

# KAFKA ARCHITECTURE: LOW LEVEL

May 18, 2017



Share

Tweet

Like 49

Share

If you are not sure what Kafka is, see What is Kafka? (<http://cloudurable.com/blog/what-is-kafka/index.html>).

## Kafka Architecture: Low-Level Design

This post really picks off from our series on Kafka architecture (<http://cloudurable.com/blog/kafka-architecture/index.html>) which includes Kafka topics architecture (<http://cloudurable.com/blog/kafka-architecture-topics/index.html>), Kafka producer architecture (<http://cloudurable.com/blog/kafka-architecture-producers/index.html>), Kafka consumer architecture (<http://cloudurable.com/blog/kafka-architecture-consumers/index.html>) and Kafka ecosystem architecture (<http://cloudurable.com/blog/kafka-architecture/index.html>).

This article is heavily inspired by the Kafka section on design (<https://kafka.apache.org/documentation/#design>). You can think of it as the cliff notes.

---

## Kafka Design Motivation

LinkedIn engineering built Kafka to support real-time analytics. Kafka was designed to feed analytics system that did real-time processing of streams. LinkedIn developed Kafka as a unified platform for real-time handling of streaming data feeds. The goal behind Kafka, build a high-throughput streaming data platform that supports high-volume event streams like log aggregation, user activity, etc.

To scale to meet the demands of LinkedIn Kafka is distributed, supports sharding and load balancing. Scaling needs inspired Kafka's partitioning and consumer model. Kafka scales writes and reads with partitioned, distributed, commit logs. Kafka's sharding is called partitioning. (Kinesis which is similar to Kafka (<http://cloudurable.com/blog/kinesis-vs-kafka/index.html>) calls partitions shards.)

A database shard is a horizontal partition of data in a database or search engine. Each individual partition is referred to as a shard or database shard. Each shard is held on a separate database server instance, to spread load. Sharding ([https://en.wikipedia.org/wiki/Shard\\_\(database\\_architecture\)](https://en.wikipedia.org/wiki/Shard_(database_architecture)))

Kafka was designed to handle periodic large data loads from offline systems as well as traditional messaging use-cases, low-latency. MOM is message oriented middleware think IBM MQSeries, JMS (<http://cloudurable.com/blog/kafka-vs-jms/index.html>), ActiveMQ, and RabbitMQ. Like many MOMs, Kafka is fault-tolerance for node failures through replication and leadership election. However, the design of Kafka is more like a distributed database transaction log than a traditional messaging system. Unlike many MOMs, Kafka replication was built into the low-level design and is not an afterthought.

## Persistence: Embrace filesystem

Kafka relies on the filesystem for storing and caching records.

The disk performance of hard drives performance of sequential writes is fast (<https://mechanical-sympathy.blogspot.com/2011/12/java-sequential-io-performance.html>) (really fast (<http://rick-hightower.blogspot.com/2013/11/fastest-java-io-circa-2013-writing-large.html>)). JBOD is just a bunch of disk drives. JBOD configuration with six 7200rpm SATA RAID-5 array is about 600MB/sec. Like Cassandra tables, Kafka logs are write only structures, meaning, data gets appended to the end of the log. When using HDD, sequential reads and writes are fast, predictable, and heavily optimized by operating systems. Using HDD, sequential disk access can be faster than random memory access and SSD.

While JVM GC overhead can be high, Kafka leans on the OS a lot for caching, which is big, fast and rock solid cache. Also, modern operating systems use all available main memory for disk caching. OS file caches are almost free and don't have the overhead of the OS. Implementing cache coherency is challenging to get right, but Kafka relies on the rock solid OS for cache coherence. Using the OS for cache also reduces the number of buffer copies. Since Kafka disk usage tends to do sequential reads, the OS read-ahead cache is impressive.

Cassandra, Netty, and Varnish use similar techniques. All of this is explained well in the Kafka Documentation (<https://kafka.apache.org/documentation/#persistence>). And, there is a more entertaining explanation at the Varnish site (<http://www.varnish-cache.org/trac/wiki/ArchitectNotes>).

Cloudurable provides Kafka training (<http://cloudurable.com/kafka-training/index.html>), Kafka consulting (<http://cloudurable.com/kafka-aws-consulting/index.html>), Kafka support ([http://cloudurable.com/subscription\\_support/index.html](http://cloudurable.com/subscription_support/index.html)) and helps setting up Kafka clusters in AWS (<http://cloudurable.com/services/index.html>).

## Big fast HDDs and long sequential access

Kafka favors long sequential disk access for reads and writes. Like Cassandra, LevelDB, RocksDB, and others Kafka uses a form of log structured storage and compaction instead of an on-disk mutable BTree. Like Cassandra, Kafka uses tombstones instead of deleting records right away.

Since disks these days have somewhat unlimited space and are very fast, Kafka can provide features not usually found in a messaging system like holding on to old messages for a long time. This flexibility allows for interesting applications of Kafka.

---

## Kafka Producer Load Balancing

The producer asks the Kafka broker for metadata about which Kafka broker has which topic partitions leaders thus no routing layer needed. This leadership data allows the producer to send records directly to Kafka broker partition leader.

The Producer client controls which partition it publishes messages to, and can pick a partition based on some application logic. Producers can partition records by key, round-robin or use a custom application-specific partitioner logic.

---

## Kafka Producer Record Batching

Kafka producers support record batching. Batching can be configured by the size of records in bytes in batch. Batches can be auto-flushed based on time.

