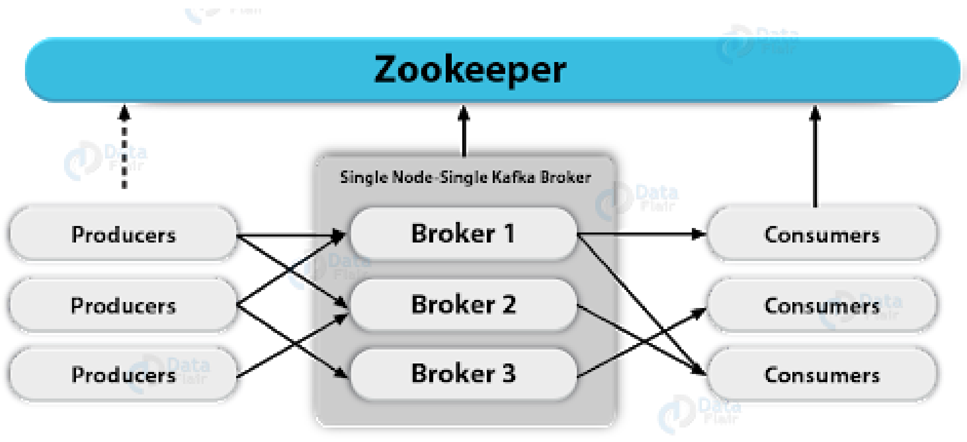
Role of Apache ZooKeeper in Kafka — Monitoring & Configuration

medium.com/@rinu.gour123/role-of-apache-zookeeper-in-kafka-monitoring-configuration-c5bd1a7e4226

What is ZooKeeper?

Apache ZooKeeper plays a very important role in system architecture as it works in the shadow of more exposed Big Data tools, as Apache Spark or Apache Kafka. In other words, Apache Zookeeper is a distributed, open-source configuration, synchronization service along with naming registry for distributed applications.



