segment =1 row | Every person will have 1 seat number and  every message will have 1 offset number  gold partition will have some offset numbers ranging 1-300, same like economy partitions will also have same offset numbers ranging from 1-300 |
| Hostel – topic | It have many partitions (many blocks/ building) |  |

Consider 1 campus is 1 topic

* Pf- 1 partition means 1 folders
* In real time 5 partitions means for kafka 5 folders -1 folder per partition

Simple example:- we **cant store huge data in single file-so** we will **store in multiple files(called segments), and we cant keep thousands of files in same folder if we keep then**

**That single folder will become big** **and u can’t open that folder at all**, so we will **keep files in multiple folders (means here folder is called partition)**

Let’s take 1 topic =many partitions like each hostel ex:- Nellore pidathapolur topic will have many partitions means it will have many blocks-like Krishna, cauvery

* 1 partition will have many segments like 1 block will have many rooms -(in 1t partition rooms ranging 1-40) these same room numbers 1-40 will be present in another partition also . so these 1-40 offset numbers are unique only inside a partition
* **Storing across partition** All the incoming messages will be spread across diff partitions, like many kinds of people will come to that hostel, broker /principal will store them separately l
* like all men juniors in -A block, men senior’s in B block, girls junior’s in C block, Girls seniors in D block, Staff in another block
* similarly all the incoming messages will not be stored in a single partition/single folder, it will be spread across multiple folders
* **storing inside a partition in all segments:-** inside a block once a room (segment/file)is opened warden will fill that entire room, if max capacity of boys/girls are reached then only they will start opening/ filling another room, in real time inside a partition (in kafka terminology it’s a folder)inside a folder we have many files each file is nothing but a segment, once a file(room/segment ) is opened until max file size is reached all the messages will be continuously written to this file so inside a file all messages will be written sequentially here. Until the max capacity of that segment is reached data will be written to that segment only. if that room is filled it will start filling another room if that segment if filled it will start filling other segment
* Still there is an option ,u can specify the partition number to store that message, if u specify 1 as partition num, that message will go to that partition only
* Each message sits in a partition will have offset id, like

If u want to locate a message u must know below

1. U should know the topic name (hostel name)/folder name
2. Partition number (like block name inside a hostel)
3. Offset number

Kafka terminologies

Kafka is capable to receive 1 Million messages per second, We can’t write all the messages to single file ( file == segment), so we write in multiple files

And we cant keep crores of files in single folder (if folder became big we can never open) that’s why we segregate the files into folder (folder == partition )

|  |
| --- |
| PF = partition == folder  1 application.exe will have many folders  1 topic will have many partitions  1 partition means ==1 folder |
| 1 folder will have many files /text files  1 partition will have many segments  Segment means=== File /Text file  Once a segment file is opened, then data will be continuously written to it until max file size is reached  each message we it will be uniquely identified by the offset number |
| Data is distributed means == data is replicated to other brokers to avoid failures |
| A message is nothing but an array of bytes  Every message will have a key, messages with same key will go to same partition  Here key decides the partition , Hash value of the key will be calculated which derives the partition/folder number |
|  |

Ideas

If u don’t have access to source code use kafka connect api(source connector for producer and sink connector for consumer api)

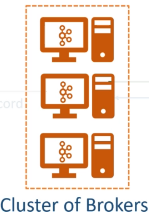
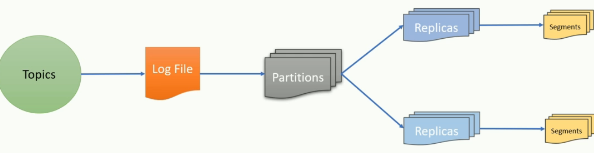
Broker

Broker is nothing but a kafka server

A kafka cluster is nothing but collection of brokers

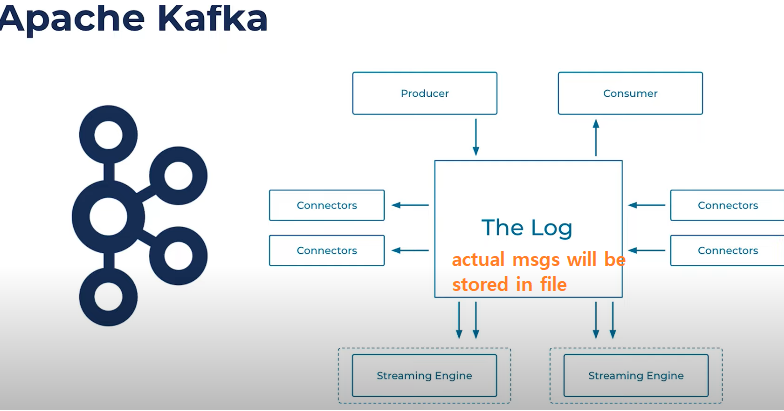
**Broker responsibilities**

1. Receive messages from the producers and acknowledge them
2. Broker responsibility is just to write the messages to files, if current broker goes down a new broker will come and write to the files, so all previous data will be saved
3. Store the messages in kafka log file so that even if one of the consumer is offline broker should be able to send him the message once consumer is back online, it internally uses 5 partitions means 5 folders inside folder we have log files which are called segment log file, so all messages will be written here
4. Deliver the message to the consumer when they requested it



TOPIC

If we send a message to kafka broker then internally the data will be stored in files once written the data is immutable



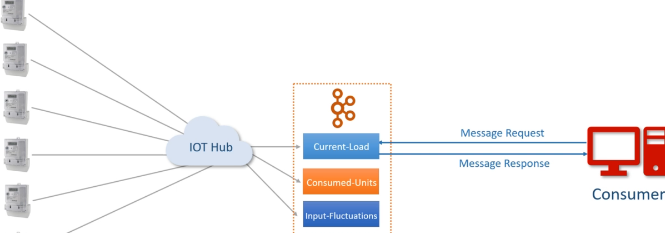
Its like a small data base table like all employee information in employee table- in database we will create table , in kafka we will create a topic

Like all sakshi related news in sakshi related topic , and etv news in etv topic

It is the unique name given for the data stream,

Its like unique given for your data

Topic can store millions of records, ofcourse kafka can receive million messages per second

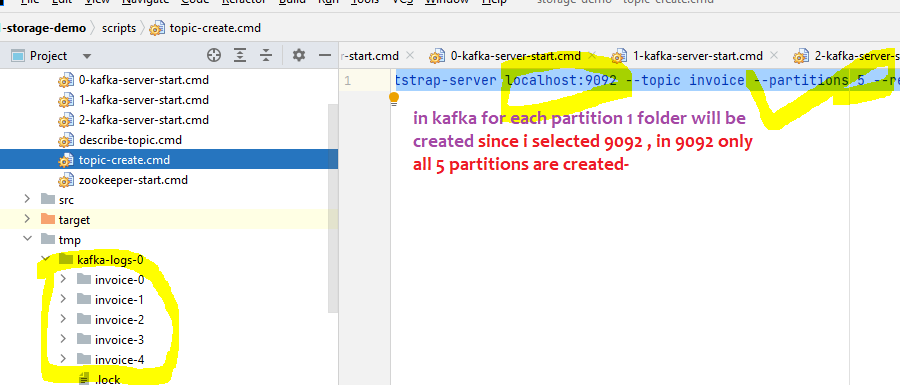
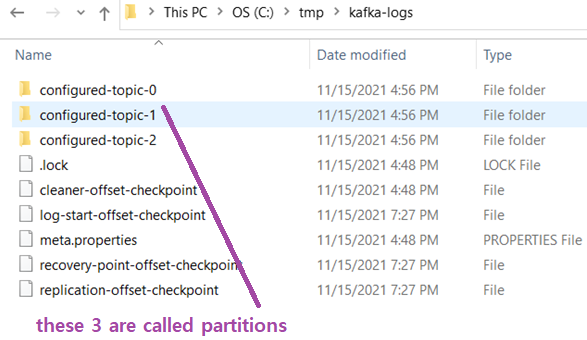


Partitions [pf= partition=folder]

If u create 5 partitions ~~internally kafka will create 5 folders as below

And always remember , if u don’t send partition number , then if u send the message the message will be send to all partitions, lets say if u are sending 5000 messages, some msg will go to partition-1, some may goto partition-2, some may goto partition-3 ..