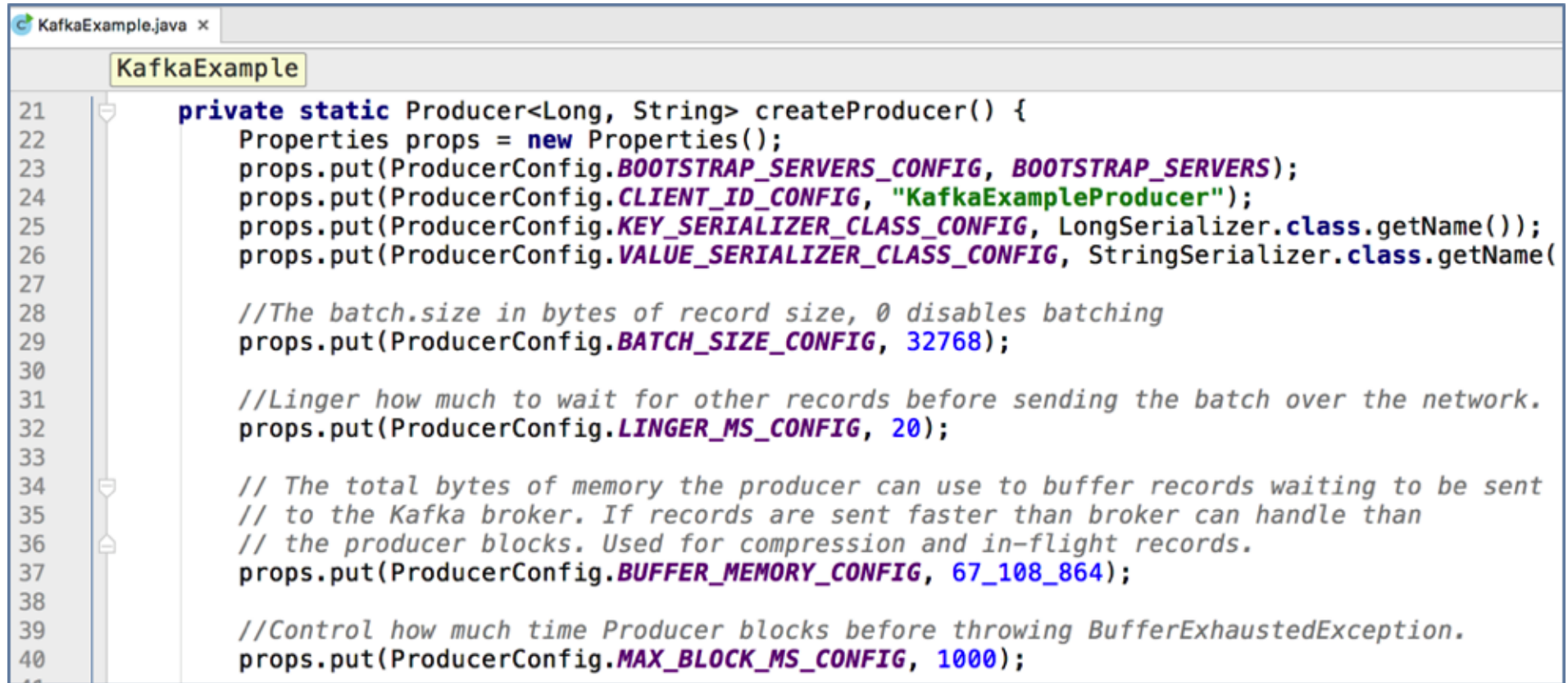
Batching is good for network IO throughput.

Batching speeds up throughput drastically.

Buffering is configurable and lets you make a tradeoff between additional latency for better throughput. Or in the case of a heavily used system, it could be both better average throughput and reduces overall latency.

Batching allows accumulation of more bytes to send, which equate to few larger I/O operations on Kafka Brokers and increase compression efficiency. For higher throughput, Kafka Producer configuration allows buffering based on time and size. The producer sends multiple records as a batch with fewer network requests than sending each record one by one.

## Kafka Producer Batching



```
KafkaExample.java x
KafkaExample

21 private static Producer<Long, String> createProducer() {
22     Properties props = new Properties();
23     props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOTSTRAP_SERVERS);
24     props.put(ProducerConfig.CLIENT_ID_CONFIG, "KafkaExampleProducer");
25     props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, LongSerializer.class.getName());
26     props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
27
28     //The batch.size in bytes of record size, 0 disables batching
29     props.put(ProducerConfig.BATCH_SIZE_CONFIG, 32768);
30
31     //Linger how much to wait for other records before sending the batch over the network.
32     props.put(ProducerConfig.LINGER_MS_CONFIG, 20);
33
34     // The total bytes of memory the producer can use to buffer records waiting to be sent
35     // to the Kafka broker. If records are sent faster than broker can handle then
36     // the producer blocks. Used for compression and in-flight records.
37     props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 67_108_864);
38
39     //Control how much time Producer blocks before throwing BufferExhaustedException.
40     props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 1000);
41 }
```

## Kafka compression

In large streaming platforms, the bottleneck is not always CPU or disk but often network bandwidth. There is even more network bandwidth issues in cloud, containerized and virtualized environments as multiple services could be sharing a NiC card. Also, network bandwidth issues can be problematic when talking datacenter to datacenter or WAN.

Batching is beneficial for efficient compression and network IO throughput.

Kafka provides end-to-end batch compression instead of compressing a record at a time, Kafka efficiently compresses a whole batch of records. The same message batch can be compressed and sent to Kafka broker/server in one go and written in compressed form into the log partition. You can even configure the compression so that no decompression happens until the Kafka broker delivers the compressed records to the consumer.

Kafka supports GZIP, Snappy and LZ4 compression protocols.

# Pull vs. Push/Streams

With Kafka consumers pull data from brokers. Other systems brokers push data or stream data to consumers. Messaging is usually a pull-based system (SQS, most MOM use pull). With the pull-based system, if a consumer falls behind, it catches up later when it can.

Since Kafka is pull-based, it implements aggressive batching of data. Kafka like many pull based systems implements a long poll (SQS, Kafka both do). A long poll keeps a connection open after a request for a period and waits for a response.

A pull-based system has to pull data and then process it, and there is always a pause between the pull and getting the data.

Push based push data to consumers (scribe, flume, reactive streams, RxJava, Akka). Push-based or streaming systems have problems dealing with slow or dead consumers. It is possible for a push system consumer to get overwhelmed when its rate of consumption falls below the rate of production. Some push-based systems use a back-off protocol based on back pressure that allows a consumer to indicate it is overwhelmed see reactive streams (<http://www.reactive-streams.org/>). This problem of not flooding a consumer and consumer recovery, are tricky when trying to track message acknowledgments.

Push-based or streaming systems can send a request immediately or accumulate requests and send in batches (or a combination based on back pressure). Push-based systems are always pushing data. The consumer can accumulate messages while it is processing data already sent which is an advantage to reduce the latency of message processing. However, if the consumer died when it was behind processing, how does the broker know where the consumer was and when does data get sent again to another Consumer. This problem is not an easy problem to solve. Kafka gets around these complexities by using a pull-based system.

---

## Traditional MOM Consumer Message State Tracking

With most MOM it is the broker's responsibility to keep track of which messages gets marked consumed. Message tracking is not an easy task. As consumer consumes messages, the broker keeps track of the state.

The goal in most MOM systems is for the broker to delete data quickly after consumption. Remember most MOMs were written when disks were a lot smaller, less capable, and more expensive.

This message tracking is trickier than it sounds (acknowledgment feature), as brokers must maintain lots of states to track per message, sent, acknowledge, and know when to delete or resend the message.

## Kafka Consumer Message State Tracking

Remember that Kafka topics get divided into ordered partitions. Each message has an offset in this ordered partition. Each topic partition is consumed by exactly one consumer per consumer group at a time.