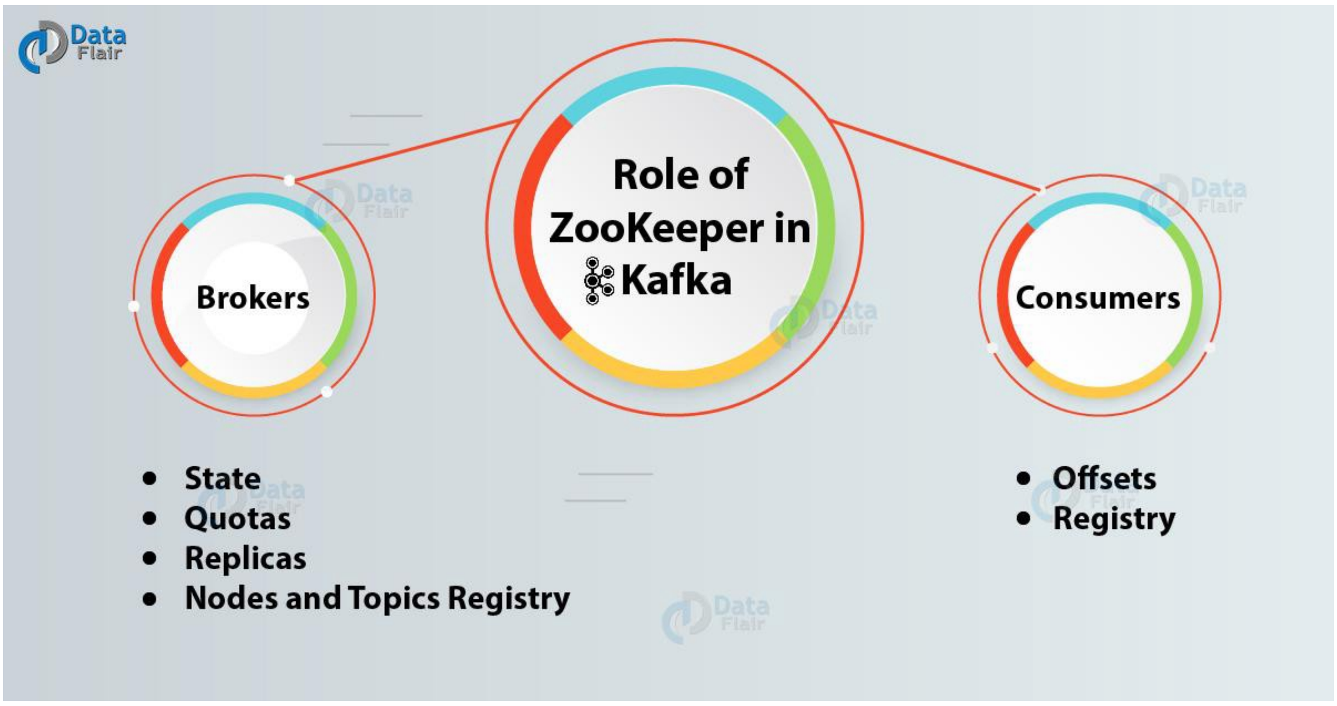
What is ZooKeeper

Originally, the ZooKeeper framework was built at “Yahoo!”. Because it helps to access their applications in an easy manner. Further, for organized service used by Hadoop, HBase, it became a standard and other distributed frameworks.

Now, let’s discuss the role of ZooKeeper in Kafka in detail:

ZooKeeper in Kafka

Basically, Kafka — ZooKeeper stores a lot of shared information about Kafka Consumers and Kafka Brokers, let’s discuss them in detail:



The requirement of ZooKeeper in Kafka

a. Kafka Brokers

Below given are the roles of ZooKeeper in Kafka Broker:

i. State

Zookeeper determines the state. That means, it notices, if the Kafka Broker is alive, always when it regularly sends heartbeats requests. Also, while the Broker is the constraint to handle replication, it must be able to follow replication needs.

ii. Quotas

In order to have different producing and consuming quotas, Kafka Broker allows some clients. This value is set in ZK under /config/clients path. Also, we can change it in bin/kafka-configs.sh script.

iii. Replicas

However, for each topic, Zookeeper in Kafka keeps a set of in-sync replicas (ISR). Moreover, if somehow previously selected leader node fails then on the basis of currently live nodes Apache ZooKeeper will elect the new leader.

iv. Nodes and Topics Registry

Basically, Zookeeper in Kafka stores nodes and topic registries. It is possible to find there all available brokers in Kafka and, more precisely, which Kafka topics are held by each broker, under /brokers/ids and /brokers/topics zNodes, they're stored. In addition, when it's started, Kafka broker create the register automatically.

b. Kafka Consumers

i. Offsets

ZooKeeper is the default storage engine, for consumer offsets, in Kafka's 0.9.1 release. However, all information about how many messages Kafka consumer consumes by each consumer is stored in ZooKeeper.

ii. Registry

Consumers in Kafka also have their own registry as in the case of Kafka Brokers. However, the same rules apply to it, ie. as ephemeral zNode, it's destroyed once the consumer goes down and the registration process is made automatically by the consumer.

How does Kafka talk to ZooKeeper?

Here, we will see how Kafka classes are responsible for working with ZooKeeper. Scala class representing Kafka is KafkaServer. Its startup() method, initZk() contains a call to method initializing ZooKeeper connection. There are several methods in this algorithm which we use in this Zookeeper method. Hence, as a result, the method creates a temporary connection to ZooKeeper, in this case. This session is responsible for creating zNodes corresponding to chroot if it's miAfterwarderwards, this connection closes and creates the final connection held by the server.
After, still inside initZk(), Kafka initializes all persistent zNodes, especially which server uses. We can retrieve there, among others: /consumers, /brokers/ids, /brokers/topics, /config, /admin/delete_topics, /brokers/seqid, /isr_change_notification, /config/topics, /config/clients.

Now, using synchronization to initialize other members, we can use this created ZooKeeper instance:

- Replica manager
- Config manager
- Coordinator, and controller

ZooKeeper Production Deployment

In order to store persistent cluster metadata, Kafka uses ZooKeeper. Suppose, we lost the Kafka data in |zk|, the mapping of replicas to Kafka Brokers and topic configurations would be lost as well, making our Kafka Cluster no longer functional and potentially resulting in total data loss.

