APACHE CASSANDRA

Apache Cassandra Architecture

- Architecture Requirements of Cassandra
- Cassandra Architecture
- Effects of the Architecture
- Cassandra Write Process
- Rack

Architecture Requirements of Cassandra

Cassandra was designed to address many architecture requirements. The most important requirement is to ensure **there is no single point of failure.** This means that if there are 100 nodes in a cluster and a node fails, the cluster should continue to operate.

This is in contrast to Hadoop where the **namenode failure can cripple the entire system**. Another requirement is to have **massive scalability** so that a cluster can **hold hundreds or thousands of nodes.** It should be possible to add a new node to the cluster without stopping the cluster.

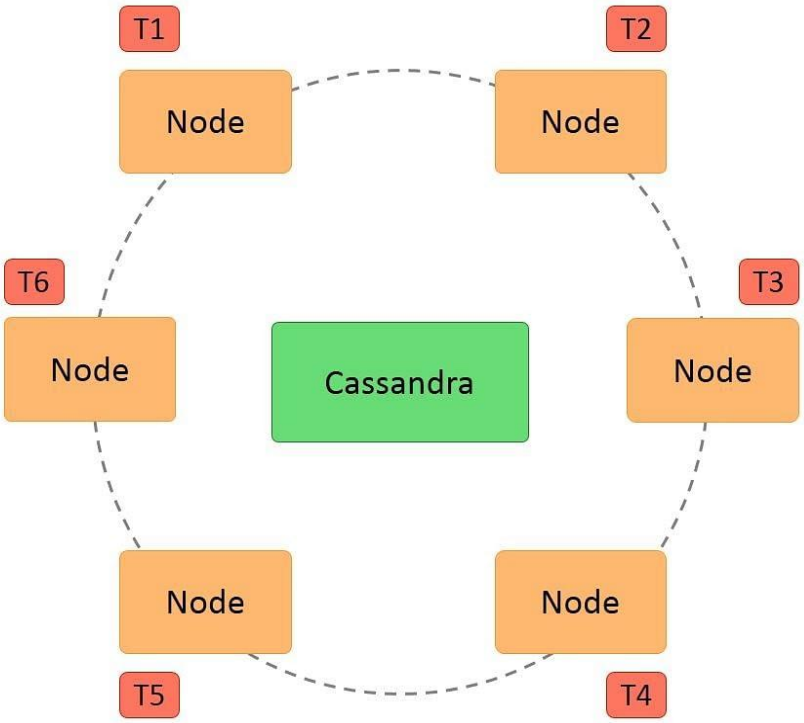
Further, the architecture should be highly distributed so that both **processing and data can be distributed.** Also, high performance of read and write of data is expected so that the system can be used in real-time.

Cassandra Architecture

Some of the features of Cassandra architecture are as follows:

- Cassandra is designed such that it has no master or slave nodes.
- It has a ring-type architecture, that is, its nodes are logically distributed like a ring.
- Data is automatically distributed across all the nodes.
- Similar to HDFS, data is replicated across the nodes for redundancy.
- Data is kept in memory and lazily written to the disk.
- Hash values of the keys are used to distribute the data among nodes in the cluster.