This partition layout means, the Broker tracks the offset data not tracked per message like MOM, but only needs the offset of each consumer group, partition offset pair stored. This offset tracking equates to a lot fewer data to track.

The consumer sends location data periodically (consumer group, partition offset pair) to the Kafka broker, and the broker stores this offset data into an offset topic.

The offset style message acknowledgment is much cheaper compared to MOM. Also, consumers are more flexible and can rewind to an earlier offset (replay). If there was a bug, then fix the bug, rewind consumer and replay the topic. This rewind feature is a killer feature of Kafka as Kafka can hold topic log data for a very long time.

---

## Message Delivery Semantics

There are three message delivery semantics: at most once, at least once and exactly once. At most once is messages may be lost but are never redelivered. At least once is messages are never lost but may be redelivered. Exactly once is each message is delivered once and only once. Exactly once is preferred but more expensive, and requires more bookkeeping for the producer and consumer.

## Kafka Consumer and Message Delivery Semantics

Recall that all replicas have exactly the same log partitions with the same offsets and the consumer groups maintain its position in the log per topic partition.

To implement “at-most-once” consumer reads a message, then saves its offset in the partition by sending it to the broker, and finally process the message. The issue with “at-most-once” is a consumer could die after saving its position but before processing the message. Then the consumer that takes over or gets restarted would leave off at the last position and message in question is never processed.

To implement “at-least-once” the consumer reads a message, process messages, and finally saves offset to the broker. The issue with “at-least-once” is a consumer could crash after processing a message but before saving last offset position. Then if the consumer is restarted or another consumer takes over, the consumer could receive the message that was already processed. The “at-least-once” is the most common set up for messaging, and it is your responsibility to make the messages idempotent, which means getting the same message twice will not cause a problem (two debits).

To implement “exactly once” on the consumer side, the consumer would need a two-phase commit between storage for the consumer position, and storage of the consumer’s message process output. Or, the consumer could store the message process output in the same location as the last offset.

Kafka offers the first two, and it up to you to implement the third from the consumer perspective.

# Kafka Producer Durability and Acknowledgement

Kafka's offers operational predictability semantics for durability. When publishing a message, a message gets "committed" to the log which means all ISRs accepted the message. This commit strategy works out well for durability as long as at least one replica lives.

The producer connection could go down in middle of send, and producer may not be sure if a message it sent went through, and then the producer resends the message. This resend-logic is why it is important to use message keys and use idempotent messages (duplicates ok). Kafka did not make guarantees of messages not getting duplicated from producer retrying until recently (June 2017).

The producer can resend a message until it receives confirmation, i.e., acknowledgment received.

The producer resending the message without knowing if the other message it sent made it or not, negates "exactly once" and "at-most-once" message delivery semantics.

## Producer Durability

The producer can specify durability level. The producer can wait on a message being committed. Waiting for commit ensures all replicas have a copy of the message.

The producer can send with no acknowledgments (0). The producer can send with just get one acknowledgment from the partition leader (1). The producer can send and wait on acknowledgments from all replicas (-1), which is the default.

As of June 2017: the producer can ensure a message or group of messages was sent "exactly once".

## Improved Producer (June 2017 release)

Kafka now supports "exactly once" delivery from producer (<https://www.slideshare.net/apurva2/introducing-exactly-once-semantics-to-apache-kafka>), performance improvements and atomic write across partitions. They achieve this by the producer sending a sequence id, the broker keeps track if producer already sent this sequence, if producer tries to send it again, it gets an ack for duplicate message, but nothing is saved to log. This improvement requires no API change.

## Kafka Producer Atomic Log Writes (June 2017 Release)

Another improvement to Kafka is the Kafka producers having atomic write across partitions. The atomic writes mean Kafka consumers can only see committed logs (configurable). Kafka has a coordinator that writes a marker to the topic log to signify what has been successfully transacted. The transaction coordinator and transaction log maintain the state of the atomic writes.

The atomic writes does require a new producer API for transactions.

Here is an example of using the new producer API.

## New Producer API for transactions

```
producer.initTransaction();

try {
    producer.beginTransaction();
    producer.send(debitAccountMessage);
    producer.send(creditOtherAccountMessage);
    producer.sendOffsetsToTxn(...);
    producer.commitTransaction();
} catch (ProducerFencedTransactionException pfte) {
    ...
    producer.close();
} catch (KafkaException ke) {
    ...
    producer.abortTransaction();
}
```

## Kafka Replication

Kafka replicates each topic's partitions across a configurable number of Kafka brokers. Kafka's replication model is by default, not a bolt-on feature like most MOMs as Kafka was meant to work with partitions and multi-nodes from the start. Each topic partition has one leader and zero or more followers.

Leaders and followers are called replicas. A replication factor is the leader node plus all of the followers. Partition leadership is evenly shared among Kafka brokers. Consumers only read from the leader. Producers only write to the leaders.

The topic log partitions on followers are in-sync to leader's log, ISRs are an exact copy of the leaders minus the to-be-replicated records that are in-flight. Followers pull records in batches from their leader like a regular Kafka consumer.

## Kafka Broker Failover

Kafka keeps track of which Kafka brokers are alive. To be alive, a Kafka Broker must maintain a ZooKeeper session using ZooKeeper's heartbeat mechanism, and must have all of its followers in-sync with the leaders and not fall too far behind.



Both the ZooKeeper session and being in-sync is needed for broker liveness which is referred to as being in-sync. An in-sync replica is called an ISR. Each leader keeps track of a set of “in sync replicas”.

If ISR/follower dies, falls behind, then the leader will remove the follower from the set of ISRs. Falling behind is when a replica is not in-sync after `replica.lag.time.max.ms` period.

A message is considered “committed” when all ISRs have applied the message to their log. Consumers only see committed messages. Kafka guarantee: committed message will not be lost, as long as there is at least one ISR.

## Replicated Log Partitions

A Kafka partition is a replicated log. A replicated log is a distributed data system primitive. A replicated log is useful for implementing other distributed systems using state machines. A replicated log models “coming into consensus” on an ordered series of values.

While a leader stays alive, all followers just need to copy values and ordering from their leader. If the leader does die, Kafka chooses a new leader from its followers which are in-sync. If a producer is told a message is committed, and then the leader fails, then the newly elected leader must have that committed message.

The more ISRs you have; the more there are to elect during a leadership failure.

---

## Kafka and Quorum

Quorum is the number of acknowledgments required and the number of logs that must be compared to elect a leader such that there is guaranteed to be an overlap for availability. Most systems use a majority vote, Kafka does not use a simple majority vote to improve availability.

In Kafka, leaders are selected based on having a complete log. If we have a replication factor of 3, then at least two ISRs must be in-sync before the leader declares a sent message committed. If a new leader needs to be elected then, with no more than 3 failures, the new leader is guaranteed to have all committed messages.

