*SREE VISVESVARAYA INSTITUTE OF TECHNOLOGY&SCIENCE*

 *DEPARTMENT:-COMPUTER SCIENCE&ENGINEERING*

 *AUTHOR:-K.VARUN KUMAR REDDY*

 *TOPIC:-KAFKA*

.INTRODUCTION

In Big Data, an enormous volume of data is used. Regarding data, we have two main challenges.The first challenge is how to collect large volume of data and the second challenge is to analyze the collected data. To overcome those challenges, you must need a messaging system.Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing application

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system. Two types of messaging patterns are available − one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow **pub-sub**.

1. **Point to Point Messaging System**

In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.

Building real-time streaming data pipelines that reliably get data between systems or applications Building real-time streaming applications that transform or react to the streams of dataTounderstandhowKafka does the somethings ,let's divein and explore Kafka's capabilities from the bottomup.

First a few concepts:

1. Kafkais runas a clusteron oneor moreservers.
2. TheKafka cluster storesstreams of records in categoriescalled
3. topics. Each recordconsistsof a key,avalue,and atimestamp.
4. Kafkahas fourcore APIs:

APACHEKAFKA 1

1. The ProducerAPIallowsanapplicationtopublisha streamofrecordstoone ormore Kafka topics.
2. TheConsumerAPI allowsanapplicationtosubscribetooneormoretopicsandprocessthe stream of records produced to them.
3. TheStreamsAPIallowsanapplicationtoactasastreamprocessor,consuminganinputstream fromoneormoretopicsandproducinganoutputstreamtooneor moreoutputtopics, effectivelytransformingtheinputstreams to outputstreams.

TheConnectorAPIallowsbuildingandrunningreusableproducersorconsumersthat connect Kafkatopicstoexistingapplicationsor data systems.Forexample,a connectortoa relational databasemight captureeverychangeto a table.

InKafka thecommunicationbetweentheclientsandtheserversisdonewitha simple,high- performance, languageagnosticTCP protocol. This protocolis versioned and maintains backwardscompatibilitywitholderversion.WeprovideaJavaclientforKafka,butclientsare available in manylanguages.

**Fig1**

APACHEKAFKA

**3.BLOCK DIAGRAM**

.

Kafkaisdeployedasaclusterimplementedononeormore servers.Theclusteris capable ofstoring‘**topics**’whichconsistsstreamsof‘**records**’.Every recordholdsthree details, a key, a value, a timestamp.**Brokers**are the abstractions which manages the

persistenceand replication ofmessage.

Figure2.1Block diagramofthesystem

Brokersprocesstopicsinpartitions.Apartition ononebrokerinaclusteristheleader.The samepartitionismirroredononeormoreotherbrokersintheclusterasreplicas.Whenaleader goesoffline,areplicaautomaticallytakesitsplaceandbecomesthenewleaderforthetopic.

An in-syncreplicais a replicathat is completelyup-to-date with theleader.

APACHEKAFKA 4

EachKafkaclusterhasonebrokerthatalsoactsasthecontroller.The controller isresponsible formanagingthestatesofpartitionsand replicas. It alsoperforms administrativetasks,such as reassigningpartitions.

[*Topics and Logs*](https://kafka.apache.org/documentation/#intro_topics)

Let'sfirst diveintothe coreabstraction Kafkaprovides for astream of records—the topic.

Atopicis acategoryor feednameto which recordsarepublished.Topics inKafka arealways multi-subscriber;thatis,atopiccanhavezero,one,ormany consumersthatsubscribetothe data written to it.

For each topic, the Kafkacluster maintains a partitioned logthat looks likethis:

Eachpartitionisanordered,immutablesequenceofrecordsthatiscontinuallyappendedto— astructuredcommitlog. Therecordsinthepartitionsareeachassignedasequentialidnumber called the *offset*that uniquelyidentifieseach record within the partition.

TheKafkaclusterretainsallpublishedrecords—whetherornotthey havebeenconsumed— usingaconfigurableretentionperiod.Forexample,iftheretentionpolicy issettotwodays, then for thetwo daysafter arecord is published, itis available for consumption, afterwhich it willbediscardedtofreeupspace.Kafka'sperformanceiseffectively constantwithrespectto

data sizeso storingdatafor alongtime is not aproblem.