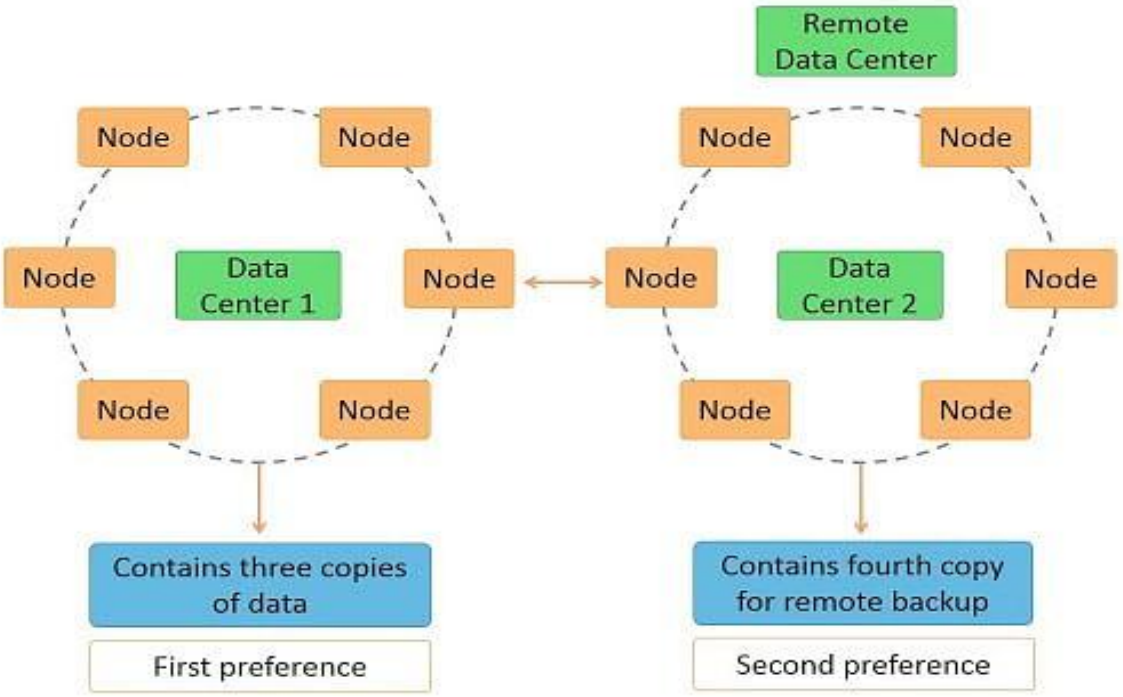
A hash value is a number that maps any given key to a numeric value. For example, the string 'ABC' may be mapped to 101, and decimal number 25.34 may be mapped to 257. A hash value is generated using an algorithm so that the same value of the key always gives the same hash value. In a ring architecture, each node is assigned a token value, as shown in the image below:



Additional features of Cassandra architecture are:

- Cassandra architecture supports multiple data centers.
- Data can be replicated across data centers.

You can keep three copies of data in one data center and the fourth copy in a remote data center for remote backup. Data reads prefer a local data center to a remote data center.



Effects of the Architecture

Cassandra architecture enables transparent distribution of data to nodes. This means you can determine the location of your data in the cluster based on the data. Any node can accept any request as there are no masters or slaves. If a node has the data, it will return the data. Else, it will send the request to the node that has the data.

You can specify the number of replicas of the data to achieve the required level of redundancy. For example, if the data is very critical, you may want to specify a replication factor of 4 or 5.

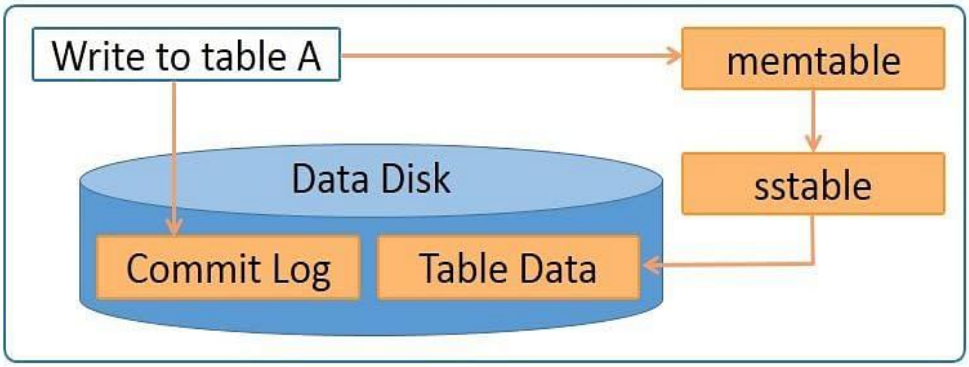
If the data is not critical, you may specify just two. It also provides tunable consistency, that is, the level of consistency can be specified as a trade-off with performance. Transactions are always written to a commit log on disk so that they are durable.

Cassandra Write Process

The Cassandra write process ensures fast writes. Steps in the Cassandra write process are:

- Data is written to a commitlog on disk.
- The data is sent to a responsible node based on the hash value.
- Nodes write data to an in-memory table called memtable.
- From the memtable, data is written to an sstable in memory. Sstable stands for Sorted String table. This has a consolidated data of all the updates to the table.
- From the sstable, data is updated to the actual table.
- If the responsible node is down, data will be written to another node identified as tempnode. The tempnode will hold the data temporarily till the responsible node comes alive.