But if u want all ur messages to go to partition – 1 u can do that while sending the messages send partition number too

In above u saw 5 partitions are there in broker-1, if u create with replication factor 3 each partition will be created 3 times means invoice-0 will be present 3 times totally- this will be in broker 0, broker -1 too

Note:- all the folder partitions should not be in same broker, if all folders are in same broker, incase if the main broker went down then all data will be gone right, so all folder partitions

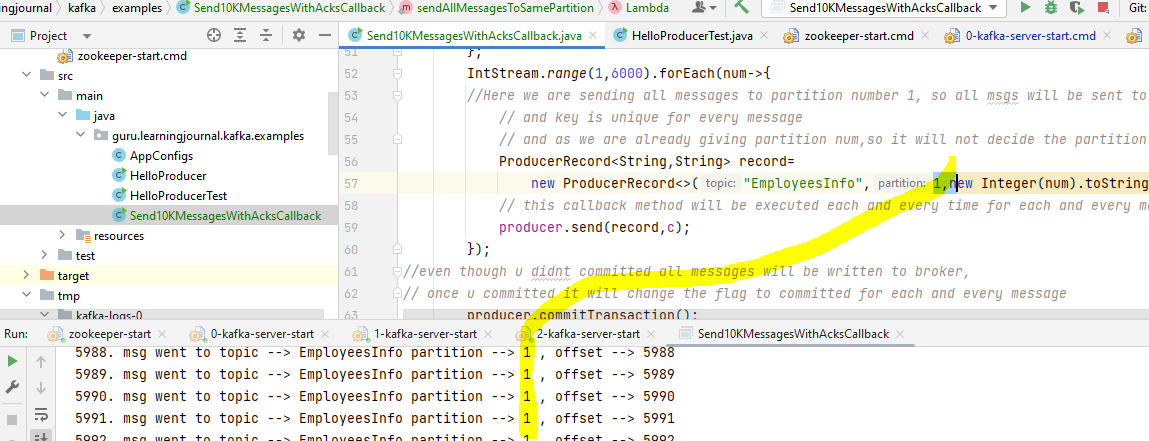
Should not be in same broker, 1st folder partition will be broker-1 and 2nd partition folder may be in broker-2 , 3rd partition will be in broker-3, and because of replication factor=2

Again broker-1 will have 3rd partition, broker-2 will have 1st partition, broker-3 will have 2nd partition

Therefore partion-2 will be present in broker-1, broker-2 and since 1 must be leader, for partition-2 broker 1 must be leader…

*//Here we are sending all messages to partition number 1, so all msgs will be sent to p1  
 // and key is unique for every message  
 // and as we are already giving partition num, so it will not decide the partition based on the key, so as we gave partition number it will goto p1 only*

ProducerRecord<String,String> record=  
 new ProducerRecord<>("EmployeesInfo",1,new Integer(num).toString(),"Msg--"+num);  
*// this callback method will be executed each and every time for each and every message*producer.send(record,c);



#### Segment means = file

Topic is divided into many partitions (folders) and partition (folder) is divided into many segment (files)

**In file segments only we will store data, at any time only 1 segment file will be open and data will be written to that file segment, until max size of that segment is reached**

If Each segment size is 1MB, until that segment is filled it won’t navigate to other segment

**log.segment.bytes:**the max size of a single segment in bytes (default 1 GB)

 By default, each segment contains either 1 GB of data or a week of data, whichever limit is attained first, if 1GB data came in 1 day then that **segment will be closed in 1 day and then it will be set for expiration**

**log.segment.ms** Another way to control when segments are closed is by using the log.segment.ms parameter, which specifies the amount of time after which a segment should be closed. The default is 1 week. Kafka will close a segment **either when the size limit is reached or when the time limit is reached, whichever comes first.**

**Once segment is closed and expiration timer will start and broker will wait for retention period, & once retention period is over then segment will be deleted**

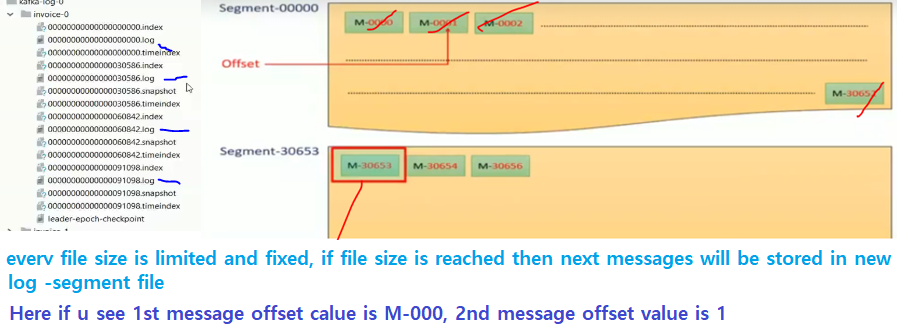
Analogy: in narayana hostel each block is a partition , each floor is a segment

Why multiple segments/files? Because assume a guy who created with 1 partition, if he is sending crores of messages if there is 1 file, then that file size will grow till be 10Gb, to avoid that, and to have every file is limited to some size, after that a new file is created to store all messages ,

If more messages comes then new messages will be stored in new file

As we have 3 partitions , in real time we can have 3 producers 1st producer can send data to 1st partition alone and 2nd producer can send to 2nd partition and we can have a key which decides the partition number and 1 partition can have many segment

Lets say if segment size is 1MB & if we have 3 segments for each partition 🡪 So when 1st partition receives the 1st message it will go to 1st segment only until that 1st segment fulls messages will continuously goes there only🡪 the segment file grows until the maximum segment file reached like once text file is opened it will continuously write until that File max size is reached



All the .log file contains our messages in encrypted format

Until max file size is reached we will keep on writing to that file segment, once max file size reached, we will create new segment file

if segment limit is reached then kafka will start writing to another segment, if we configure segment size is 1MB – until 1MB is filled it will writes there only, all the messages will be written here sequentially

If u observe the image all the messages are written synchronously inside a segment

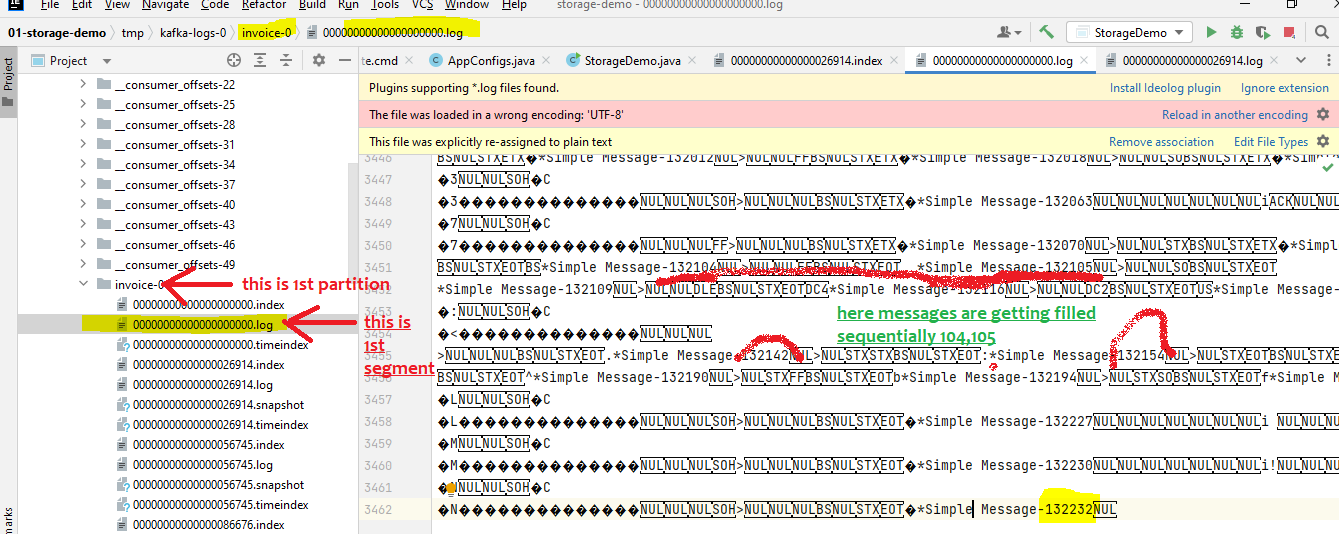
Note:- offset number will be continued to other segment, in first segment we have messages ranging from 0-30652 and in second segment the offset number will continue from 30653-62000 and from 3rd segment offset number start with 61,000

* Every message is identified by an offset number- when messages are written it will be written with address –like every object in java have an address, every message will have some address , and offset number will be unique in the entire folder (means across all segments as offset number continues to other segments also)