APACHEKAFKA 5

Infact,theonly metadataretainedonaper-consumerbasisistheoffsetorpositionofthat consumerinthelog.Thisoffsetiscontrolledby theconsumer:normallyaconsumerwill advanceitsoffsetlinearlyasitreadsrecords,but,infact,sincethepositioniscontrolledbythe consumeritcan consumerecordsin anyorderitlikes.Forexampleaconsumercanresetto an older offsettoreprocessdata fromthepastorskipaheadtothe mostrecentrecordandstart consumingfrom "now".

ThiscombinationoffeaturesmeansthatKafkaconsumersareverycheap—theycancomeand gowithoutmuchimpact ontheclusteroronotherconsumers.Forexample,youcanuseour commandlinetoolsto"tail"thecontentsofany topicwithoutchangingwhatisconsumedby anyexistingconsumers.

Thepartitionsinthelogserveseveralpurposes.First,theyallowthelogtoscalebeyondasize that willfit on asingleserver. Eachindividual partition must fiton the servers that hostit, but atopicmayhavemanypartitionssoitcanhandle anarbitraryamountofdata.Secondtheyact as the unit of parallelism—moreon that in a bit.

[*Distribution*](https://kafka.apache.org/documentation/#intro_distribution)

Thepartitionsofthelogare distributedovertheserversintheKafka clusterwitheach server handling dataandrequestsforashareofthepartitions.Eachpartitionisreplicatedacrossa configurable number ofserversforfaulttolerance.

Eachpartitionhasoneserverwhichactsasthe"leader" andzeroormoreserverswhichactas

"followers".Theleaderhandlesallreadandwriterequestsforthepartitionwhilethefollowers passivelyreplicatetheleader.Iftheleaderfails,oneofthefollowerswillautomaticallybecome

APACHEKAFKA 6

thenewleader.Eachserveractsasaleaderforsomeofitspartitionsandafollowerforothers so load is wellbalancedwithin the cluster.

[*Producers*](https://kafka.apache.org/documentation/#intro_producers)

Producers publishdata tothe topicsof theirchoice.The producer isresponsible forchoosing whichrecordtoassigntowhichpartitionwithinthetopic.Thiscanbe done ina round-robin fashionsimplytobalanceloadoritcan bedoneaccordingtosomesemanticpartitionfunction (saybased on somekeyin the record). Moreon the useof partitioningin asecond!

[*Consumers*](https://kafka.apache.org/documentation/#intro_consumers)

Consumerslabelthemselveswitha *consumergroup*name,andeachrecordpublishedtoatopic isdeliveredtooneconsumerinstancewithineachsubscribing consumergroup.Consumer instances can bein separate processes oron separate machines.

Ifalltheconsumerinstanceshave thesameconsumergroup, thentherecords willeffectively beload balanced over theconsumer instances.

Ifalltheconsumerinstanceshavedifferent consumergroups,theneachrecordwillbe broadcast to alltheconsumer processes.

AtwoserverKafkaclusterhostingfourpartitions(P0-P3)withtwo consumergroups. Consumer groupA has two consumer instances and groupBhasfour.

APACHEKAFKA 7

Morecommonly,however,wehavefoundthattopicshaveasmallnumberofconsumergroups, oneforeach"logicalsubscriber".Eachgroupiscomposedofmany consumerinstancesfor scalability andfaulttolerance.Thisisnothingmorethanpublish-subscribesemanticswhere the subscriber is aclusterof consumers instead ofasingle process.

Thewayconsumption isimplemented in Kafkaisbydividingupthe partitionsin thelogover theconsumerinstancessothateachinstanceistheexclusiveconsumerofa"fairshare" of partitionsatanypointintime.Thisprocessofmaintainingmembershipinthegroupishandled by theKafkaprotocoldynamically.Ifnewinstancesjointhegroupthey willtakeoversome partitionsfromothermembersofthegroup;ifaninstancedies,itspartitionswillbedistributed to theremaininginstances.