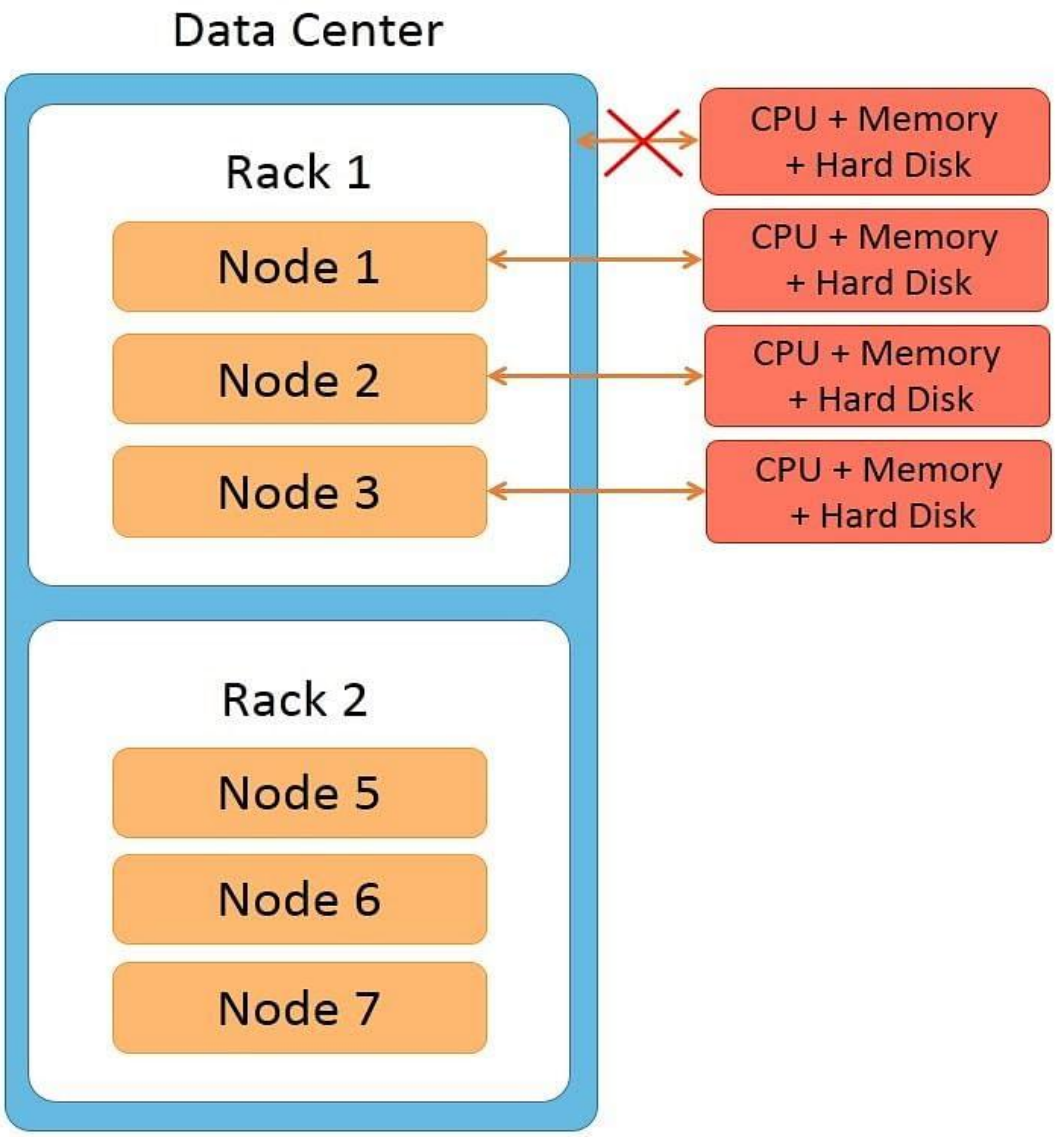
The diagram below depicts the write process when data is written to table A.



Data is written to a commitlog on disk for persistence. It is also written to an in-memory memtable. Memtable data is written to sstable which is used to update the actual table.

Rack

The term ‘rack’ is usually used when explaining network topology. A rack is a group of machines housed in the same physical box. Each machine in the rack has its own CPU, memory, and hard disk. However, the rack has no CPU, memory, or hard disk of its own.