Among the followers there must be at least one replica that contains all committed messages. Problem with majority vote Quorum is it does not take many failures to have inoperable cluster.

## Kafka Quorum Majority of ISRs

Kafka maintains a set of ISRs per leader. Only members in this set of ISRs are eligible for leadership election. What the producer writes to partition is not committed until all ISRs acknowledge the write. ISRs are persisted to ZooKeeper whenever ISR set changes. Only replicas that are members of ISR set are eligible to be elected leader.

This style of ISR quorum allows producers to keep working without the majority of all nodes, but only an ISR majority vote. This style of ISR quorum also allows a replica to rejoin ISR set and have its vote count, but it has to be fully re-synced before joining even if replica lost un-flushed data during its crash.

## All nodes die at same time. Now what?

Kafka's guarantee about data loss is only valid if at least one replica is in-sync.

If all followers that are replicating a partition leader die at once, then data loss Kafka guarantee is not valid. If all replicas are down for a partition, Kafka, by default, chooses first replica (not necessarily in ISR set) that comes alive as the leader (config `unclean.leader.election.enable=true` is default). This choice favors availability to consistency.

If consistency is more important than availability for your use case, then you can set config `unclean.leader.election.enable=false` then if all replicas are down for a partition, Kafka waits for the first ISR member (not first replica) that comes alive to elect a new leader.

---

## Producers pick Durability

Producers can choose durability by setting acks to - none (0), the leader only (1) or all replicas (-1 ).

The acks=all is the default. With all, the acks happen when all current in-sync replicas (ISRs) have received the message.

You can make the trade-off between consistency and availability. If durability over availability is preferred, then disable unclean leader election and specify a minimum ISR size.

The higher the minimum ISR size, the better the guarantee is for consistency. But the higher minimum ISR, the more you reduces availability since partition won't be unavailable for writes if the size of ISR set is less than the minimum threshold.

---

## Quotas

Kafka has quotas for consumers and producers to limits bandwidth they are allowed to consume. These quotas prevent consumers or producers from hogging up all the Kafka broker resources. The quota is by client id or user. The quota data is stored in ZooKeeper, so changes do not necessitate restarting Kafka brokers.

---

## Kafka Low-Level Design and Architecture Review

## How would you prevent a denial of service attack from a poorly written consumer?

Use Quotas to limit the consumer's bandwidth.

## What is the default producer durability (acks) level?

All. Which means all ISRs have to write the message to their log partition.

## What happens by default if all of the Kafka nodes go down at once?

Kafka chooses the first replica (not necessarily in ISR set) that comes alive as the leader as `unclean.leader.election.enable=true` is default to support availability.

## Why is Kafka record batching important?

Optimized IO throughput over the wire as well as to the disk. It also improves compression efficiency by compressing an entire batch.

## What are some of the design goals for Kafka?

To be a high-throughput, scalable streaming data platform for real-time analytics of high-volume event streams like log aggregation, user activity, etc.

## What are some of the new features in Kafka as of June 2017?

Producer atomic writes, performance improvements and producer not sending duplicate messages.

## What is the different message delivery semantics?

There are three message delivery semantics: at most once, at least once and exactly once.

## Related content

- What is Kafka? (<http://cloudurable.com/blog/what-is-kafka/index.html>)
- Kafka Architecture (<http://cloudurable.com/blog/kafka-architecture/index.html>)
- Kafka Topic Architecture (<http://cloudurable.com/blog/kafka-architecture-topics/index.html>)
- Kafka Consumer Architecture (<http://cloudurable.com/blog/kafka-architecture-consumers/index.html>)
- Kafka Producer Architecture (<http://cloudurable.com/blog/kafka-architecture-producers/index.html>)
- Kafka Architecture and low level design (<http://cloudurable.com/blog/kafka-architecture-low-level/index.html>)
- Kafka and Schema Registry (<http://cloudurable.com/blog/kafka-avro-schema-registry/index.html>)
- Kafka and Avro (<http://cloudurable.com/blog/avro/index.html>)
- Kafka Ecosystem (<http://cloudurable.com/blog/kafka-ecosystem/index.html>)

- Kafka vs. JMS (<http://cloudurable.com/blog/kafka-vs-jms/index.html>)
- Kafka versus Kinesis (<http://cloudurable.com/blog/kinesis-vs-kafka/index.html>)
- Kafka Tutorial: Using Kafka from the command line (<http://cloudurable.com/blog/kafka-tutorial-kafka-from-command-line/index.html>)
- Kafka Tutorial: Kafka Broker Failover and Consumer Failover (<http://cloudurable.com/blog/kafka-tutorial-kafka-failover-kafka-cluster/index.html>)
- Kafka Tutorial (<http://cloudurable.com/ppt/kafka-tutorial-cloudurable-v2.pdf>)
- Kafka Tutorial: Writing a Kafka Producer example in Java (<http://cloudurable.com/blog/kafka-tutorial-kafka-producer/index.html>)
- Kafka Tutorial: Writing a Kafka Consumer example in Java (<http://cloudurable.com/blog/kafka-tutorial-kafka-consumer/index.html>)
- Kafka Architecture: Log Compaction (<http://cloudurable.com/blog/kafka-architecture-log-compaction/index.html>)
- Kafka Architecture: Low-Level PDF Slides (<http://cloudurable.com/ppt/4-kafka-detailed-architecture.pdf>)

## About Cloudurable

We hope you enjoyed this article. Please provide feedback (<http://cloudurable.com/contact/index.html>). Cloudurable provides Kafka training (<http://cloudurable.com/kafka-training/index.html>), Kafka consulting (<http://cloudurable.com/kafka-aws-consulting/index.html>), Kafka support ([http://cloudurable.com/subscription\\_support/index.html](http://cloudurable.com/subscription_support/index.html)) and helps setting up Kafka clusters in AWS (<http://cloudurable.com/services/index.html>).

Check out our new GoLang course. We provide onsite Go Lang training which is instructor led (<http://cloudurable.com/golang-onsite-instructor-led-training/index.html>).