In that partition



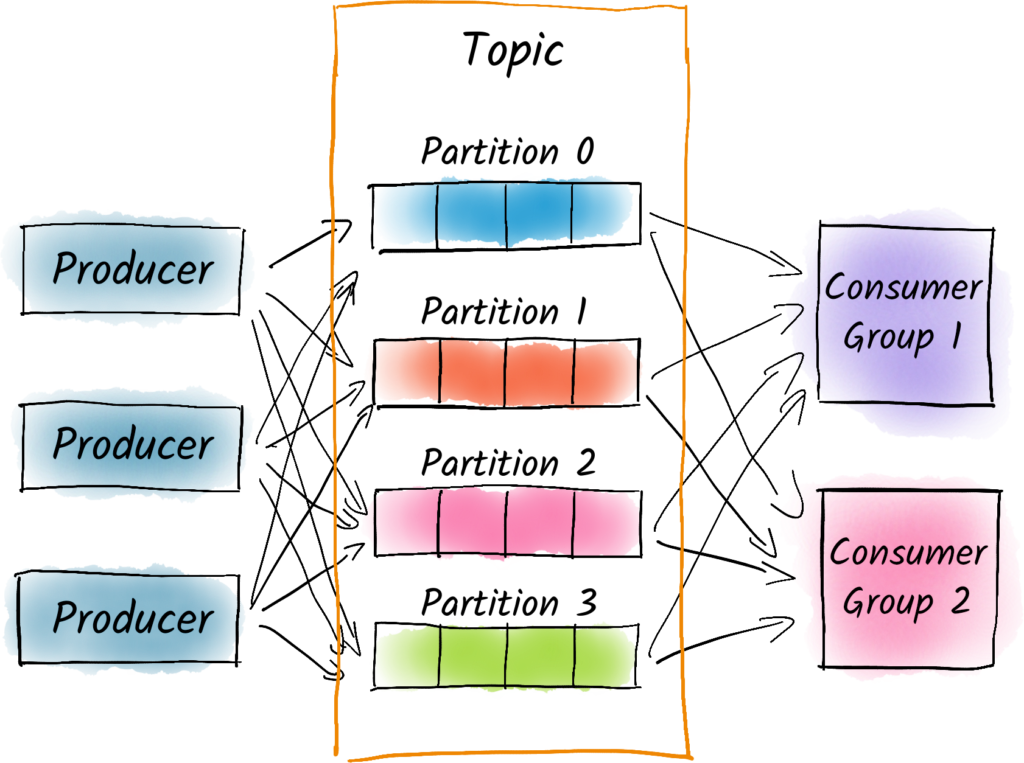
* Inside a partition folder pf, offset is unique across all segments files
* And for easy identification purpose, starting offset is stored as the file name
* Offset number is unique only inside a partition and if u go to another partition same offset number could be there



In the above understand that, inside a partition all will be written synchronously, 104,105 both are in same segment, notice that **132190 is here but 91,92 are not here they are in second partition of same topic**, only inside a partition they will be written sequentially

If huge 30 lakhs data is coming, to handle it, in 1 topic they created 3 partitions, 10 lakh records will go to first partition, and to read data from each partition we will be having 1 consumer, and 10-20 lakh records will goto second partition which will be read by second consumer

Therefore, so much of data , divide it into multiple partitions and have 1 consumer per partition



Think if every current meter is sending data (called stream (flow of data)) for every minute to a topic it can’t store in a single computer the topic would have been divided into many partitions, **each partition would get stored in separate computer**, but each partition will further be divided into multiple segments, based upon the load we should decide how many partitions we want, in case of huge data prefer more partitions , so that more folders will be there and each folder can accommodate lot many number of files

1. Huge load== huge partitions (20,000 partitions) == we may need huge consumers (20,000 consumers in same consumer group, therefore each consumer reads from 1 single group) or 1 consumer in a consumer group must read data from 500-1000 partitions

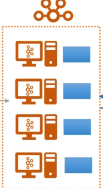
So its our responsibility to decide the number of partitions we want

1. Be careful while deciding segment size-

Low segment size🡪 A smaller segment size means that files must be closed and allocated more often, which reduces the overall efficiency of disk writes. Having a small segment size (ex:- 1MB/2MB means only few messages will be written into it, if u have huge data kafka should create multiple files) would mean while reading also if consumers are reading huge data then reading messages Kafka has to keep a lot of files open which may lead to **Too many open files**error**.**

Ex:- if u are writing 1GB data per day, if u keep 1MB as segment size then per day 1,024 files will be created in that single folder, & if consumer wants to read all data at once

Then Kafka has to open minimum 100 files at a time which leads to **Too many open files**error, so carefully choose segment size



#### **Log Retention**

**Segment Deletion:** When the oldest message in the segment exceeds the retention period then entire segment will be deleted

|  |  |
| --- | --- |
| **log.segment.bytes (max 1 GB): / log.segment.ms (max 1 week)-**   * **This is the max size of a segment after that segment will be closed**   **means expiration timer will start for that log file only after segment is locked**  **This specifies** the amount of time after which messages may be deleted  means either when segment max size 1GB is reached or when segment max time (1 week) is reached only then only segment is locked and then expiration time will be started for that segment with this we are only telling max segment size  ex:- if segement 1GB filled in 1 day, then on that day itself segment will be closed and expiration timer will be started after retention period segment will be deleted | **log.retention.bytes**   * This is the max partition folder size (means size of all segments present inside folder)   It is the maximum amount of data that a partition can hold, including all segments size  If there is more data than the configured old segments will be deleted |

**log.retention.hours –**

Remember retention period is always calculated based on last modified timestamp of that segment file

**by default** parameter, and it is set to **168 hours, the equivalent of one week**. Means once data in that segment is reached to 1GB & once segment is closed for modification after that moment 1 week expiration timer data will be deleted

will be started ,

log.retention.ms

log.retention.minutes

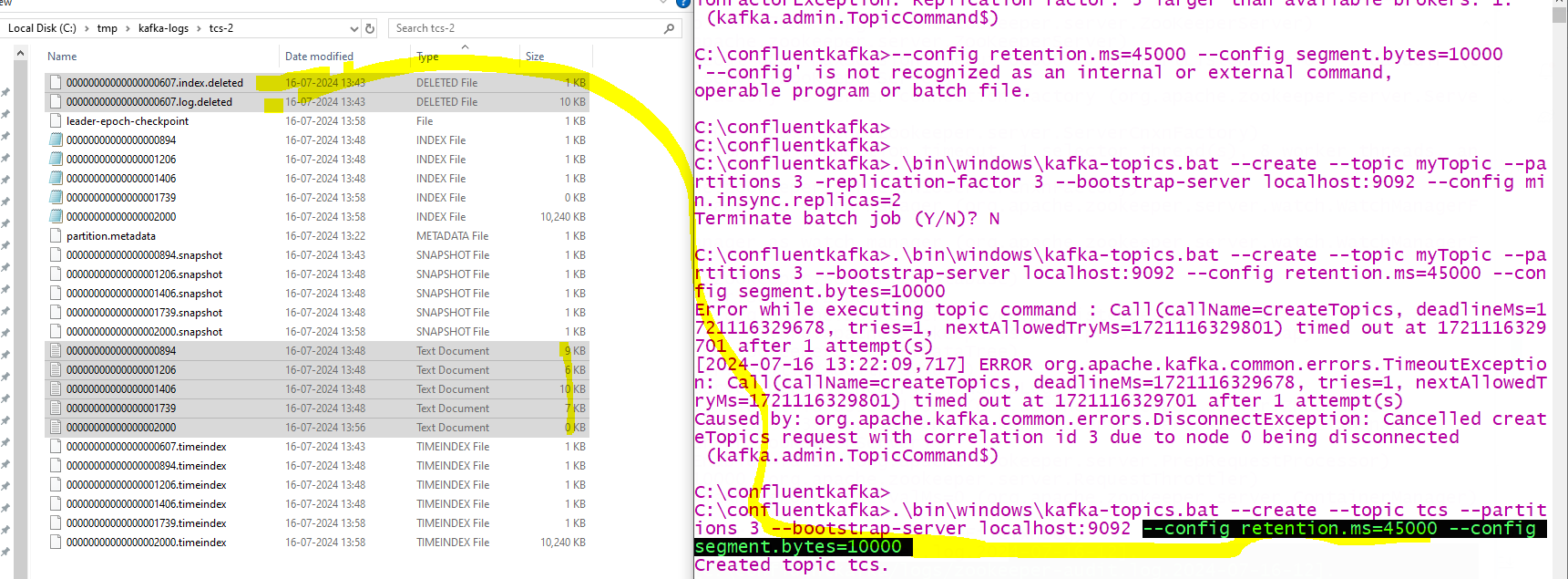
if more time is specified, the smaller unit size will take precedence.