Stable Version of ZooKeeper

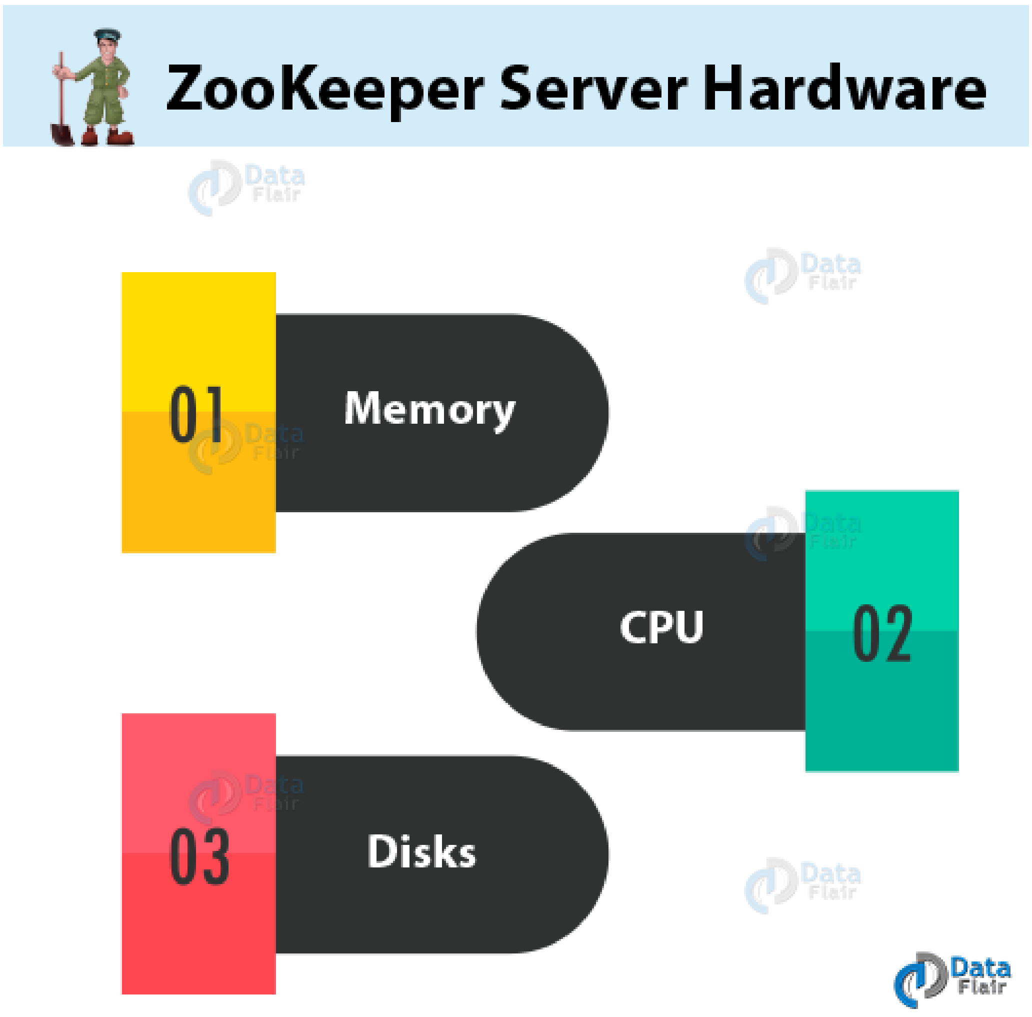
However, the current stable branch is 3.4 and the latest release of that branch is 3.4.9.
Also, we can use “four letter word” ENVI, to find the current version of a running server.
For example:

```
echo envi | nc localhost 2181
```

It shows all of the environment information for the ZooKeeper server, including the version.
Note: Only with this version of ZooKeeper, the ZooKeeper start script and tests the functionality of ZooKeeper.

The hardware of ZooKeeper Server

Here are some guidelines, for choosing proper hardware for a cluster of ZooKeeper servers.