Kafka only provides a total order over records*within* a partition, not between different partitionsinatopic.Per-partitionorderingcombinedwiththeabilitytopartitiondatabykeyis sufficientfor mostapplications.However, ifyou requirea totalorderoverrecordsthiscanbe achievedwithatopicthathasonlyonepartition,thoughthiswillmeanonlyoneconsumer

process per consumergroup.

APACHEKAFKA 8

**4.APPLICATION**

.

[*Messaging*](https://kafka.apache.org/documentation/#uses_messaging)

Kafkaworkswellasareplacementforamoretraditionalmessagebroker.Messagebrokersare usedforavariety ofreasons(todecoupleprocessingfromdataproducers,tobuffer unprocessedmessages,etc).Incomparisontomostmessaging systemsKafka hasbetter throughput,built-inpartitioning,replication,andfault-tolerancewhichmakesitagoodsolution forlargescale messageprocessingapplications.

Inourexperiencemessagingusesareoftencomparatively low-throughput,butmayrequire low end-to-end latencyand often depend on the strongdurabilityguarantees Kafkaprovides.

In this domain Kafka is comparable to traditional messaging systems such as [ActiveMQor](http://activemq.apache.org/)[RabbitMQ.](https://www.rabbitmq.com/)

[*WebsiteActivityTracking*](https://kafka.apache.org/documentation/#uses_website)

TheoriginalusecaseforKafkawastobeabletorebuildauseractivity trackingpipelineasa setofreal-timepublish-subscribefeeds.Thismeanssiteactivity(pageviews,searches,orother actionsusersmay take)ispublishedtocentraltopicswithonetopicperactivity type.These feedsareavailablefor subscriptionforarangeofusecasesincluding real-timeprocessing, real-timemonitoring, and loadingintoHadoopor offlinedatawarehousingsystemsforoffline processingand reporting.

Activity trackingisoftenveryhighvolumeasmanyactivitymessagesaregeneratedforeach user pageview.

[*Metrics*](https://kafka.apache.org/documentation/#uses_metrics)

Kafka isoftenusedforoperationalmonitoring data.Thisinvolvesaggregating statisticsfrom distributed applications to producecentralized feeds of operational data.

[*Log Aggregation*](https://kafka.apache.org/documentation/#uses_logs)

Many peopleuseKafkaasareplacementforalogaggregationsolution.Log aggregation typicallycollects physical logfiles offservers andputs them ina central place(afileserver or

APACHEKAFKA 9

HDFSperhaps)forprocessing.Kafkaabstractsaway thedetailsoffilesandgivesacleaner abstractionoflog oreventdataasastreamofmessages.Thisallowsforlower-latency processing andeasiersupportformultipledatasourcesanddistributed dataconsumption.In comparisontolog-centricsystemslike ScribeorFlume,Kafkaoffersequally good performance,strongerdurability guaranteesduetoreplication,andmuchlowerend-to-end latency.

[*StreamProcessing*](https://kafka.apache.org/documentation/#uses_streamprocessing)

Manyusers ofKafkaprocess datain processingpipelines consistingof multiple stages,where rawinputdataisconsumedfromKafka topicsandthenaggregated,enriched,orotherwise transformed intonew topicsfor further consumptionor follow-upprocessing.For example,a processing pipelineforrecommending newsarticlesmightcrawlarticlecontentfromRSS feedsandpublishittoan"articles" topic;furtherprocessingmightnormalizeordeduplicate thiscontentandpublishedthecleansedarticlecontenttoanewtopic;afinalprocessing stage mightattempttorecommendthiscontenttousers.Suchprocessing pipelinescreategraphsof real-time dataflowsbasedontheindividualtopics. Startingin0.10.0.0,a light-weightbut powerfulstreamprocessinglibrarycalled [KafkaStreamsis](https://kafka.apache.org/%7B%7Bversion%7D%7D/documentation/streams)availableinApacheKafkato perform such data processingas describedabove.Apart fromKafkaStreams, alternativeopen

sourcestream processing tools include [ApacheStorma](https://storm.apache.org/)nd [ApacheSamza.](http://samza.apache.org/)

[*Event Sourcing*](https://kafka.apache.org/documentation/#uses_eventsourcing)

[Eventsourcingis](http://martinfowler.com/eaaDev/EventSourcing.html)astyleofapplicationdesignwherestatechangesareloggedasatime-ordered sequenceofrecords.Kafka'ssupportforvery largestoredlogdatamakesitanexcellent backend for anapplication builtin this style.