Features of racks are:

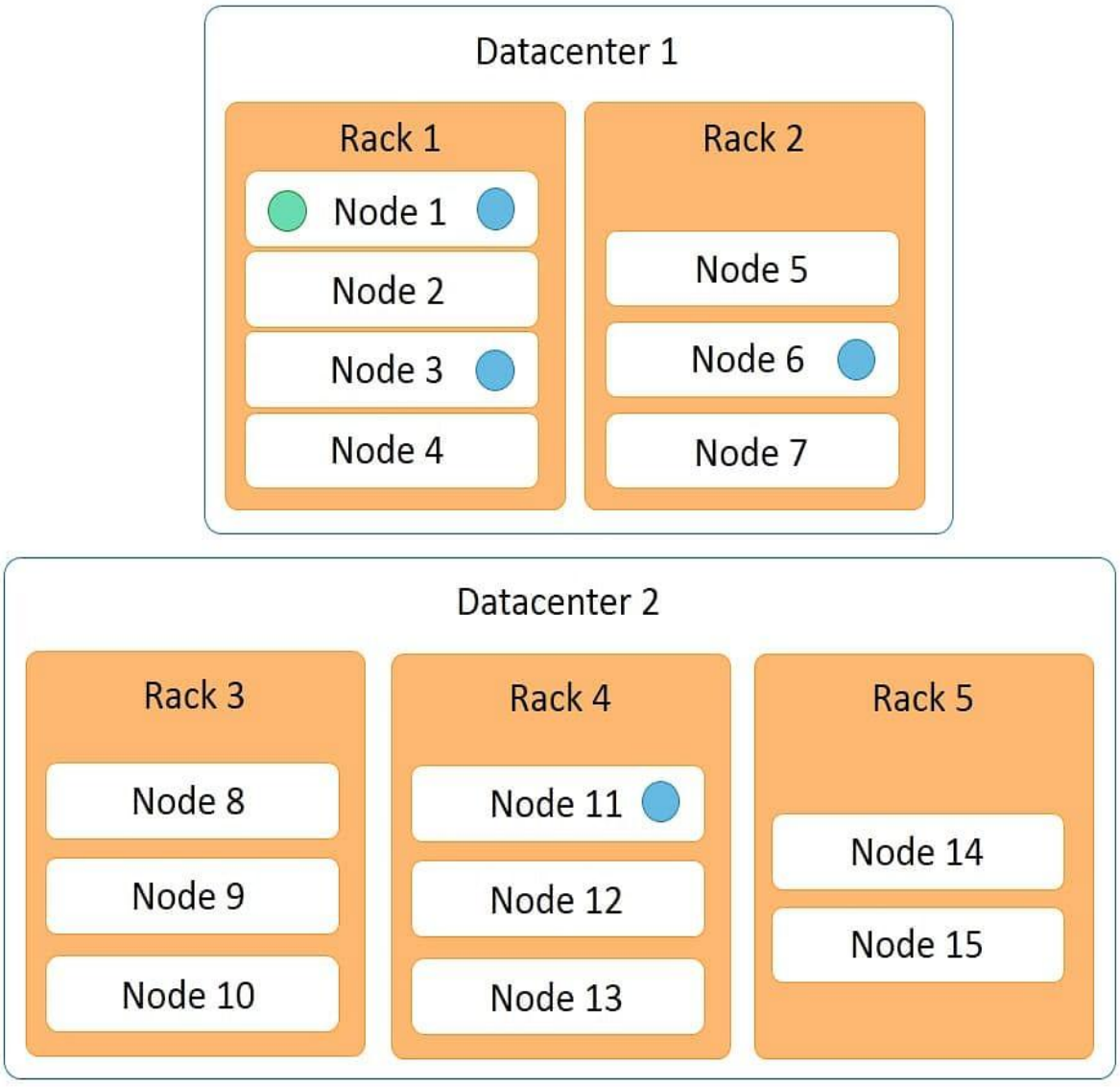
- All machines in the rack are connected to the network switch of the rack
- The rack’s network switch is connected to the cluster.
- All machines on the rack have a common power supply. It is important to notice that a rack can fail due to two reasons: a network switch failure or a power supply failure.
- If a rack fails, none of the machines on the rack can be accessed. So it would seem as though all the nodes on the rack are down.