Retention period is calculated basis on the last modified time on that segment log file, This is the time that log segment was closed & represents the timestamp of the last segment in the file, once segment file is locked then only the retention timer will be started for that file

Means any log file will be deleted after retention period from the point of file segment’s last modified time ex:- 7 days after the files last modified timestamp

Means message by message will not be deleted from a log file, entire segment file will be deleted at once after the retention period

|  |  |
| --- | --- |
| 1 week of retention (ir-respective of size) which ever is met earlier | 500 MB size |
| Remember – retention period will be counted only after the segment is closed  retention.bytes = -1 (This must be -1, because of we need 1 week, default size is 1GB which can be met in 4 days so we should not set the segment size at all)  retention.ms = 604800000 (this means 1 week) | retention.bytes = 524288000  retention.ms = -1 (we have to set time as -1,bec to get 500MB when it takes more than a week, whereas default max segment time is 1 week, segment might be closed within a week  so to avoid this we should not set the segment max time ) |



C:\confluentkafka>.\bin\windows\kafka-topics.bat --create --topic tcs --partitions 3 --bootstrap-server localhost:9092 **--config retention.ms=45000 --config segment.bytes=10000**

If u see above command we created a topic with max segment size as 10KB, so all the log files are mostly less than 10 KB and

We configured retention period as 45 seconds, means broker will scan all segment files and if it notices any file **aged more than 45 seconds** from last modified timestamp then that segment file will be deleted, already if u see above many segment files are showing as deleted.

#### Offset

Like when we came to movie theatre, each and every person in theatre will get the seat right and seat will have seat number right similarly, every message written in the file will have offset number , if we are writing 10k messages to a topic which consists of 3 partitions , if the incoming message is not having the partition number or if it doesn’t have the key all the messages will be sent equally to all partitions so as we don’t send key the partition number each partition will get 3333 messages and those will be stored in segment files , if file / segment size is cfgd as 1 MB if all 3333 messages can fall under 1MB, then all messages will store in same file /segment.

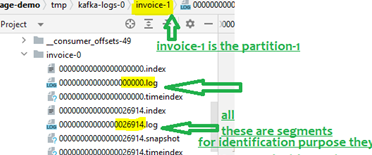
And most imp thing is every message will have some offset number like every person will have the aadhar number

But to uniquely identify a message in that topic, we should know topic name and partition number (folder number) and offset number all 3 are mandatory

Inside folder we have many files, files are called segments, message will have unique number across all files, ex:- if we know the offset number

Means offset number will be carried out to the next segment file … in the entire folder, so offset number will be unique & will not be repeated in the entire segment folder

It’s very easy to search this message across all file, because each file is numbered it have the range from which off to which offset messages it holds

in the image u see each file /segment name is having some number indicates the starting offset number which that files hold  
There are 2 types of offset

1. committed offsets: means offset will be committed frequently to the broker

props.setProperty("enable.auto.commit", "true"); by default its true

Kafka allows consumers to start fetching messages from any available offset. In order to help brokers quickly locate the message for a given offset, Kafka maintains two indexes for each segment:

* An offset to position index - It helps Kafka know what part of a segment to read to find a message
* A timestamp to offset index - It allows Kafka to find messages with a specific timestamp

#### **Fetching messages in order and fetching messages after certain time**

Two important files in above- along with .log files (which have actual messages)

.index file 🡪 This file is useful when we want to receive 1st 100 messages or from 500-680th messages, it will maintain the index, bec for that topic if we have 5 partitions/folders

All message will be distributed to all partitions , 1st msg to p1, 3rdmsg to p2,2nd pmsg to p5 like that, if those are distributed like that this index file will help to fetch all messages in order

.timeindex file 🡪 this file is useful when we want to get some messages after say 2:00 pm or after 8:00PM

When consumer asks broker to receive messages in order then .index file will be used

With offset number also we can fetch /ask broker

I still don’t know why 2 files- why in one file itself can’t we write the full data?

**Replicas**

C:\kafka\_2.13-3.3.1\bin\windows\kafka-topics.bat --create --bootstrap-server localhost:9092 --topic **invoice --partitions 5 --replication-factor 3** --config segment.bytes=1000000

In the above command we said 5 partitions and replication factor of 3

Therefore total folders=5\*3=15 folders means 5 partitions each partitions should have 3 replicas

Replication factor

If there are 2 servers then while creating 2 topics, we can have that topic max availability in 2 server

2 servers=2 replicas in total

4 sever= max 4 replicas of that topic

For 1 server we cant have 2 replicas , what is the use of having 2 replicas of that topic in same server

If that server goes down both replicas will go down

C:\kafka\_2.13-3.3.1>.\bin\windows\kafka-topics.bat --create --topic stocks --partitions 3 --replication-factor 2 --bootstrap-server localhost:9094

Error while executing topic command : Replication factor: 2 larger than available brokers: 1.

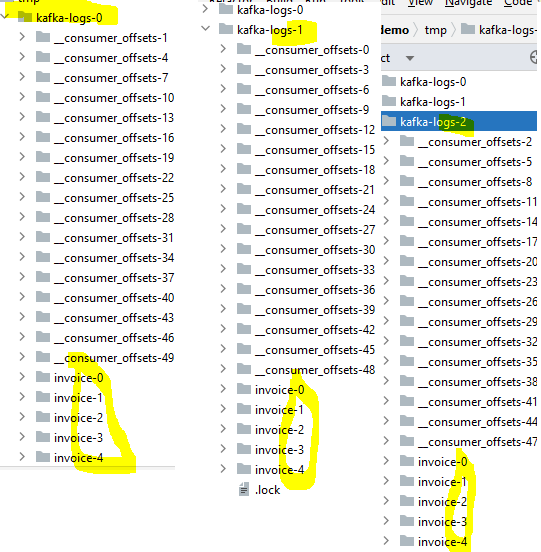
[2023-02-04 19:44:14,861] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 2 larger than available brokers: 1.

(kafka.admin.TopicCommand$)

See here we got error saying , we have only 1 server , but we are asking for 2 replicas , but we cant have both replicas in same server ,because if that server went down both replicas will go down , so always both replicas will never be created on the same server

that’s why topic creation failed due to insufficient replicas,

if we have 2 replicas then we can ask for 2 replication factor



These partitions are distributed among multiple brokers

To be a fault tolerant system kafka will not store **all partitions and its replicas** in same broker if it places think if that broker went down, all the 2 partitions folders copies present in same broker will go down & that’s all resiliency fails

Topic Replication factor means- if topic have 5 partitions and replication factor is 3, each partition will present 3 times totally so place duplicate copies on different machines

Partition offsets

Offset is a 64 bit integer giving a unique id to a message in a given partition

Like every person in Cinema Theater have a seat number

Since offset is not unique across partitions, if u want to locate a message u must know below

1. U should know the topic name (hostel name)
2. Partition name (like block name inside a hostel)
3. Offset number

Offset number /sequence number=in cinema theater it’s our seat number=message arrival order number

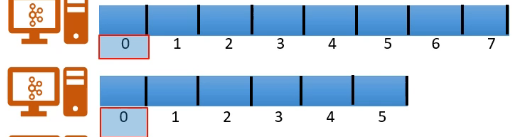
It same like in cinema theater how we get seat number , when we came to cinema theater the theater management will give us that number so that we can sit in that seat

It is the unique sequence id that is given to each arrived message in to the partitions topic by the kafka broker

Its clearly an arrival order number, when a message came broker will give

But these offset numbers are local to that partition, like another partition can also have same partition number , within each partition offset is going to start by 0 and increase by 1.

These provided numbers are immutable



Therefore if u want to locate a message then u have to know the topic name, partition or folder name (this partition number is mandatory because 1 topic may have many partitions),offset number

What if an big message came ?