[*Commit Log*](https://kafka.apache.org/documentation/#uses_commitlog)

Kafkacanserveasakindofexternalcommit-logforadistributedsystem.Thelog helps replicatedatabetweennodesandactsasare-syncing mechanismfor failednodestorestore theirdata.The [logcompactionf](https://kafka.apache.org/documentation.html#compaction)eatureinKafkahelpssupportthisusage.InthisusageKafka

is similar to [ApacheBookKeeperproj](http://zookeeper.apache.org/bookkeeper/)ect.

APACHEKAFKA 10

[**5. Connect toimport/export data**](https://kafka.apache.org/documentation/#quickstart_kafkaconnect)

Writing datafromtheconsoleandwriting itbacktotheconsoleisaconvenientplacetostart, butyou'llprobably wanttousedatafromothersourcesorexportdatafromKafkatoother systems.Formany systems,insteadofwritingcustomintegrationcodeyoucanuseKafka Connect to import or export data.

Kafka Connectisatool includedwithKafka thatimportsandexportsdatatoKafka.Itisan extensibletoolthatruns*connectors*,which implementthecustomlogicforinteractingwith an externalsystem.Inthisquickstartwe'llseehowtorunKafka Connectwithsimpleconnectors that import data from afile to aKafkatopic and export data from aKafkatopic to afile.

First, we'llstart bycreatingsomeseed data to testwith:

>**echo -e"foo\nbar"> test.txt**

Next,we'llstarttwoconnectorsrunningin*standalone* mode,whichmeanstheyruninasingle, local,dedicatedprocess.Weprovidethreeconfigurationfilesasparameters.Thefirstisalways theconfigurationfortheKafkaConnectprocess,containingcommonconfigurationsuchasthe Kafkabrokers to connectto and theserialization format fordata. Theremainingconfiguration fileseachspecify aconnectortocreate.Thesefilesincludeauniqueconnectorname,the connectorclass to instantiate, and anyother configuration required bytheconnector.

>**bin/connect-standalone.shconfig/connect-standalone.propertiesconfig/connect-file- source.properties config/connect-file-sink.properties**

These sampleconfigurationfiles,includedwithKafka,use the defaultlocalcluster configurationyoustartedearlierandcreatetwoconnectors:thefirstisasourceconnectorthat readslinesfromaninputfileandproduceseachtoaKafka topicandthesecondisasink connectorthatreadsmessagesfromaKafkatopicandproduceseachasalineinanoutputfile.

Duringstartup you'llseeanumberoflogmessages,includingsomeindicatingthatthe connectorsare being instantiated.OncetheKafkaConnectprocesshas started,thesource connector shouldstartreadinglines fromtest.txt andproducingthemtothetopicconnect-test,

andthesinkconnectorshouldstartreadingmessagesfromthetopicconnect-test andwrite

APACHEKAFKA 11

themtothefile test.sink.txt.Wecanverifythedatahasbeendeliveredthroughtheentire pipeline byexaminingthe contents of theoutput file:

>**cat test.sink.txt**

foo bar

Notethat the data is beingstored in theKafkatopicconnect-test, so wecanalso run aconsole consumer to seethe datain thetopic(oruse custom consumer codeto process it):

>**bin/kafka-console-consumer.sh--bootstrap-serverlocalhost:9092--topicconnect-test-**

**-from-beginning**

{"schema":{"type":"string","optional":false},"payload":"foo"}

{"schema":{"type":"string","optional":false},"payload":"bar"}

...

Theconnectorscontinuetoprocessdata,sowecanadddatatothefileandseeitmovethrough the pipeline:

>**echo "Anotherline">>test.txt**

You should seethe line appear in the console consumeroutputand in thesink file.