Share

Tweet

Like 49

Share

## SEARCH

Search



## SHARE

Tweet



Share

facebook

49

Like

Share

## FOLLOW

Follow @cloudurable

 Follow

64

facebook

## CATEGORIES

amazon-ebs (1) (<http://cloudurable.com/categories/amazon-ebs/index.html>)

amazon-ec2 (1) (<http://cloudurable.com/categories/amazon-ec2/index.html>)

amazon-vpc (1) (<http://cloudurable.com/categories/amazon-vpc/index.html>)

ansible (4) (<http://cloudurable.com/categories/ansible/index.html>)

avro (2) (<http://cloudurable.com/categories/avro/index.html>)

aws (4) (<http://cloudurable.com/categories/aws/index.html>)

aws-cassandra (6) (<http://cloudurable.com/categories/aws-cassandra/index.html>)

aws-command-line (1) (<http://cloudurable.com/categories/aws-command-line/index.html>)

cassandra (12) (<http://cloudurable.com/categories/cassandra/index.html>)

cassandra-aws (3) (<http://cloudurable.com/categories/cassandra-aws/index.html>)

cassandra-cluster (1) (<http://cloudurable.com/categories/cassandra-cluster/index.html>)

cassandra-database (2) (<http://cloudurable.com/categories/cassandra-database/index.html>)

cassandra-training (5) (<http://cloudurable.com/categories/cassandra-training/index.html>)

cassandra-tutorial (5) (<http://cloudurable.com/categories/cassandra-tutorial/index.html>)

cloud (4) (<http://cloudurable.com/categories/cloud/index.html>)

cloudformation (1) (<http://cloudurable.com/categories/cloudformation/index.html>)

cloudurable (15) (<http://cloudurable.com/categories/cloudurable/index.html>)

cluster (1) (<http://cloudurable.com/categories/cluster/index.html>)

devops (16) (<http://cloudurable.com/categories/devops/index.html>)

ebs (3) (<http://cloudurable.com/categories/ebs/index.html>)

ec2 (1) (<http://cloudurable.com/categories/ec2/index.html>)

kafka (13) (<http://cloudurable.com/categories/kafka/index.html>)

kafka-advanced-consumers (1) (<http://cloudurable.com/categories/kafka-advanced-consumers/index.html>)

kafka-architecture (13) (<http://cloudurable.com/categories/kafka-architecture/index.html>)

kafka-avro-serialization (1) (<http://cloudurable.com/categories/kafka-avro-serialization/index.html>)

kafka-consulting (2) (<http://cloudurable.com/categories/kafka-consulting/index.html>)

kafka-consumer (1) (<http://cloudurable.com/categories/kafka-consumer/index.html>)

kafka-consumers (1) (<http://cloudurable.com/categories/kafka-consumers/index.html>)

kafka-ecosystem (1) (<http://cloudurable.com/categories/kafka-ecosystem/index.html>)

kafka-schema-registry (1) (<http://cloudurable.com/categories/kafka-schema-registry/index.html>)

kafka-training (23) (<http://cloudurable.com/categories/kafka-training/index.html>)

kafka-tutorial (20) (<http://cloudurable.com/categories/kafka-tutorial/index.html>)

kaka-replication (1) (<http://cloudurable.com/categories/kaka-replication/index.html>)

kinesis (1) (<http://cloudurable.com/categories/kinesis/index.html>)

kinesis-consulting (1) (<http://cloudurable.com/categories/kinesis-consulting/index.html>)

linux (1) (<http://cloudurable.com/categories/linux/index.html>)

metricsd (1) (<http://cloudurable.com/categories/metricsd/index.html>)

microservices (3) (<http://cloudurable.com/categories/microservices/index.html>)

nodetool (1) (<http://cloudurable.com/categories/nodetool/index.html>)

schema-registry (1) (<http://cloudurable.com/categories/schema-registry/index.html>)

smack (3) (<http://cloudurable.com/categories/smack/index.html>)

spark (3) (<http://cloudurable.com/categories/spark/index.html>)

spark-cassandra (1) (<http://cloudurable.com/categories/spark-cassandra/index.html>)

spark-kafka (1) (<http://cloudurable.com/categories/spark-kafka/index.html>)

spark-training (3) (<http://cloudurable.com/categories/spark-training/index.html>)

spark-tutorial (3) (<http://cloudurable.com/categories/spark-tutorial/index.html>)

ssh (1) (<http://cloudurable.com/categories/ssh/index.html>)

ssh-config (1) (<http://cloudurable.com/categories/ssh-config/index.html>)

ssl (1) (<http://cloudurable.com/categories/ssl/index.html>)

systemd (1) (<http://cloudurable.com/categories/systemd/index.html>)

tls (1) (<http://cloudurable.com/categories/tls/index.html>)

vagrant (5) (<http://cloudurable.com/categories/vagrant/index.html>)

## TAGS

AKKA ([HTTP://CLOUDURABLE.COM/TAGS/AKKA/INDEX.HTML](http://cloudurable.com/tags/akka/index.html))

AKKA-CONSULTING ([HTTP://CLOUDURABLE.COM/TAGS/AKKA-CONSULTING/INDEX.HTML](http://cloudurable.com/tags/akka-consulting/index.html))

AMAZON-EBS ([HTTP://CLOUDURABLE.COM/TAGS/AMAZON-EBS/INDEX.HTML](http://cloudurable.com/tags/amazon-ebs/index.html))

AMAZON-EC2 ([HTTP://CLOUDURABLE.COM/TAGS/AMAZON-EC2/INDEX.HTML](http://cloudurable.com/tags/amazon-ec2/index.html))

AMI ([HTTP://CLOUDURABLE.COM/TAGS/AMI/INDEX.HTML](http://cloudurable.com/tags/ami/index.html))

ANSIBLE ([HTTP://CLOUDURABLE.COM/TAGS/ANSIBLE/INDEX.HTML](http://cloudurable.com/tags/ansible/index.html))

AVRO ([HTTP://CLOUDURABLE.COM/TAGS/AVRO/INDEX.HTML](http://cloudurable.com/tags/avro/index.html))

AVRO-KAFKA ([HTTP://CLOUDURABLE.COM/TAGS/AVRO-KAFKA/INDEX.HTML](http://cloudurable.com/tags/avro-kafka/index.html))

AWS ([HTTP://CLOUDURABLE.COM/TAGS/AWS/INDEX.HTML](http://cloudurable.com/tags/aws/index.html))

AWS-CASSANDRA ([HTTP://CLOUDURABLE.COM/TAGS/AWS-CASSANDRA/INDEX.HTML](http://cloudurable.com/tags/aws-cassandra/index.html))

AWS-COMMAND-LINE ([HTTP://CLOUDURABLE.COM/TAGS/AWS-COMMAND-LINE/INDEX.HTML](http://cloudurable.com/tags/aws-command-line/index.html))



🔗 [CASSANDRA \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA/INDEX.HTML\)](http://cloudurable.com/tags/cassandra/index.html)

🔗 [CASSANDRA-ARCHITECTURE \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-ARCHITECTURE/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-architecture/index.html)

🔗 [CASSANDRA-AWS \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-AWS/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-aws/index.html)

🔗 [CASSANDRA-CLOUD \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-CLOUD/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-cloud/index.html)