Will it be broken and stored across many partitions and each part will have some offset number?? – Never every message will sit and will be assigned with some offset number , message cam not be broken at all

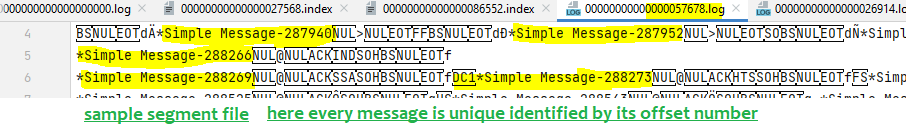
**Detailed explanation:-**

Analogy:- lets take a Narayana hostel, we have multiple blocks like Krishna, Cauveri, Godavari

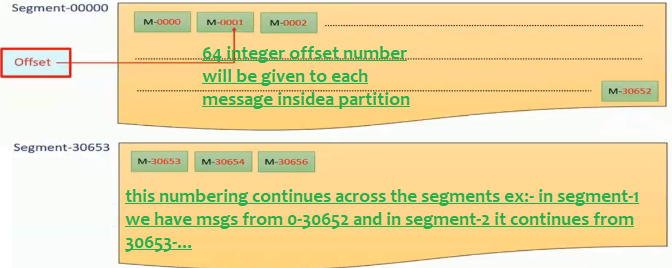
For Krishna block /partition we have room num ranging from 1-40 (here 1-40 is offset number, consider each block is a partition

Inside a partition/Block, all offset number are unique, and for another partition same offset number will be there )

Same for Cauvery block also we have rooms ranging from 1-40 (these offset number 1-40 are same for previous partition Krishna also, so offset numbers are unique inside a partition only)



Every message inside a partition can be uniquely identified with its offset number

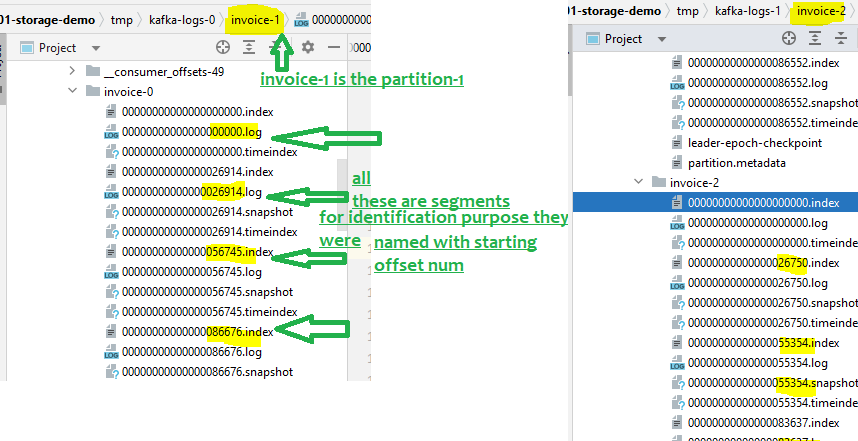


64 bit integer offset number will be given to each arrived message,

offset number will start with zero inside every partition , offsets number ranging is unique only inside a partition

1st partition- offset numb of a message starts with 0-30653(Numb in Krishna block)

2nd partition also offset starts with 0-30653(Numb in Cauvery block)



As each partition will be divided into segments

From the above understand that, offset numbers from previous segment numbers will be carry forwarded to other segment also and offset numbers are sequential across the segments inside a partition

Consumer group

If multiple consumers want to share the work, then they will form as a group and share the load, its same like podupu group ,

Ex:- if multiple people wants to share work they will be in same team as like a software team

Let’s say if topic is having 3 partitions, if data is coming to 3 partitions then if we create a consumer group, if we have 3 consumers in that group each consumer will read from different partitions

Log compactions

Here log means data~~ because messages/data is stored in log files ultimately

Log compactions means deleting the log files,

**Time Based Retention**

All these are broker properties

1. U can delete based on the age of the segment file–this is default one, so by default msgs will be deleted based on the age of messages

Log.cleanup.**policy=delete** //by default once segment max size (1GB) or segment max time (1 week) is reached, the segment is locked for modification & expiration timer will start on that

Segment file, once retention period is over then that file segment will be deleted, this is called time based retention

* log.retention.hours=168 // Deleting based on hours
* log.retention.minute=1 //Delete for every minute
* log.retention.ms=1 //deleting for every few milliseconds

|  |  |  |
| --- | --- | --- |
| **log.retention.check.interval.ms** | Default is **= 5minutes** | This is the time that log cleaner will check if there is any log file is eligible for deletion or not  Here for every 5 minutes log cleaner will scan all the segment files if for any file if retention period is over then broker will delete that file |

|  |  |
| --- | --- |
| Log.cleanup.**policy=delete** | In this entire segment file will be deleted after retention period, ex:- Thread will monitor every segment file last modified time stamp  Current time > last modified timestamp+ retention period entire segment file will be deleted at once |
| **log.cleanup.policy =compact** | Here only messages with old values for same keys will be deleted like hashmap |

**Key Based Retention**

There is another way of retention – this is called key based retention, we can decide what retention we want based on value for key **log.cleanup.policy =compact**

With this policy, Kafka retains only the latest value for each key, any older message with same key will be discarded

**This exactly works like a Hashmap – if another entry came with new key- old value will be replaced with new value, here also data with old keys will be deleted if another key came with new value**

1. Delete the data based on the key –old, if an entry with same key is repeated, the old entry will be marked for deletion & only new entry will be retained

With this policy kafka will stores only most recent value for a key, old values will be overridden

Generally the cleaner will scan for every 15 seconds **(log.cleaner.backoff.ms=15000),** only after this time the cleaner will check all the messages present in segment file to detect if there are any old values for that key,

new values will be copied to new segment file called replacement segment

if u want cleaner to run so sooner then decrease the run frequency

Log.cleanup.**policy=compact**

log.cleaner.min.cleanable.ratio = 0.5 –means perform cleaning when 50% of the segment is having duplicate keys

Example for log compaction

Create a new segment for every 5 seconds with cleanup policy = compact so that old keys value pairs will be overridden by new key value pairs like hashmap

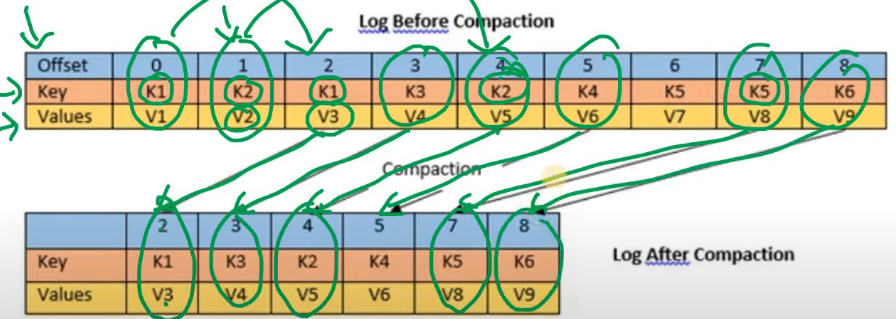
.\bin\windows\kafka-topics.bat --create --topic myTopic --partitions 1 –replication-factor 3 --bootstrap-server localhost:9092 --config cleanup.policy=compact

Benefits:-

With this approach we can keep only the latest info,

Delete the messages based on duplicate key, old messages with same key will be deleted

Ok this is the concept but I am not sure whether these properties are broker configs alone or topic properties too? As per below topic creation command he gave these values while creating topic too



Here in right side image if u see old Ale, Ben values were deleted & only new were left

Here log compaction means records/messages will be deleted based on the key of the message, if after sometime if same key comes then old entry will be deleted and new entry will be created. Generally after certain size reached all will be deleted but now deletion is happening based on the keys

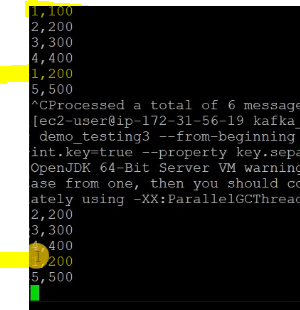
Here after deleting the old message with the same key re-ordering will not happen, only deletion will happen

Generally older messages will be deleted based on time, means older messages will be deleted first

But if u want to delete messages based on the duplicate keys

bin/kafka-topics.sh --create --topic demo\_testing3 --bootstrap-server 54.90.61.129:9092 --replication-factor 1 --partitions 1 --config cleanup.policy=compact --config min.cleanable.dirty.ratio=0.5 --config segment.ms=5000

here min.cleanable.dirty.ratio means when this compaction should happen ,when database is 0.5 or 50% dirty then this should happen, means if 50% keys are duplicate then do this

if u see this message here 1,100 where 1 is key and 100 is value and second time we kept 1,200 back scenes 1,100 that old value will be deleted , so now when I started the new consumer with new consumer group old 1,100 was gone as compaction ran based on the key not based on the time

As of now unable to replicate this scenario, my kafka server is shutting down automatically so unable to replicate it locally  
if ur application is having an incremental change then only prefer this compaction (deleting data with old keys)

How compaction will happen internally

1. 1st we need to identify the dirty segment- a segment is called dirty when that file contains more duplicate keys

When to go for this log compaction??