Cassandra Read Process

The Cassandra read process ensures fast reads. Read happens across all nodes in parallel. If a node is down, data is read from the replica of the data. Priority for the replica is assigned on the basis of distance. Features of the Cassandra read process are:

- Data on the same node is given first preference and is considered data local.
- Data on the same rack is given second preference and is considered rack local.
- Data on the same data center is given third preference and is considered data center local.
- Data in a different data center is given the least preference.

Data in the memtable and sstable is checked first so that the data can be retrieved faster if it is already in memory. The diagram below represents a Cassandra cluster.



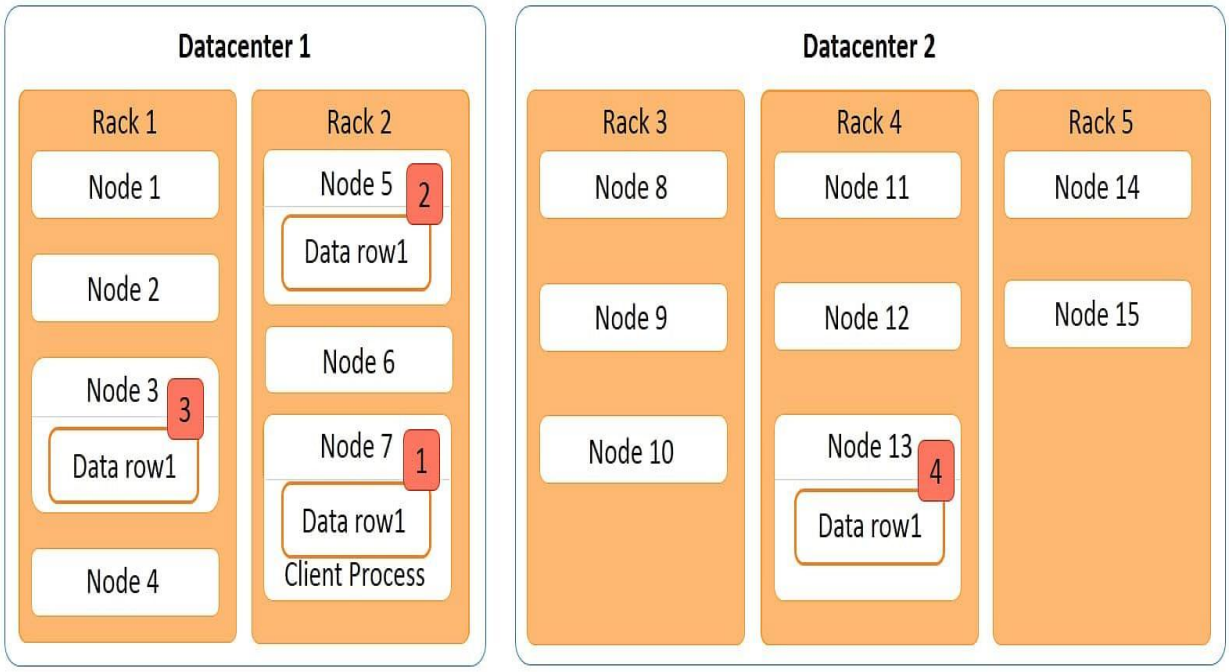
It has two data centers:

- data center 1
- data center 2

Data center 1 has two racks, while data center 2 has three racks. Fifteen nodes are distributed across this cluster with nodes 1 to 4 on rack 1, nodes 5 to 7 on rack 2, and so on.

Example of Cassandra Read Process

The Cassandra read process is illustrated with an example below.



The diagram below explains the Cassandra read process in a cluster with two data centers, five racks, and 15 nodes. In the image, place data row1 in this cluster. Data row1 is a row of data with four replicas.

- The first copy is stored on node 3
- The second copy is stored on node 5
- The third copy is stored on node 7

All these nodes are in data center 1. The fourth copy is stored on node 13 of data center 2. If a client process is running on data node 7 wants to access data row1; node 7 will be given the highest preference as the data is local here. The next preference is for node 5 where the data is rack local. The next preference is for node 3 where the data is on a different rack but within the same data center.