Hardware in ZooKeeper

a. Memory

Basically, ZooKeeper is not a memory intensive application when handling only data stored by Kafka. Make sure, a minimum of 8 GB of RAM should be there for ZooKeeper use, in a typical production use case.

b. CPU

As a Kafka metadata, ZooKeeper store does not heavily consume CPU resources. ZooKeeper also offers a latency sensitive function. That implies we must consider providing a dedicated CPU core to ensure context switching is not an issue if it must compete for CPU with other processes.

c. Disks

In order to maintain a healthy ZooKeeper cluster, Disk performance is very essential. To perform optimally, we recommend using Solid state drives (SSD) as ZooKeeper must have low latency disk writes.

8. JVM(Java Virtual Machine)

Generally, ZooKeeper runs as a JVM. When running for the Kafka use case, it is not notably heap. For most use cases and monitoring heap usage, we recommend 1 GB heap size to stop delays due to garbage collection.

9. Important Configuration Options for ZooKeeper

There is no need for configuration tuning for most deployments. Some important parameters to consider are:

a. clientPort

In this port ZooKeeper, clients will listen on. Here we will connect the Brokers to ZooKeeper. Typically this is set to 2181.
Type: int
Importance: required

b. dataDir

It is the directory where we store ZooKeeper data. However, it should be a dedicated disk that is ideally an SSD.
Type: string
Importance: required

c. tickTime

For ZooKeeper, the unit of time is translated to milliseconds. It governs all ZooKeeper time-dependent operations. We use it for heartbeats and timeouts especially.
Note: minimum session timeout will be two ticks.
Type: int
Default: 2000
Importance: high

d. maxClientCnxns

It is the maximum number of client connections for a ZooKeeper server. Set this to 0 (unlimited), to avoid running out of allowed connections.
Type: int
Default: 60
Importance: high

e. autopurge.snapRetainCount

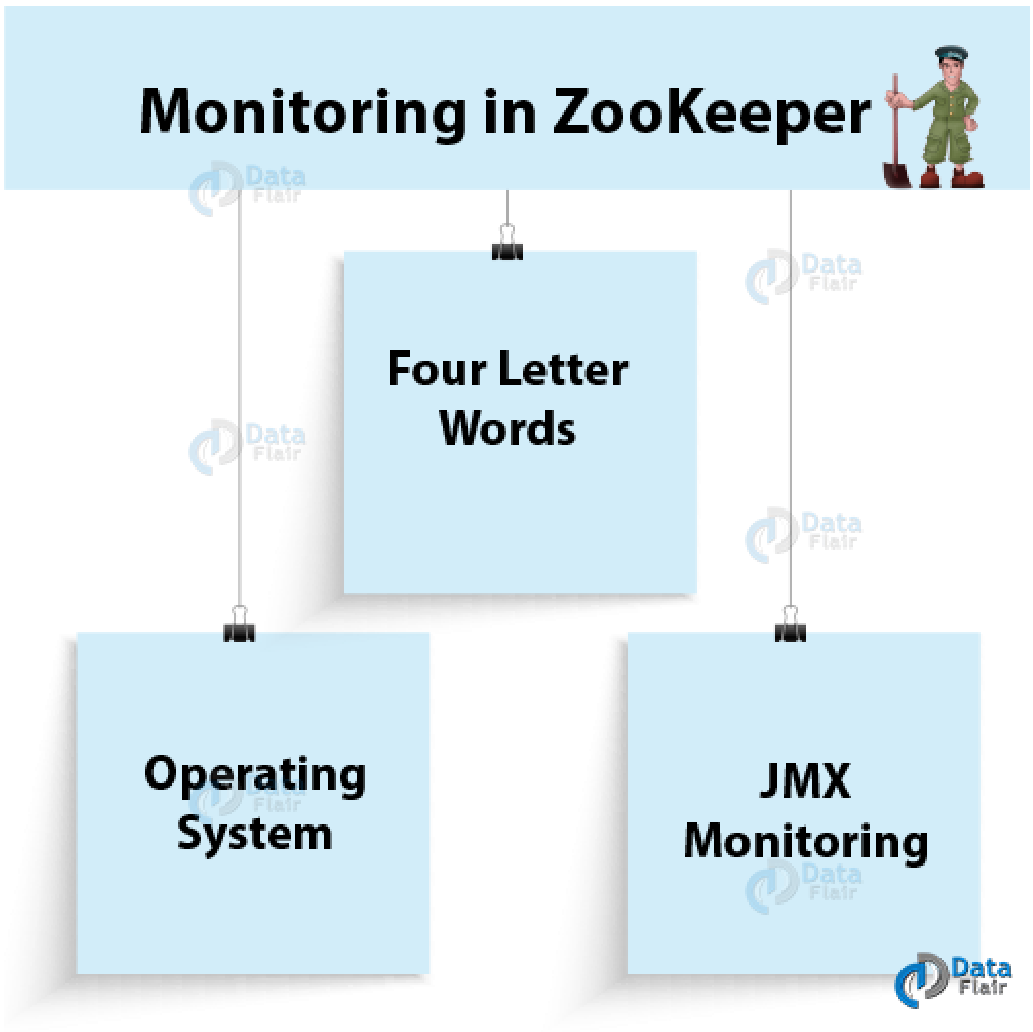
It simply retains the autopurge.snapRetainCount most recent snapshots and the corresponding transaction logs in the dataDir and dataLogDir respectively and also deletes the remaining, at the time of enabling it.
Type: int
Default: 3
Importance: high