If the consumer only wants latest data then prefer this

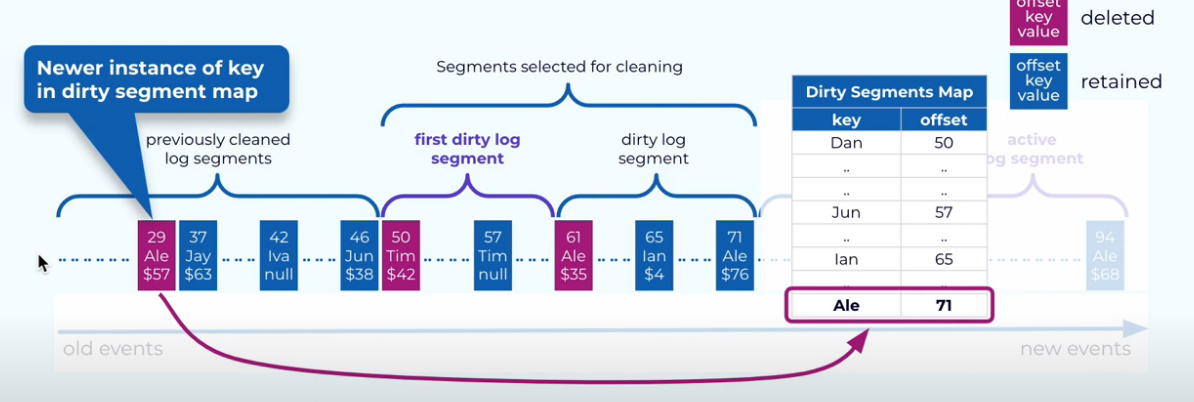
Ex:- let say we keep a message in kafka when his salary is incremented, but mostly all consumer wants his latest salary, as they are not interested in salary updates

Benefits:-

Saves storage space by removing duplicate data. All entries with old key will be removed & only new entries with new key will be retained

Makes it easier to keep track of the latest changes to data. Un-necessary Old data will be deleted automatically so that only new data will be present always

How this compaction happens internally??



Here if you see we are building a map contains a key & latest offset, it will scan through all segments in current folder partition and updates the latest offset number for that key

Once map is built, since the map have the key & latest offset number all the old offset numbers will be deleted for that key

**So whats problem here**?? – here the comparison is happening only with segment files present in same folder, but what if the duplicate keys are present in multiple segments belonging to multiple folders? It cant even compare because the map contains the offset number alone, but same offset numbers are duplicated/repeated in another folder also?

Log compaction in Kafka is limited to removing duplicate keys within the segments of the same partition, It does not operate across multiple partitions.

Why because it internally delete based on offset number which is unique only inside a partition

**Partition-based:** Kafka is inherently partitioned. Each partition is an ordered, append-only log.

**Duplicate Removal:** Older messages with the same key are deleted, leaving only the most recent value.

**Internally it will delete old segments & it will paste that new filtered data into new segment files**

|  |  |
| --- | --- |
| Key | Value as a list |
| Mani | [~~101,201~~,301] |
| Sai | [~~299,399~~,499] |
| Santhoshi | ~~[285,385~~,485] |

So as 1 folder will have many segment files, this cleaner will open each & every segment file & scans through all messages and maintains keys for the message as keys in hashmap

And offset numbers will be the value in hashmap

All those old values of that key will be marked with some delete tombstone markers, and after log.cleaner.delete.retention.ms time these old values for that keys will be deleted

* The default value for log.cleaner.delete.retention.ms is **24 hours (one day)**.

The offsets are kept, only some messages are deleted.

Concept of tombstone

Problem with log compaction:-

1. Consumers won't see every update to a record, they can see only the latest one.
2. Cleaning happens only partition wise- This cleaning and comparison of data will happen only among segment files present inside a partition folder

But what about duplicate data present in multiple folder partitions,

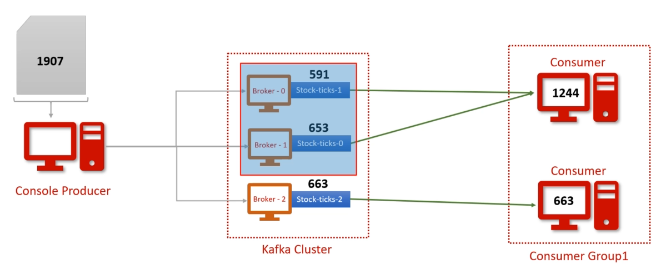
Ex:- in the above image key named “mani” is present in 2 partitions (partition-1,partition-2), but even after cleaning key – “mani” will be present in 2 places

1st occurrence In partition -1 & 2nd occurrence in partition-2, **so duplicates will be deleted only inside a partition not across partitions**

Zookeeper

Zookeeper will have all brokers information , kafka announced zookeeper will be retiring soon

Examples-sending 1 file to 3 consumers



Now in this case, we are sending bulk of records 10,000 records

Producer will send the data to 3 partitions evenly , even though its single message/single file it will be split and 1st 5k rows to partition 1 and next 2.5k to partition 2 and rest to partition 3

And since we have only 1 consumer(1 consumer group) but as we have 2 consumers in same group these are workers , these will share the load among the existing partitions

1st consumer of that consumer grp will read the records from 2 partitions

2nd consumer of that same consumer grp will read data from 3rd partition

Create a new topic called stocks

.\bin\windows\kafka-topics.bat --create --topic stocks --partitions 3 –replication-factor 1 --bootstrap-server localhost:9092

Console Producer who is pushing records from excel file

Create a producer and gave a sample csv file as below then the producer will put that file to the kafka server

**Command**

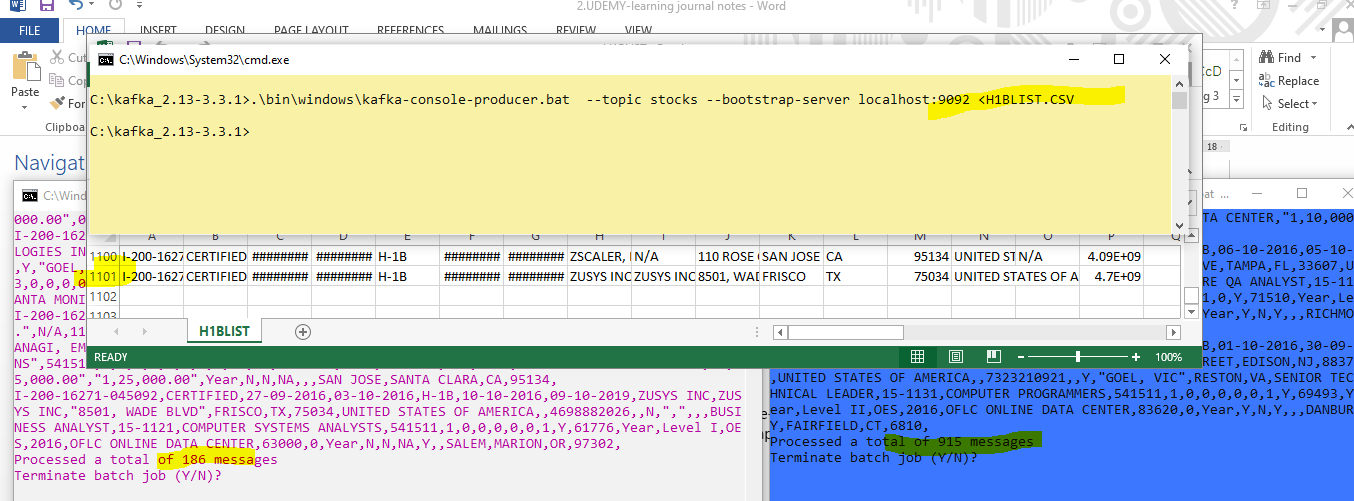
C:\kafka\_2.13-3.3.1>**.\bin\windows\kafka-console-producer.bat --topic stocks --bootstrap-server localhost:9092 <H1BLIST.CSV**