🔗 [CASSANDRA-CLUSTER \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-CLUSTER/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-cluster/index.html)

🔗 [CASSANDRA-DATABASE \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-DATABASE/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-database/index.html)

🔗 [CASSANDRA-DBA \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-DBA/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-dba/index.html)

🔗 [CASSANDRA-DEVOPS \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-DEVOPS/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-devops/index.html)

🔗 [CASSANDRA-OS-SYSTEM-MEMORY \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-OS-SYSTEM-MEMORY/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-os-system-memory/index.html)

🔗 [CASSANDRA-TRAINING \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-TRAINING/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-training/index.html)

🔗 [CASSANDRA-TUTORIAL \(HTTP://CLOUDURABLE.COM/TAGS/CASSANDRA-TUTORIAL/INDEX.HTML\)](http://cloudurable.com/tags/cassandra-tutorial/index.html)

🔗 [CLOUD \(HTTP://CLOUDURABLE.COM/TAGS/CLOUD/INDEX.HTML\)](http://cloudurable.com/tags/cloud/index.html)

🔗 [CLOUDFORMATION \(HTTP://CLOUDURABLE.COM/TAGS/CLOUDFORMATION/INDEX.HTML\)](http://cloudurable.com/tags/cloudformation/index.html)

🔗 [CLOUDFORMATION-TUTORIAL \(HTTP://CLOUDURABLE.COM/TAGS/CLOUDFORMATION-TUTORIAL/INDEX.HTML\)](http://cloudurable.com/tags/cloudformation-tutorial/index.html)

🔗 [CLOUDURABLE \(HTTP://CLOUDURABLE.COM/TAGS/CLOUDURABLE/INDEX.HTML\)](http://cloudurable.com/tags/cloudurable/index.html)

🔗 [CLUSTER \(HTTP://CLOUDURABLE.COM/TAGS/CLUSTER/INDEX.HTML\)](http://cloudurable.com/tags/cluster/index.html)

🔗 [COMPUTE \(HTTP://CLOUDURABLE.COM/TAGS/COMPUTE/INDEX.HTML\)](http://cloudurable.com/tags/compute/index.html)

🔗 [CONSUMERS \(HTTP://CLOUDURABLE.COM/TAGS/CONSUMERS/INDEX.HTML\)](http://cloudurable.com/tags/consumers/index.html)

🔗 [DBA \(HTTP://CLOUDURABLE.COM/TAGS/DBA/INDEX.HTML\)](http://cloudurable.com/tags/dba/index.html)

🔗 [DEVOPS \(HTTP://CLOUDURABLE.COM/TAGS/DEVOPS/INDEX.HTML\)](http://cloudurable.com/tags/devops/index.html)

🔗 [EBS \(HTTP://CLOUDURABLE.COM/TAGS/EBS/INDEX.HTML\)](http://cloudurable.com/tags/ebs/index.html)

🔗 [EC2 \(HTTP://CLOUDURABLE.COM/TAGS/EC2/INDEX.HTML\)](http://cloudurable.com/tags/ec2/index.html)

🔗 [EC2-INSTANCE-STORE \(HTTP://CLOUDURABLE.COM/TAGS/EC2-INSTANCE-STORE/INDEX.HTML\)](http://cloudurable.com/tags/ec2-instance-store/index.html)

🔗 [ECU \(HTTP://CLOUDURABLE.COM/TAGS/ECU/INDEX.HTML\)](http://cloudurable.com/tags/ecu/index.html)

🔗 [FAILOVER \(HTTP://CLOUDURABLE.COM/TAGS/FAILOVER/INDEX.HTML\)](http://cloudurable.com/tags/failover/index.html)

🔗 [IMAGES \(HTTP://CLOUDURABLE.COM/TAGS/IMAGES/INDEX.HTML\)](http://cloudurable.com/tags/images/index.html)

🔗 [INSTANCES \(HTTP://CLOUDURABLE.COM/TAGS/INSTANCES/INDEX.HTML\)](http://cloudurable.com/tags/instances/index.html)

🔗 [JMS \(HTTP://CLOUDURABLE.COM/TAGS/JMS/INDEX.HTML\)](http://cloudurable.com/tags/jms/index.html)

🔗 [KAFKA \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA/INDEX.HTML\)](http://cloudurable.com/tags/kafka/index.html)

🔗 [KAFKA-ADVANCED-CONSUMERS \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-ADVANCED-CONSUMERS/INDEX.HTML\)](http://cloudurable.com/tags/kafka-advanced-consumers/index.html)

🔗 [KAFKA-ADVANCED-PRODUCERS \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-ADVANCED-PRODUCERS/INDEX.HTML\)](http://cloudurable.com/tags/kafka-advanced-producers/index.html)

🔗 [KAFKA-ARCHITECTURE \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-ARCHITECTURE/INDEX.HTML\)](http://cloudurable.com/tags/kafka-architecture/index.html)

🔗 [KAFKA-AVRO \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-AVRO/INDEX.HTML\)](http://cloudurable.com/tags/kafka-avro/index.html)

🔗 [KAFKA-CONNECT \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-CONNECT/INDEX.HTML\)](http://cloudurable.com/tags/kafka-connect/index.html)

🔗 [KAFKA-CONSULTING \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-CONSULTING/INDEX.HTML\)](http://cloudurable.com/tags/kafka-consulting/index.html)

🔗 [KAFKA-CONSUMERS \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-CONSUMERS/INDEX.HTML\)](http://cloudurable.com/tags/kafka-consumers/index.html)

🔗 [KAFKA-CONSUMERS-ADVANCED \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-CONSUMERS-ADVANCED/INDEX.HTML\)](http://cloudurable.com/tags/kafka-consumers-advanced/index.html)

🔗 [KAFKA-ECOSYSTEM \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-ECOSYSTEM/INDEX.HTML\)](http://cloudurable.com/tags/kafka-ecosystem/index.html)

🔗 [KAFKA-LOG-COMPACTION \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-LOG-COMPACTION/INDEX.HTML\)](http://cloudurable.com/tags/kafka-log-compaction/index.html)

🔗 [KAFKA-REST-PROXY \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-REST-PROXY/INDEX.HTML\)](http://cloudurable.com/tags/kafka-rest-proxy/index.html)

🔗 [KAFKA-STREAMS \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-STREAMS/INDEX.HTML\)](http://cloudurable.com/tags/kafka-streams/index.html)

🔗 [KAFKA-TRAINING \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-TRAINING/INDEX.HTML\)](http://cloudurable.com/tags/kafka-training/index.html)