[*Step 8:UseKafka Streams to process data*](https://kafka.apache.org/documentation/#quickstart_kafkastreams)

KafkaStreamsisaclientlibrary ofKafkaforreal-timestreamprocessingandanalyzingdata storedinKafka brokers.Thisquickstartexamplewilldemonstratehowtorunastreaming applicationcodedinthislibrary.Hereisthegistofthe [WordCountDemoe](https://github.com/apache/kafka/blob/%7BdotVersion%7D/streams/examples/src/main/java/org/apache/kafka/streams/examples/wordcount/WordCountDemo.java)xamplecode (converted to useJava8 lambda expressions foreasyreading).

// Serializers/deserializers (serde) for StringandLongtypes final Serde<String>stringSerde=Serdes.String();

final Serde<Long>longSerde=Serdes.Long();

//Construct a `KStream`from theinputtopic""streams-file-input", where messagevalues

APACHEKAFKA 12

//represent lines oftext (forthe sakeof this example, weignore whatevermaybestored

//in themessagekeys).

KStream<String,String>textLines=builder.stream(stringSerde,stringSerde,"streams-file- input");

KTable<String,Long>wordCounts =textLines

// Split each textline, bywhitespace, into words.

.flatMapValues(value->Arrays.asList(value.toLowerCase().split("\\W+")))

//Group the text words as messagekeys

.groupBy((key, value)->value)

//Count theoccurrences of each word(messagekey).

.count("Counts")

// Storethe runningcounts as a changelogstream to the output topic. wordCounts.to(stringSerde, longSerde,"streams-wordcount-output");

Itimplementsthe WordCountalgorithm,whichcomputesa wordoccurrence histogramfrom theinputtext.However,unlikeother WordCountexamplesyoumighthaveseenbefore that operateonboundeddata,theWordCountdemoapplicationbehavesslightlydifferentlybecause itisdesignedtooperateonan **infinite,unboundedstream** ofdata.Similartothebounded variant,itisastatefulalgorithmthattracksandupdatesthecountsofwords.However,sinceit mustassumepotentially unboundedinputdata,itwillperiodically outputitscurrentstateand resultswhilecontinuingtoprocessmore databecause itcannotknowwhenithasprocessed "all"the input data.

Asthefirststep,wewillprepareinputdatatoaKafkatopic,whichwillsubsequently be processed byaKafkaStreams application.

>**echo -e "allstreamsleadto kafka\nhello kafkastreams\njoinkafkasummit"> file- input.txt**

Oron Windows:

APACHEKAFKA 13

>**echo allstreams lead to kafka>file-input.txt**

>**echo hello kafka streams>>file-input.txt**

>**echo|set /p=join kafkasummit>>file-input.txt**

Next,wesendthisinputdatatotheinputtopicnamed**streams-file-input** usingtheconsole producer, which reads thedata from STDIN line-by-line, and publishes each line as aseparate Kafka messagewithnullkey andvalueencodedastring tothetopic(inpractice,streamdata willlikelybeflowingcontinuouslyinto Kafkawheretheapplication willbeup and running):

>**bin/kafka-topics.sh--create\**

**--zookeeperlocalhost:2181\**

**--replication-factor1\**

**--partitions 1\**

**--topicstreams-file-input**

>**bin/kafka-console-producer.sh--broker-listlocalhost:9092--topicstreams-file-input<**

**file-input.txt**

We can now run the WordCount demo application to process theinputdata:

>**bin/kafka-run-class.sh org.apache.kafka.streams.examples.wordcount.WordCountDemo**

The demo application will read from the input topic **streams-file-input**, perform the computationsof the WordCountalgorithmoneachof thereadmessages,andcontinuously writeitscurrentresultstotheoutputtopic **streams-wordcount-output**.Hencetherewon'tbe anySTDOUToutputexceptlogentriesastheresultsarewrittenbackintoinKafka.Thedemo willrunforafewsecondsandthen,unliketypicalstreamprocessing applications,terminate automatically.