make sure U have the file

Create 2 console consumers who is present in same group

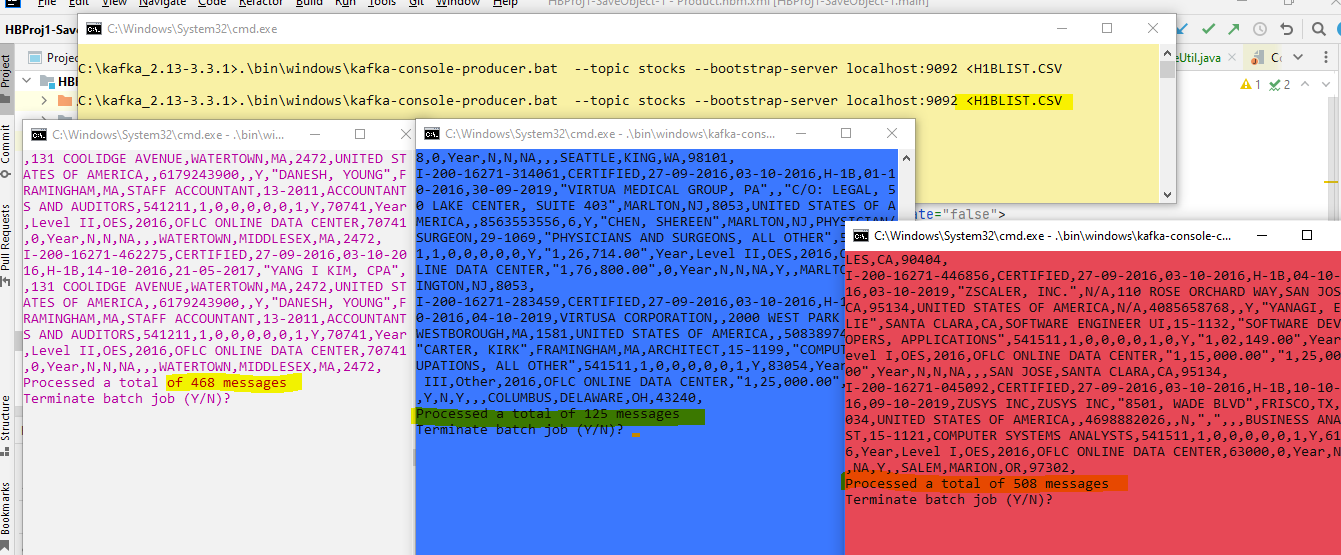
C:\kafka\_2.13-3.3.1**>.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic stocks --from-beginning --group g1**

Above command says create a topic called stocks and read from beginning and make it as a consumer under group g1



This time for 3 partitions take 3 consumers in same consumer group then load will be distributed

Among 3 consumers



Kafka cluster

Kafka can manage hundreds of brokers in a single cluster

And every broker in a cluster will have the broker id

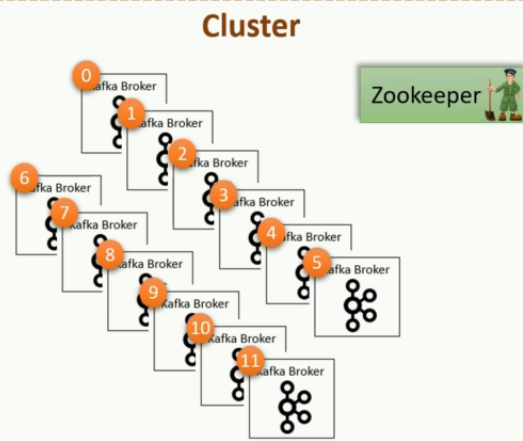
If a broker is active zookeeper will maintain an ephemeral (temporary) node, once the broker is down , zookeeper will delete that node from him

Kafka doesn’t encourage master and slave architecture, just one of the broker will act as a controller it will do both jobs –regular broker job and activities like when a broker dies that broker task will be reassigned to another broker (which ever broker joins cluster first ) by this controller only

At any point of time there will be only 1 controller

As work load goes increasing keep/ cascade more brokers

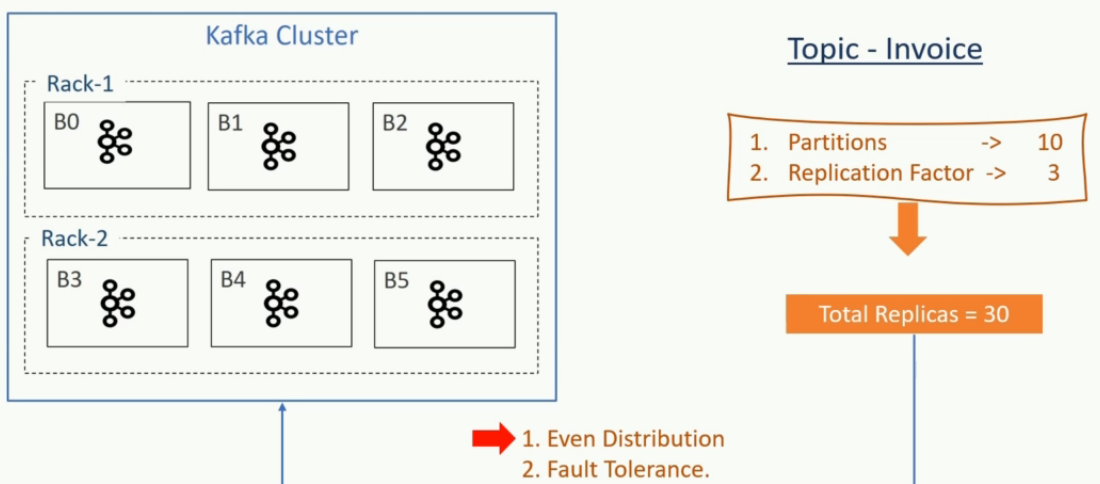
If broker-0 who is the leader went down , then broker-1 will be elected as the new leader and after some time even if the old died leader broker-0 came back also he will not become the leader because the leader is already there

if u ask for 3 replicas of a partition each replica will be stored in a different kafka broker, so that if one broker goes down I, we still have another broker as backup as we have stored 2nd replica in second broker

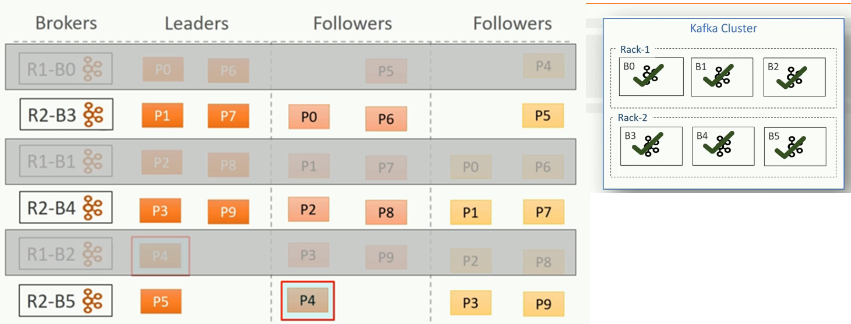
Partition assignment

Think 1 topic is having 5 partitions, but all 5 partitions should not be in same broker,

If the broker goes down all replicas of that topic will go down, hence we should place each replica in different broker or in diff rack

In real time in production each broker will be in separate machine , since kafka is horizantally scalable those are deployed on separate machines whereas in local to get the feel we will start kafka server 6 timeson 6 diff port numbershere 30 partitions means 30 folders

Fault tolerant system means not only if the broker goes down , it will consider the case if the entire rack which consists of many brokers goes down then also it should be able to withstand so that partition arrangement should be like that



So far we used to think what if the broker goes down**? But now think what if the entire rack (which consists of many brokers) goes down?**

**we should distribute the replicas partition/replica folders in such a fashion**

See In above even if that entire rack-1 goes down where if B0,B1,B2 all 3 goes down also it should be able to manage with the replicas available

This is how kafka will become fault tolerant

Partition follower

The follower responsibility is to copy the data from the producer and stay up to date, these followers should stay in sync with the leader else they can’t become the leader when the leader goes down

They cant get elected as the leader when they fall behind the leader

Its same like team members and team lead, all the team members should be in sync knowledge with the leader else when the leader kept resignation they cant become the leader

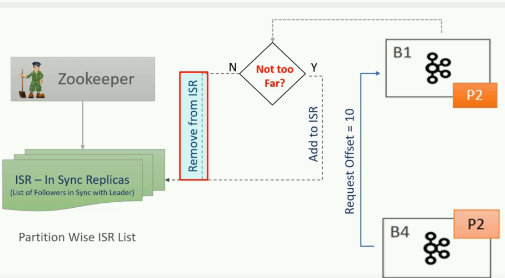
How can a follower stay in sync with leader?

In this case follower only requests leader and leader will some messages and follower will persists them in some segment files and again follower asks for more and leader will give some more and this process continues

How does a leader know if the follower is in sync or not?