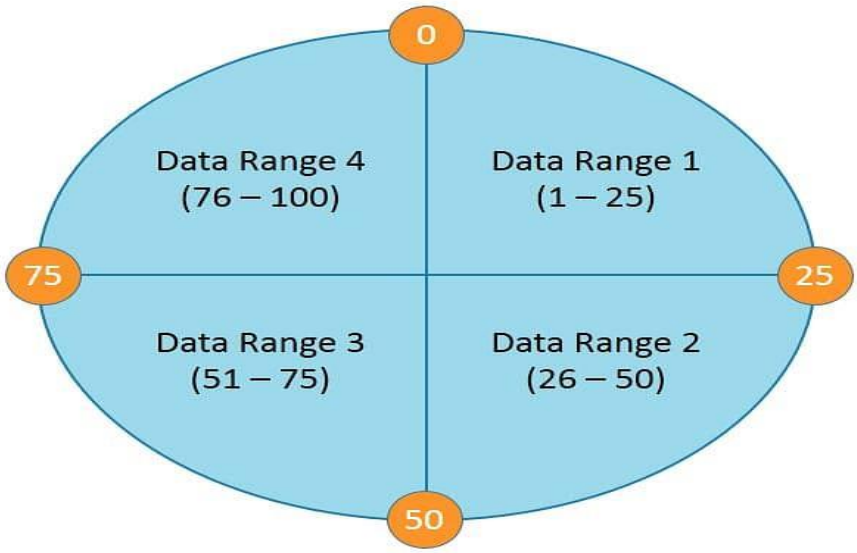
The least preference is given to node 13 that is in a different data center. So the read process preference in this example is node 7, node 5, node 3, and node 13 in that order.

Data Partitions

Cassandra performs transparent distribution of data by horizontally partitioning the data in the following manner:

- A hash value is calculated based on the primary key of the data.
- The hash value of the key is mapped to a node in the cluster
- The first copy of the data is stored on that node.
- The distribution is transparent as you can both calculate the hash value and determine where a particular row will be stored.

The following diagram depicts a four node cluster with token values of 0, 25, 50 and 75.



For a given key, a hash value is generated in the range of 1 to 100. Keys with hash values in the range 1 to 25 are stored on the first node, 26 to 50 are stored on the second node, 51 to 75 are stored on the third node, and 76 to 100 are stored on the fourth node. Please note that actual tokens and hash values in Cassandra are 127-bit positive integers.

Replication in Cassandra

Replication refers to the number of replicas that are maintained for each row. Replication provides redundancy of data for fault tolerance. A replication factor of 3 means that 3 copies of data are maintained in the system.

In this case, even if 2 machines are down, you can access your data from the third copy. The default replication factor is 1. A replication factor of 1 means that a single copy of the data is maintained, so if the node that has the data fails, you will lose the data.

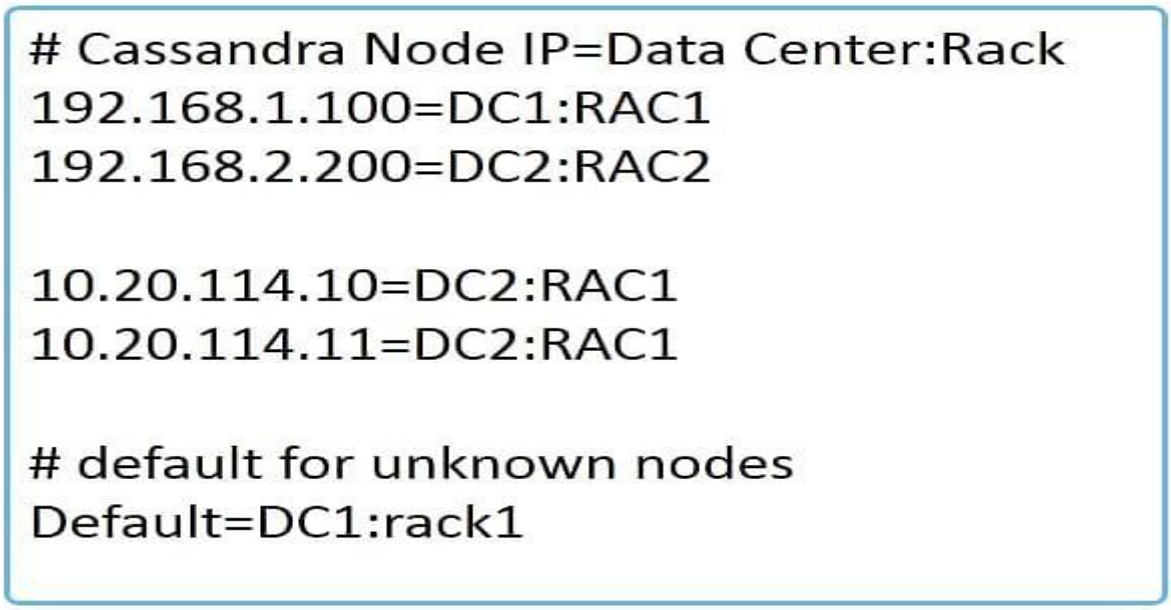
Cassandra allows replication based on nodes, racks, and data centers, unlike HDFS that allows replication based on only nodes and racks. Replication across data centers guarantees data availability even when a data center is down.