f. autopurge.purgeInterval

It is important that purge task is triggered while the time interval in hours. To enable the auto purging, set it to a positive integer (1 and above).
Type: int
Default: 0
Importance: high

Monitoring in ZooKeeper — Kafka

In order to ensure servers are functioning properly and proactively identify issues, we need to monitor ZooKeeper servers. Best practices for monitoring are:



Monitoring in ZooKeeper

a. Operating System

When the ZooKeeper service will start to struggle, the underlying OS metrics can help predict. We should monitor:

i. Number of open file handles

This should be done system-wide and for the user running the ZooKeeper process. Values should be considered with respect to the maximum allowed number of open file handles. It also opens and closes connections often, and moreover, it needs an available pool of file handles to choose from.

ii. Network bandwidth usage

ZooKeeper is sensitive to timeouts caused by network latency just because it keeps track of state. If somehow the network bandwidth is saturated then only it is possible that we may experience hard to explain timeouts with client sessions, although that results in making Kafka cluster less reliable.

b. “Four Letter Words”

Basically, ZooKeeper only response to a set of commands, each one is of four letters. To run the commands we must send a message (via netcat or telnet) to the ZooKeeper client port. For example echo stat | nc localhost 2181 would return the output of the STAT command to stdout.

c. JMX Monitoring

JMX metrics that are important to monitor:
NumAliveConnections
OutstandingRequests
AvgRequestLatency
MaxRequestLatency
HeapMemoryUsage (Java built-in)
Moreover, by the SessionExpireListener, Kafka, tracks the number of relevant ZooKeeper events. Basically, it is monitored to ensure the health of ZooKeeper-Kafka interactions:
ZooKeeperAuthFailuresPerSec (secure environments only)
ZooKeeperDisconnectsPerSec
ZooKeeperExpiresPerSec
ZooKeeperReadOnlyConnectsPerSec
ZooKeeperSaslAuthenticationsPerSec (secure environments only)
ZooKeeperSyncConnectsPerSec

ZooKeeper Multi-node Setup

The ZooKeeper servers deploy on multiple nodes, in a real production environment. This is what we call an ensemble. Basically, an ensemble is a set of 2n + 1 ZooKeeper servers, where n refers to any number greater than 0. Also, to perform majority elections for leadership, the odd number of servers allows ZooKeeper. There can be up to n failed servers in an ensemble and the ZooKeeper cluster will keep the quorum.

Post Deployment

ZooKeeper largely runs without much maintenance, once we deploy our ZooKeeper cluster.
So, this was all about ZooKeeper role in Kafka, Hope you like our explanation.

Conclusion

Hence, in this role of ZooKeeper in Kafka tutorial, we have seen that Kafka really needs ZooKeeper to work efficiently in the Kafka cluster. In conclusion, we have learned that all Kafka broker configuration stores in ZooKeeper zNodes. Even the Kafka consumers need Zookeeper to know about the last consumed message. Moreover, we also studied ZooKeeper server monitoring, hardware in the ZooKeeper server. At last, we discussed important configurations options for ZooKeeper. However, if any doubt occurs, regarding ZooKeeper in Kafka, feel free to ask in the comment section.