🔗 [KAFKA-TUTORIAL \(HTTP://CLOUDURABLE.COM/TAGS/KAFKA-TUTORIAL/INDEX.HTML\)](http://cloudurable.com/tags/kafka-tutorial/index.html)

🔗 [KINESIS \(HTTP://CLOUDURABLE.COM/TAGS/KINESIS/INDEX.HTML\)](http://cloudurable.com/tags/kinesis/index.html)

🔗 [KINESIS-CONSULTING \(HTTP://CLOUDURABLE.COM/TAGS/KINESIS-CONSULTING/INDEX.HTML\)](http://cloudurable.com/tags/kinesis-consulting/index.html)

🔗 [LINUX \(HTTP://CLOUDURABLE.COM/TAGS/LINUX/INDEX.HTML\)](http://cloudurable.com/tags/linux/index.html)

🔗 [METRICSD \(HTTP://CLOUDURABLE.COM/TAGS/METRICSD/INDEX.HTML\)](http://cloudurable.com/tags/metricSD/index.html)

🔗 [MICROSERVICES \(HTTP://CLOUDURABLE.COM/TAGS/MICROSERVICES/INDEX.HTML\)](http://cloudurable.com/tags/microservices/index.html)

🔗 [MICROSERVICES-ARCHITECTURE \(HTTP://CLOUDURABLE.COM/TAGS/MICROSERVICES-ARCHITECTURE/INDEX.HTML\)](http://cloudurable.com/tags/microservices-architecture/index.html)

🔗 [NAT \(HTTP://CLOUDURABLE.COM/TAGS/NAT/INDEX.HTML\)](http://cloudurable.com/tags/nat/index.html)

🔗 [NODETOOL \(HTTP://CLOUDURABLE.COM/TAGS/NODETOOL/INDEX.HTML\)](http://cloudurable.com/tags/nodeTool/index.html)

🔗 [NUMA \(HTTP://CLOUDURABLE.COM/TAGS/NUMA/INDEX.HTML\)](http://cloudurable.com/tags/numa/index.html)

🔗 [PRODUCERS \(HTTP://CLOUDURABLE.COM/TAGS/PRODUCERS/INDEX.HTML\)](http://cloudurable.com/tags/producers/index.html)

🔗 [QBIT \(HTTP://CLOUDURABLE.COM/TAGS/QBIT/INDEX.HTML\)](http://cloudurable.com/tags/qbit/index.html)

🔗 [RAM \(HTTP://CLOUDURABLE.COM/TAGS/RAM/INDEX.HTML\)](http://cloudurable.com/tags/ram/index.html)

🔗 [REAKT \(HTTP://CLOUDURABLE.COM/TAGS/REAKT/INDEX.HTML\)](http://cloudurable.com/tags/reakt/index.html)

🔗 [REPLICATION \(HTTP://CLOUDURABLE.COM/TAGS/REPLICATION/INDEX.HTML\)](http://cloudurable.com/tags/replication/index.html)

🔗 [SCHEMA-REGISTRY \(HTTP://CLOUDURABLE.COM/TAGS/SCHEMA-REGISTRY/INDEX.HTML\)](http://cloudurable.com/tags/schema-registry/index.html)

🔗 [SMACK \(HTTP://CLOUDURABLE.COM/TAGS/SMACK/INDEX.HTML\)](http://cloudurable.com/tags/smack/index.html)

🔗 [SPARK \(HTTP://CLOUDURABLE.COM/TAGS/SPARK/INDEX.HTML\)](http://cloudurable.com/tags/spark/index.html)

🔗 [SPARK--CASSANDRA \(HTTP://CLOUDURABLE.COM/TAGS/SPARK--CASSANDRA/INDEX.HTML\)](http://cloudurable.com/tags/spark--cassandra/index.html)

🔗 [SPARK-TRAINING \(HTTP://CLOUDURABLE.COM/TAGS/SPARK-TRAINING/INDEX.HTML\)](http://cloudurable.com/tags/spark-training/index.html)

🔗 [SPARK-TUTORIAL \(HTTP://CLOUDURABLE.COM/TAGS/SPARK-TUTORIAL/INDEX.HTML\)](http://cloudurable.com/tags/spark-tutorial/index.html)

🔗 [SSH \(HTTP://CLOUDURABLE.COM/TAGS/SSH/INDEX.HTML\)](http://cloudurable.com/tags/ssh/index.html)

🔗 [SSH-CONFIG \(HTTP://CLOUDURABLE.COM/TAGS/SSH-CONFIG/INDEX.HTML\)](http://cloudurable.com/tags/ssh-config/index.html)

🔗 [SSL \(HTTP://CLOUDURABLE.COM/TAGS/SSL/INDEX.HTML\)](http://cloudurable.com/tags/ssl/index.html)

🔗 [SYSTEMD \(HTTP://CLOUDURABLE.COM/TAGS/SYSTEMD/INDEX.HTML\)](http://cloudurable.com/tags/systemd/index.html)

🔗 [TLS \(HTTP://CLOUDURABLE.COM/TAGS/TLS/INDEX.HTML\)](http://cloudurable.com/tags/tls/index.html)

🔗 [VAGRANT \(HTTP://CLOUDURABLE.COM/TAGS/VAGRANT/INDEX.HTML\)](http://cloudurable.com/tags/vagrant/index.html)

🔗 [VCPU \(HTTP://CLOUDURABLE.COM/TAGS/VCPU/INDEX.HTML\)](http://cloudurable.com/tags/vcpu/index.html)

🔗 [VPC \(HTTP://CLOUDURABLE.COM/TAGS/VPC/INDEX.HTML\)](http://cloudurable.com/tags/vpc/index.html)

🔗 [WHAT-IS-KAFKA \(HTTP://CLOUDURABLE.COM/TAGS/WHAT-IS-KAFKA/INDEX.HTML\)](http://cloudurable.com/tags/what-is-kafka/index.html)

Apache Spark Training (<http://cloudurable.com/spark-training/index.html>)

Kafka Tutorial (<http://cloudurable.com/blog/kafka-tutorial/index.html>)

Akka Consulting (<http://cloudurable.com/akka-consulting/index.html>)

Cassandra Training (<http://cloudurable.com/cassandra-course/index.html>)

AWS Cassandra Database Support ([http://cloudurable.com/subscription\\_support\\_benefits\\_cassandra/index.html](http://cloudurable.com/subscription_support_benefits_cassandra/index.html))

Kafka Support Pricing ([http://cloudurable.com/subscription\\_support/index.html?q=kafka](http://cloudurable.com/subscription_support/index.html?q=kafka))

Cassandra Database Support Pricing ([http://cloudurable.com/subscription\\_support/index.html?q=cassandra](http://cloudurable.com/subscription_support/index.html?q=cassandra))