Wecannowinspectthe outputoftheWordCountdemoapplicationbyreadingfromitsoutput topic:

>**bin/kafka-console-consumer.sh--bootstrap-serverlocalhost:9092\**

**--topicstreams-wordcount-output \**

**--from-beginning\**

APACHEKAFKA 14

**--formatterkafka.tools.DefaultMessageFormatter\**

**--property print.key=true\**

**--property print.value=true\**

**--property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer\**

**--property**

**value.deserializer=org.apache.kafka.common.serialization.LongDeserializer**

with thefollowingoutputdata beingprinted to theconsole:

all 1 lead 1 to 1 hello 1

streams 2 join 1 kafka 3

summit 1

Here,thefirstcolumnistheKafkamessagekeyinjava.lang.Stringformat,andthesecond columnisthemessagevaluein java.lang.Longformat.Notethattheoutputisactuallya continuousstreamof updates,whereeachdata record(i.e.eachline inthe originaloutput above)isanupdatedcountofasingleword,akarecordkey suchas"kafka".Formultiple recordswith thesame key,each later record is anupdate ofthe previous one.

Thetwodiagramsbelowillustratewhatisessentially happeningbehindthescenes.Thefirst columnshowstheevolutionofthecurrentstateoftheKTable<String,Long> thatiscounting wordoccurrencesforcount.Thesecondcolumnshowsthechangerecordsthatresultfromstate updatestotheKTableandthatarebeingsenttotheoutputKafkatopic**streams-wordcount-**

**output**.

APACHEKAFKA 15

APACHEKAFKA 16

.

Whenthesecondtextline “hellokafka streams”isprocessed,we observe,for thefirsttime, thatexistingentriesinthe KTablearebeingupdated(here:forthewords“kafka”andfor “streams”). And again, changerecords arebeingsent to the output topic.

Andsoon(weskiptheillustrationofhowthethirdlineisbeingprocessed).Thisexplainswhy the outputtopic hasthecontentsweshowedabove,because itcontainsthe full recordof changes.

Looking beyondthescopeofthisconcreteexample,whatKafka Streamsisdoinghereisto leveragetheduality betweenatableandachangelogstream(here:table=theKTable, changelog stream=the downstreamKStream):youcanpublishevery changeofthetabletoa stream,andifyouconsumetheentire changelog streamfrombeginning toend,youcan reconstruct thecontentsof thetable.

Nowyoucanwritemoreinputmessagestothe**streams-file-input** topicandobserveadditional messagesaddedto**streams-wordcount-output** topic,reflectingupdatedwordcounts(e.g., usingthe consoleproducer and theconsole consumer, as described above).

You can stop the consoleconsumer via **Ctrl-C**.

[1.4 Ecosystem](https://kafka.apache.org/documentation/#ecosystem)

There are a plethora of tools that integrate with Kafka outside the main distribution. The [ecosystem pagel](https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem)ists many of these, including stream processing systems, Hadoop

integration, monitoring, and deployment tools.

APACHEKAFKA 17

 6.TECHINICAL IMPLICATION

[*Log Compaction Basics*](https://kafka.apache.org/documentation/#design_compactionbasics)

Here isa high-levelpicture thatshowsthelogicalstructure ofa Kafka logwiththeoffsetfor each message.

Theheadofthelog isidenticaltoatraditional Kafka log.Ithasdense,sequentialoffsetsand retainsallmessages.Logcompactionaddsanoptionforhandlingthetailofthelog.Thepicture aboveshowsalogwithacompactedtail.Notethatthemessagesinthetailofthelogretainthe originaloffsetassignedwhenthey werefirstwritten—thatneverchanges.Notealsothatall offsetsremainvalidpositionsinthelog,evenifthemessagewiththatoffsethasbeen

compactedaway;inthiscasethispositionisindistinguishablefromthenexthighestoffsetthat

APACHEKAFKA

 18

doesappearinthelog.Forexample,inthepicture abovetheoffsets36,37,and38areall equivalentpositionsandareadbeginning atany oftheseoffsetswouldreturnamessageset beginningwith 38.

Compactionalsoallowsfordeletes.Amessagewithakeyandanullpayloadwillbetreated asadeletefromthelog.Thisdeletemarkerwillcauseanypriormessagewiththatkey tobe removed(aswouldanynewmessagewiththatkey),butdeletemarkersarespecialinthatthey willthemselvesbecleanedoutofthelog afteraperiodoftimetofreeupspace.Thepointin timeatwhichdeletesarenolongerretainedismarkedasthe"deleteretentionpoint" in the abovediagram.