Network Topology

Network topology refers to how the nodes, racks and data centers in a cluster are organized. You can specify a network topology for your cluster as follows:

- Specify in the Cassandra-topology.properties file.
- Your data centers and racks can be specified for each node in the cluster.
- Specify <ip-address>=<data center>:<rack name>.
- For unknown nodes, a default can be specified.
- You can also specify the hostname of the node instead of an IP address.

The following diagram depicts an example of a topology configuration file



This file shows the topology defined for four nodes. The node with IP address 192.168.1.100 is mapped to data center DC1 and is present on the rack RAC1. The node with IP address 192.168.2.200 is mapped to data center DC2 and is present on the rack RAC2.

Similarly, the node with IP address 10.20.114.10 is mapped to data center DC2 and rack RAC1 and the node with IP address 10.20.114.11 is mapped to data center DC2 and rack RAC1. There is also a default assignment of data center DC1 and rack RAC1 so that any unassigned nodes will get this data center and rack.

Snitches

Snitches define the topology in Cassandra. A snitch defines a group of nodes into racks and data centers. Two types of snitches are most popular:

- Simple Snitch - A simple snitch is used for single data centers with no racks.
- Property File Snitch - A property file snitch is used for multiple data centers with multiple racks.

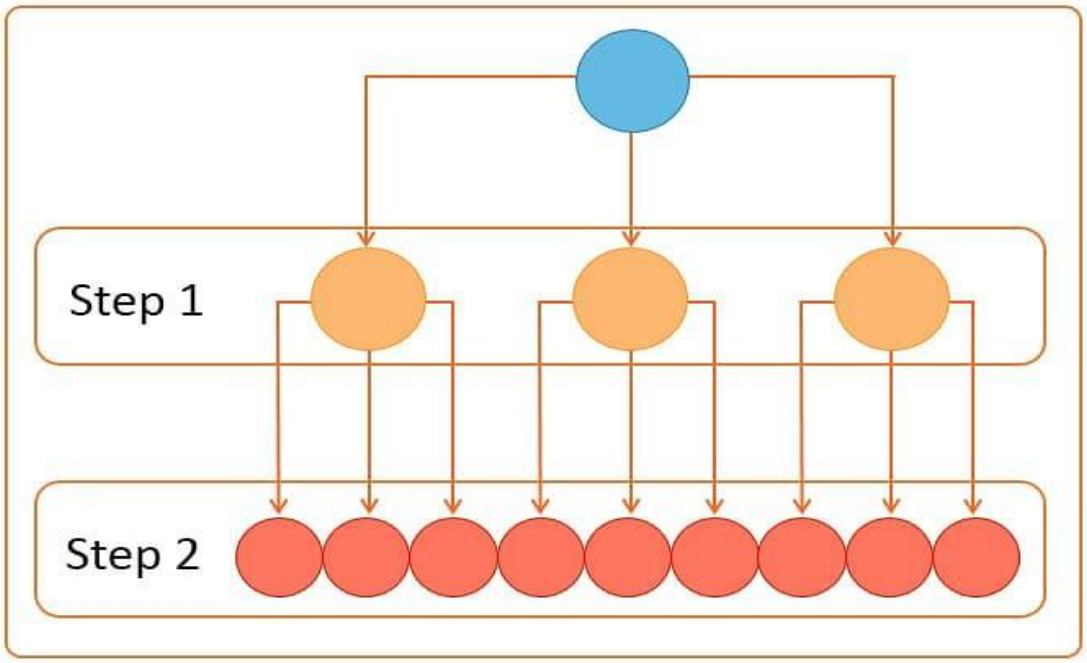
Replication in Cassandra is based on the snitches.