Non-stop Cassandra (<http://cloudurable.com/cloudurable-cassandra-watchdog/index.html?q=cassandra>)

Watchdog (<http://cloudurable.com/cloudurable-cassandra-watchdog/index.html?q=watchdog>)

Advantages of using Cloudurable™ (<http://cloudurable.com/advantages/index.html>)

Cassandra Consulting (<http://cloudurable.com/service-quick-start-mentoring-cassandra-or-kafka-aws-ec2/index.html>)

Cloudurable™| Guide to AWS Cassandra Deploy (<http://cloudurable.com/ppt/amazon-cassandra.pdf>)

Cloudurable™| AWS Cassandra Guidelines and Notes (<http://cloudurable.com/ppt/amazon-cassandra-notes.pdf>)

Free guide to deploying Cassandra on AWS (<http://cloudurable.com/cassandra-aws-consulting/index.html>)

Kafka Training (<http://cloudurable.com/kafka-training/index.html>)

Kafka Consulting (<http://cloudurable.com/kafka-aws-consulting/index.html>)

DynamoDB Training (<http://cloudurable.com/dynamodb-training/index.html>)

DynamoDB Consulting (<http://cloudurable.com/dynamodb-consulting/index.html>)

Kinesis Training (<http://cloudurable.com/kinesis-training/index.html>)

Kinesis Consulting (<http://cloudurable.com/kinesis-consulting/index.html>)

Kafka Tutorial PDF (<http://cloudurable.com/blog/kafka-tutorial-v1/index.html>)

Redis Consulting (<http://cloudurable.com/redis-consulting/index.html>)

Redis Training (<http://cloudurable.com/redis-onsite-instructor-led-training/index.html>)

ElasticSearch / ELK Consulting (<http://cloudurable.com/elk-consulting/index.html>)

ElasticSearch Training (<http://cloudurable.com/elasticsearch-onsite-instructor-led-training/index.html>)

InfluxDB/TICK Training (<http://cloudurable.com/influxdb-onsite-instructor-led-training/index.html>) TICK Consulting (<http://cloudurable.com/tick-consulting/index.html>)

## ABOUT US

Cloudurable™: Leader in AWS cloud computing for Kafka™, Cassandra™ Database, Apache Spark, AWS CloudFormation™ DevOps. We do **Cassandra training, Apache Spark, Kafka training, Kafka consulting** and **cassandra consulting** with a focus on AWS and data engineering. (FAQ (<http://cloudurable.com/faq/index.html>))

---

## FOLLOW CLOUDURABLE™

facebook page (<https://www.facebook.com/cloudurable>)

google plus (<https://plus.google.com/116648719730180908239>)

twitter (<https://twitter.com/cloudurable>)

linkedin (<https://www.linkedin.com/company/17964258/>)

*Why Cloudurable™?*

Advantage of using Cloudurable™ (<http://cloudurable.com/advantages/index.html>)

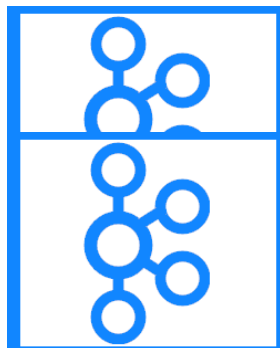
*About Cloudurable™?*

About Cloudurable™ (<http://cloudurable.com/faq/index.html>)

*What are the benefits of using subscription support?*

Benefits of Subscription Cassandra Support ([http://cloudurable.com/subscription\\_support\\_benefits\\_cassandra/index.html](http://cloudurable.com/subscription_support_benefits_cassandra/index.html))

## RECENT POSTS



**KAFKA CONSUMER: ADVANCED CONSUMERS** ([HTTP://CLOUDURABLE.COM/BLOG/KAFKA-ADVANCED-CONSUMER-1/INDEX.HTML](http://cloudurable.com/blog/kafka-advanced-consumer-1/index.html))

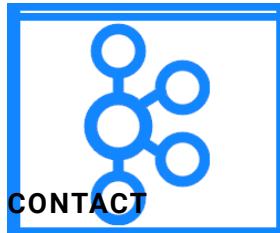
---

**KAFKA TUTORIAL** ([HTTP://CLOUDURABLE.COM/BLOG/KAFKA-TUTORIAL/INDEX.HTML](http://cloudurable.com/blog/kafka-tutorial/index.html))

---

m/blog/kafka-

**KAFKA TUTORIAL: CREATING ADVANCED KAFKA PRODUCERS IN JAVA** ([HTTP://CLOUDURABLE.COM/BLOG/KAFKA-TUTORIAL-KAFKA-PRODUCER-ADVANCED-JAVA-EXAMPLES/INDEX.HTML](http://cloudurable.com/blog/kafka-tutorial-kafka-producer-advanced-java-examples/index.html))

[n/blog/kafka-](http://cloudurable.com/blog/kafka-tutorial-kafka-consulting/index.html)**KAFKA CONSULTING (HTTP://CLOUDURABLE.COM/BLOG/KAFKA-CONSULTING/INDEX.HTML)**

<http://cloudurable.com/blog/kafka-tutorial-kafka-consulting/index.html>  
Cloudurable tech  
100 California Street  
San Francisco  
CA 94111  
USA  
examples/index.html)

**America**

(415) 758-1113 (tel:14157581113)

**GO TO CONTACT PAGE (/CONTACT/INDEX.HTML)**

Copyright © 2015 - 2018, Cloudurable™, all rights reserved. Streamline your Cassandra Database, Apache Spark and Kafka DevOps in AWS. SMACK/Lambda architecture consulting! Spark, Mesos, Akka, Cassandra and Kafka in AWS.

Apache Spark Training (<http://cloudurable.com/spark-training/index.html>), Akka Consulting (<http://cloudurable.com/akka-consulting/index.html>), AWS Cassandra Support ([http://cloudurable.com/subscription\\_support\\_benefits\\_cassandra/index.html](http://cloudurable.com/subscription_support_benefits_cassandra/index.html)), Cassandra Training (<http://cloudurable.com/cassandra-course/index.html>), Kafka Training (<http://cloudurable.com/kafka-training/index.html>), Cassandra Consulting (<http://cloudurable.com/service-architecture-analysis-cassandra-or-kafka-aws-ec2/index.html>), Kafka Consulting (<http://cloudurable.com/kafka-aws-consulting/index.html>), Spark Training (<http://cloudurable.com/spark-aws-emr-training/index.html>), Spark Consulting (<http://cloudurable.com/spark-aws-emr-consulting/index.html>), Kafka Tutorial (<http://cloudurable.com/blog/kafka-tutorial-v1/index.html>)

Template by DevCows (<https://github.com/devcows/hugo-universal-theme>)