Thecompactionisdoneinthebackgroundby periodicallyrecopyinglogsegments.Cleaning doesnotblockreadsandcanbethrottledtousenomore thanaconfigurableamountofI/O throughputtoavoidimpactingproducersandconsumers.Theactualprocessofcompactinga

logsegment looks somethinglikethis:

APACHEKAFKA 19

APACHEKAFKA 20

**7.FUTURE ENHANCEMENTS**

 Redundancyinthephysical/hardware/networklayout:trynottoputthemallinthesame rack,decent(butdon'tgonuts)hardware,try tokeepredundantpowerandnetwork paths,etc.AtypicalZooKeeper ensemble has5or 7servers,whichtolerates2and3 serversdown, respectively.Ifyouhave asmalldeployment,thenusing3serversis acceptable,butkeepinmindthatyou'llonly beabletotolerate1serverdowninthis case.

 I/Osegregation:ifyoudoalotofwritetypetrafficyou'llalmostdefinitelywantthe transactionlogsonadedicated diskgroup.Writestothe transactionlogare synchronous(butbatchedfor performance), andconsequently,concurrentwritescan significantly affectperformance.ZooKeepersnapshotscanbeonesuchasourceof concurrentwrites,andideally shouldbewrittenonadiskgroupseparatefromthe transactionlog.Snapshotsarewrittentodiskasynchronously,soitistypically okto sharewiththe operatingsystemandmessage logfiles.Youcanconfigurea serverto useaseparate diskgroupwith thedataLogDir parameter.

 Applicationsegregation:Unless you reallyunderstandtheapplicationpatternsofother appsthatyouwanttoinstallonthe same box,itcanbe a goodidea torunZooKeeper in isolation (though this can beabalancingact with the capabilities ofthe hardware).

 Use care with virtualization: It can work, depending on your cluster layout and read/writepatternsandSLAs,butthetiny overheadsintroducedbythevirtualization layer can add up and throw offZooKeeper, as itcan beverytime sensitive

 ZooKeeperconfiguration:It'sjava,makesureyougiveit'enough'heapspace(We usually runthemwith3-5G,butthat'smostly duetothedatasetsizewehavehere). Unfortunately wedon't haveagoodformulaforit,butkeepinmindthatallowing for moreZooKeeper statemeansthatsnapshotscanbecome large,andlargesnapshots affectrecoverytime.Infact,ifthesnapshotbecomestoolarge(afewgigabytes),then youmay needtoincreasetheinitLimitparametertogiveenoughtimeforserversto recover and join the ensemble.

 Monitoring:BothJMXandthe4letterwords(4lw)commandsareveryuseful,theydo overlapinsomecases(andinthosecaseswepreferthe4lettercommands,they seem more predictable, or at the very least, they work better with the LI monitoring

infrastructure)

APACHEKAFKA 21

Don'toverbuildthecluster:largeclusters,especiallyinawriteheavyusagepattern,

meansalotofintracluster communication(quorumsonthewritesandsubsequent cluster

memberupdates),butdon'tunderbuildit(andriskswamping thecluster). Havingmore

serversadds toyour readcapacity.

 Overall,wetry tokeeptheZooKeepersystemassmallaswillhandletheload(plusstandard growth capacityplanning) andas simple as possible. Wetrynot to doanythingfancywith the configurationor applicationlayoutas comparedto theofficialreleaseas wellaskeep itas self containedaspossible.Forthesereasons,wetendtoskiptheOSpackagedversions,sinceithas atendencytotrytoputthingsintheOSstandardhierarchy,whichcan be'messy',forwantof

abetter wayto word it.

