Zookeeper Quorum

Apache Kafka uses ZooKeeper to store persistent cluster metadata and is a critical component of the Confluent Platform deployment.

For example, if you lost the Kafka data in ZooKeeper, the mapping of replicas to Brokers and topic configurations would be lost as well, making your Kafka cluster no longer functional and potentially resulting in total data loss.

In a production environment, the ZooKeeper servers will be deployed on multiple nodes. This is called an ensemble. An ensemble is a set of `2n + 1` ZooKeeper servers where `n` is any number greater than 0. The odd number of servers allows ZooKeeper to perform majority elections for leadership.

At any given time, there can be up to `n` failed servers in an ensemble and the ZooKeeper cluster will keep `quorum`. If at any time, `quorum` is lost, the ZooKeeper cluster will go down.

A few general considerations for multi-node ZooKeeper ensembles:

- Start with a small ensemble of three or five servers and only scale as truly necessary (or as required for fault tolerance). Each write must be propagated to a `quorum` of servers in the ensemble. This means that if you choose to run an ensemble of three servers, you are tolerant to one server being lost, and writes must propagate to two servers before they are committed. If you have five servers, you are tolerant to two servers being lost, but writes must propagate to three servers before they are committed.
- Redundancy in the physical/hardware/network layout: try not to put them all in the same rack, decent (but don't go nuts) hardware, try to keep redundant power and network paths, etc.
- Virtualization: place servers in different availability zones to avoid correlated crashes and to make sure that the storage system available can accommodate the requirements of the transaction logging and snapshotting of ZooKeeper.
- When it doubt, keep it simple. ZooKeeper holds important data, so prefer stability and durability over trying a new deployment model in production.

Quorum:

Ensemble is nothing but a cluster of Zookeeper servers. **Quorum** defines the rule to form a *healthy Ensemble*.

Quorum size should be calculated by Majority Rule

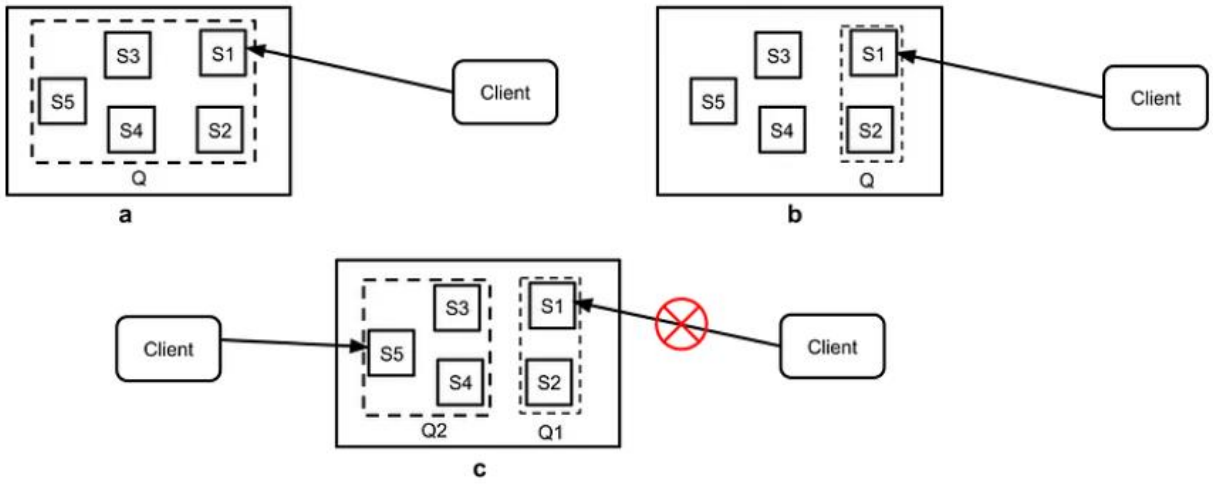
Majority rule: QN = (N + 1) / 2

Healthy Ensemble: A cluster with only one active Leader at any given point of time, hence *fault tolerant*.

Points to remember about Zookeeper before moving further:

1. The Zookeeper client has all the Zookeeper server IP list. If Zookeeper client disconnects from any server, then Zookeeper client tries to connect with another IP(Zookeeper server).
2. All the write requests are processed via Leader. In the following example, S1 is the Leader.

Case 1 - Less Than Majority Rule (Less than 3 servers):



In this case, Qn = 2 (we are not following the Majority rule). If 2 servers are available in Quorum, then Zookeeper will process the request in the following way:

1) Initially, we have 5 Zookeeper servers in quorum Q **(as shown in above image a)**. A client wants to create a node N on the Zookeeper server. So, the client will get connected to any of the servers. The server will transfer the request to the Leader of the quorum Q(S1).