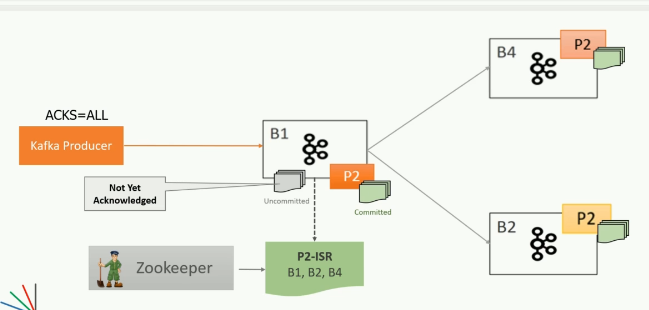
It is based on offset number , for every message sits in a segment of a partition kafka will have 64 bit integer offset number , like how we have seat number in the cinema hall ,

Follower will ask the offset number from 0-100 and leader will give messages from offset 0-100 similarly follower will ask the next set of messages with the offset number



Replica will be in ISR list only when the replica is not too far behind the leader , generally the follower should ask and leader should provide the messages from the offset this will take sometime so configure it as 10 seconds, so maximum the follower can be 10 seconds behind the leader , so if the consumer offset’s are not too far from the producer then the follower will be in ISR-In sync replica list else that follower will be removed from the ISR list

#### Make more Kafka more fault tolerant



Generally the follower can be 10 sec behind the leader , lets say if all the followers are 20 sec behind the leader then no one will be in the In sync replica list suppose in case if suddenly if the leader goes down and in sync replica list is also empty then which follower should be elected as leader , if I select any other follower who is not in ISR list they all will be 20 seconds behind the leader I will face data loss because the followers are 20 sec behind means many messages they don’t have which leader have

**So if we choose them there is a data loss what is the solution?**

1. Only solution is make sure message is considered as received only when all the followers replicas also consumed , Leader should provide the acknowledgement back to the producer only when all the followers /replicas received the message, so configure kafka in such a way acks=all so that broker will send positive acknowledgement only when all ISR received it and if any ISR fails to receive the message then broker wont give acknowledgement, then producer will retry to send that message again

Minimum ISR

Minimum ISR must be set to 2, by definition, committed data exists on all ISRs.

Analogy:- suddenly if team leader left the organization, who among the team will become next leader

Problem:- if all the followers are not in sync with the leader they will get removed from the ISR list

Then only leader will be there , if the leader suddenly dies and no one is present in the ISR list

If all the follwers are soo much behind if we elect those followers also we will loose some messages

* + - lets say acks=all means broker will give acknowledgement only when all ISR’s received the messages, lets say all followers are 20 sec behind the leader then broker will remove the all followers from ISR, so ISR list will be empty, now when a message received acks=all means only when all ISR’s received the message it should send back acknowledgement , now there is no follower in ISR list other than leader, now once leader receives message it will send acknowledgement which is wrong , so always configure minimum number of ISRs in else leader only will send acknowledgement, if leader fails no backup which is data loss
  + If u do like this, the broker will send back the acknowledgement only if all followers in ISR received the messages if one of the follower in ISR didn’t received the message the broker wont send the acknowledgement back the producer and producer will think message is not received and it will resend again
  + But if u do like this when message is received by leader when its not received by follower in ISR and so borker wont send ack and producer will resend it then in this case broker will receive the duplicate message na??? how will you solve the problem

Min.insync.replicas=2

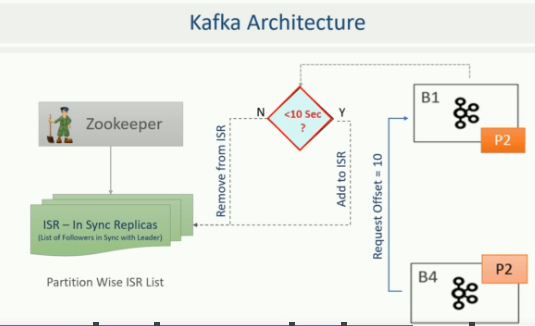
Even after configuring min.iSR.REPLCIAS=2 What will happen if suddenly one ISR went donw??

Then the broker wont accept any more message and instead it will throw not enough replicas exception.

Until the a follower joins ISR it wont accept any message and you can read from the partition but u cant write to that

The only issue is when 2 –broker+1 follower is not ready then Kafka won’t allow to insert

But still u can read a message



The default value of not too far is 10 seconds, means the follower can be 10 seconds behind the leader

How do we know whether the leader is in the ISR list or not?

Every Kafka message will have the offset id and timestamp, If Consumer ask the message who offset is more than 10 seconds then broker will understand its asking old messages and it will remove that from the ISR list, its same like if we ask silly questions to the team lead he will understand that this guy don’t know this also so he consider him as a poor knowledge guy and removes from the team lead list

## Follower brokers

These brokers doesn’t server producers and consumer requests, their only job is to stay in –sync with leader, because when the leader nodes goes down if followers are insync

Then only a follower can become leader, else they can’t become leader means if followers are falling behind or if out to sync then they can’t become the leader, same like real life scenario

How can these followers can stay in sync?

These followers (Follower threads) only should ask leader node, bro u got latest messages right give us, likewise how we ask these team lead

Team lead knows who are in sync- similarly leader broker knows which follower is in sync (in sync replicas)

How leader knows if a follower is in sync or not? Based on the offset number a follower is asking, if follower is asking a offset which is very old then leader understands that guy is out of sync

Like when veera ask such a basic question then sriram comes to know where he stands , so by looking at the offset number a follower is asking by that itself leader can guage

If follower is very much behind leader will remove that follower from isr list

Producer

Producer always sends the data to the leader partition folder, similarly consumer always consumes from the leader partition folder not replicas, because replicas may not have new data

Later sync will happen from leader partition for follower partition

#### Callbacks

Make sure u attach a call back for each and every message u sent

After sending we can always get acknolwedgements

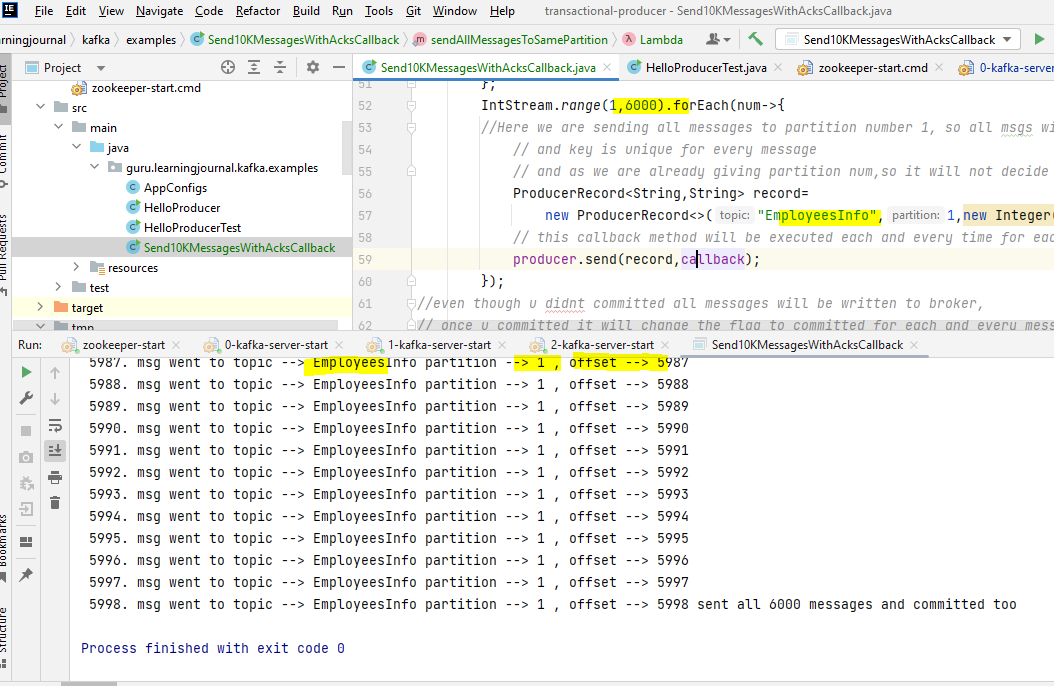
If we send 1 million message we will receive acknowledgements for all those messages and we will get confirm as and when callbacks are executed automatically

This is the call back object

Callback callback=(recMetadata,z)->{  
 int callbackExecutedCount= ac.getAndIncrement();  
 System.*out*.printf("\n %d. msg went to topic --> %s partition --> %d , offset --> %s ",  
 callbackExecutedCount,recMetadata.topic(),recMetadata.partition(),recMetadata.offset());  
  
};

// For send method we should pass callback

ProducerRecord<String,String> record=  
 new ProducerRecord<>("EmployeesInfo",1,new Integer(num).toString(),"Msg--"+num);  
*// this callback method will be executed each and every time for each and every message*producer.send(record,callback);



In the above message u can see my callback executed 6000 times almost I received acknowledgements for all the messages which I sent