APACHEKAFKA 22

[**8.SECURITY**](https://kafka.apache.org/documentation/#security)

 [SecurityOverview](https://kafka.apache.org/documentation/#security_overview)

Inrelease0.9.0.0,theKafkacommunityaddedanumberoffeaturesthat,usedeitherseparately ortogether,increasessecurityinaKafkacluster.Thefollowingsecuritymeasuresarecurrently supported:

1. Authenticationofconnectionstobrokersfromclients(producersandconsumers),other brokersandtools,usingeitherSSL orSASL.KafkasupportsthefollowingSASL mechanisms:

oSASL/GSSAPI(Kerberos)-startingat version 0.9.0.0

oSASL/PLAIN-startingat version 0.10.0.0

oSASL/SCRAM-SHA-256andSASL/SCRAM-SHA-512-startingatversion

0.10.2.0

2. Authentication of connections from brokers toZooKeeper

3. Encryption of data transferred between brokers and clients, between brokers, or between brokersandtoolsusing SSL (Notethatthereisaperformancedegradation whenSSL isenabled,themagnitudeofwhichdependsontheCPUtypeandthe JVM implementation.)

4. Authorization ofread / writeoperations byclients

5. Authorization is pluggableand integration with external authorization services is supported

It'sworthnotingthatsecurityisoptional -non-secured clustersaresupported,aswellasamix ofauthenticated,unauthenticated,encryptedandnon-encryptedclients.Theguidesbelow

explain how to configureand usethe securityfeatures in both clients and brokers.

**9.CONCLUSION**

As you can see, Kafka has a unique design that makes it very useful for solving a wide range of architectural challenges. It is important to make sure you use the right approach for your use case and use it correctly to ensure high throughput, low latency, high availability, and no loss of data.

Apache Kafka is creating a lot of buzz these days. While LinkedIn, where Kafka was founded, is the most well known user, there are many companies successfully using this technology.So now that the word is out, it seems the world wants to know: What does it do? Why does everyone want to use it? How is it better than existing solutions? Do the benefits justify replacing existing systems and infrastructure? 23

Apache Kafka is publish-subscribe based fault tolerant messaging system. It is fast, scalable and distributed by design. This tutorial will explore the principles of Kafka, installation, operations and then it will walk you through with the deployment of Kafka cluster. Finally, we will conclude with real-time applica-tions and integration with Big Data Technologies In Big Data, an enormous volume of data is used. Regarding data, we have two main challenges.The first challenge is how to collect large volume of data and the second challenge is to analyze the collected data. To overcome those challenges, you must need a messaging system.Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.In summary, Kafka is a distributed publish-subscribe messaging system that is designed to be fast, scalable, and durable.

 **24**

**REFERENCES**

1. ["Mirror of ApacheKafka at GitHub]".](https://github.com/apache/kafka) github.com. .

2. ["Open-sourcing Kafka, LinkedIn's distributed message queue".](https://blog.linkedin.com/2011/01/11/open-source-linkedin-kafka)Retrieved 27

October 2016.

3. [Monitoring Kafkaperformancemetrics,D](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics)atadog Engineering Blog,accessed23May

2016/

[4. TheLog:Whateverysoftwareengineershouldknowaboutreal-timedata'sunifyingabstraction,L](http://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying)inkedIn Engineering Blog,accessed5 May2014

5. Primack, Dan. ["LinkedIn engineers spin out to launch 'Kafka' startup](http://fortune.com/2014/11/06/linkedin-kafka-confluent/)

[Confluent".](http://fortune.com/2014/11/06/linkedin-kafka-confluent/)fortune.com. Retrieved 10 February 2015.

6. ["Exchange Market DataStreaming with Kafka".](http://betsandbits.com/2015/07/22/exchange-market-data-streaming-with-kafka/)

7. ["OpenSOC: An Open Commitment to Security".](http://blogs.cisco.com/security/opensoc-an-open-commitment-to-security) Cisco blog. Retrieved2016-02-03.

8. ["Moredata, moredata".](https://blog.cloudflare.com/more-data-more-data/)

9. [**^** Do](https://en.wikipedia.org/wiki/Apache_Kafka#cite_ref-9)yung Yoon.["S2Graph : ALarge-Scale Graph Databasewith HBase".](http://apachebigdata2015.sched.org/event/de6abfbd8f0b9e66b1c03feb2b9e2078?iframe=yes&w=i:100;&sidebar=yes&bg=no)

10.[**^**](https://en.wikipedia.org/wiki/Apache_Kafka#cite_ref-10) ["Kafka Usagein EbayCommunications DeliveryPipeline".](https://www.youtube.com/watch?v=Vh2Rcwtz8RY)

25