2) At the same time, due to network partition, quorum Q is left with only 2 servers(S1 and S2).

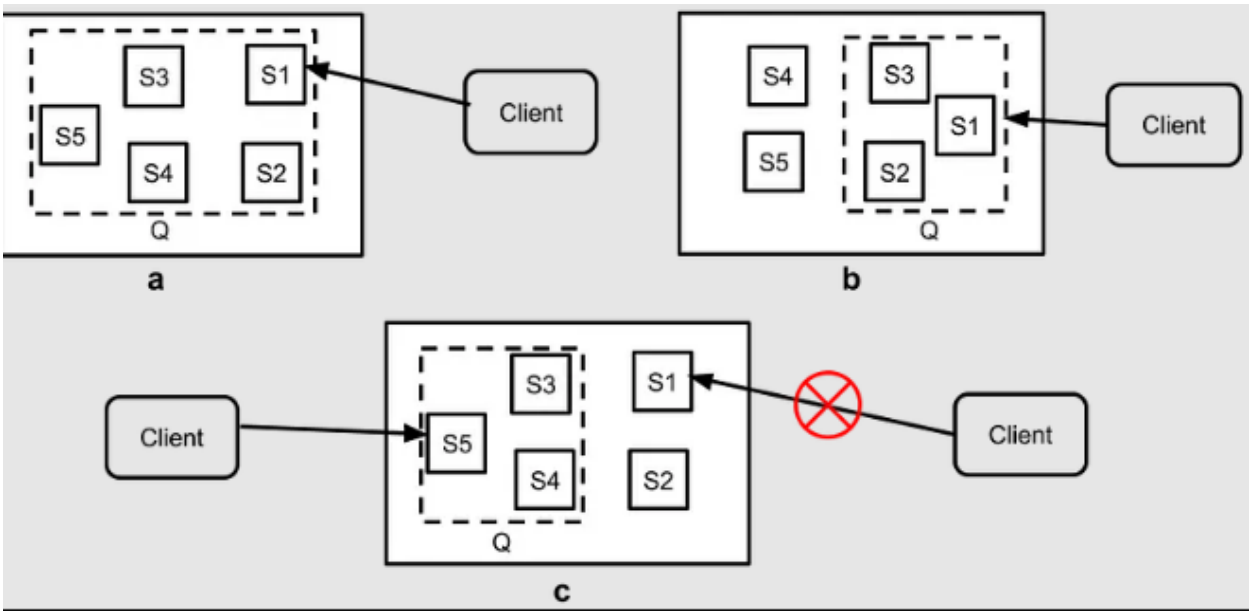
So, only S1 and S2 servers will have the data (Node N) as server S3, S4 and S5 got disconnected. **(as shown in above image b).**

3) After some time, the client makes a request to get data from the same node N but server S1, S2 got disconnected from the client due to network partition(**as shown above image c**).

So now the client got connected with any of the servers S3, S4, S5. As we know, none of these servers have the data (Node N).

This will create the **Split Brain problem**.

Case 2 - With Majority Rule (Minimum 3 servers):



In this case Qn = 3. If 3 servers are available in Quorum, then Zookeeper will process the request in the following way:

1. The client connects to the S1(Leader) server to create a node.

2. To process the client request or to create a quorum, a minimum of 3 servers are required.

So say as, at the time client requests, Server S1, S2 got disconnected from servers S3, S4, S5. Hence the request will not be processed and the client has to handle this disconnection error.

3. Let's suppose, S1, S2, and S3 connected and created a quorum Q. Server S4 and S5 are still disconnected.

Hence, node request will be processed only on three servers S1, S2, S3.

4. Later on, the client requests to get data for the same node N from any of the servers S1, S2, S3.

Now suppose, the client got disconnected from servers(S1, S2) and after a while, S3,

S4, and S5 make a quorum after they got connected.
In this case, S3 is the leader as it has the requested data but before processing the request, S3 will be syncing with S4 and S5 to have the same data in both the nodes. After sync only, client request will be processed.

Case 3 - More than Majority rule(More than 3 servers):
In this case Qn = 4, If 4 servers are available in Quorum, then Zookeeper will process the request. But quorum servers more than Majority rule has one disadvantage. Before return success result to the client, we have to write on one extra server. This will increase latency.
If the total number of servers (N) is even, Then the above case also happens. Let's suppose N = 6. According to Majority rule Qn = 4. This extra server will increase the latency to perform write operation.

Majority Wins Rule Proof:

Majority wins

As we know, we required 3 servers for successful write,
W: Write successfully.
N: Write failed, due to a network partition.
When we say we write on the majority of servers and Quorum create with the majority of servers then one of a server(that have the latest data) intersect with both quorum. you can see in the above case1 in majority wins rule proof.

S3 intersect with both left (S1, S2)and right quorum (S4, S5).