Gossip Protocol

Cassandra uses a gossip protocol to communicate with nodes in a cluster. It is an inter-node communication mechanism similar to the heartbeat protocol in Hadoop. Cassandra uses the gossip protocol to discover the location of other nodes in the cluster and get state information of other nodes in the cluster.

The gossip process runs periodically on each node and exchanges state information with three other nodes in the cluster. Eventually, information is propagated to all cluster nodes. Even if there are 1000 nodes, information is propagated to all the nodes within a few seconds.

The following image depicts the gossip protocol process.



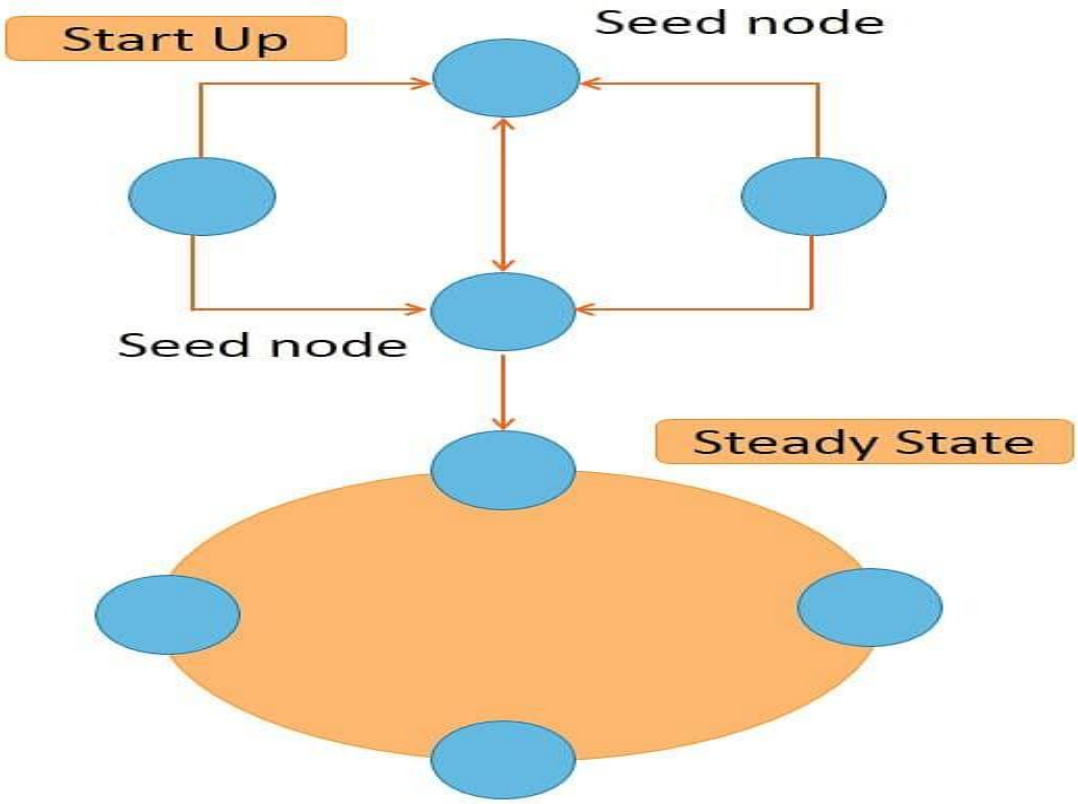
In step 1, one node connects to three other nodes. In step 2, each of the three nodes connects to three other nodes, thus connecting to nine nodes in total in step 2. So a total of 13 nodes are connected in 2 steps.

Seed Nodes

Seed nodes are used to bootstrap the gossip protocol. The features of seed nodes are:

- They are specified in the configuration file Cassandra.yaml.
- Seed nodes are used for bootstrapping the gossip protocol when a node is started or restarted.
- They are used to achieve a steady state where each node is connected to every other node but are not required during the steady state.

The diagram depicts a startup of a cluster with 2 seed nodes. Initially, there is no connection between the nodes.



On startup, two nodes connect to two other nodes that are specified as seed nodes. Once all the four nodes are connected, seed node information is no longer required as steady state is achieved.

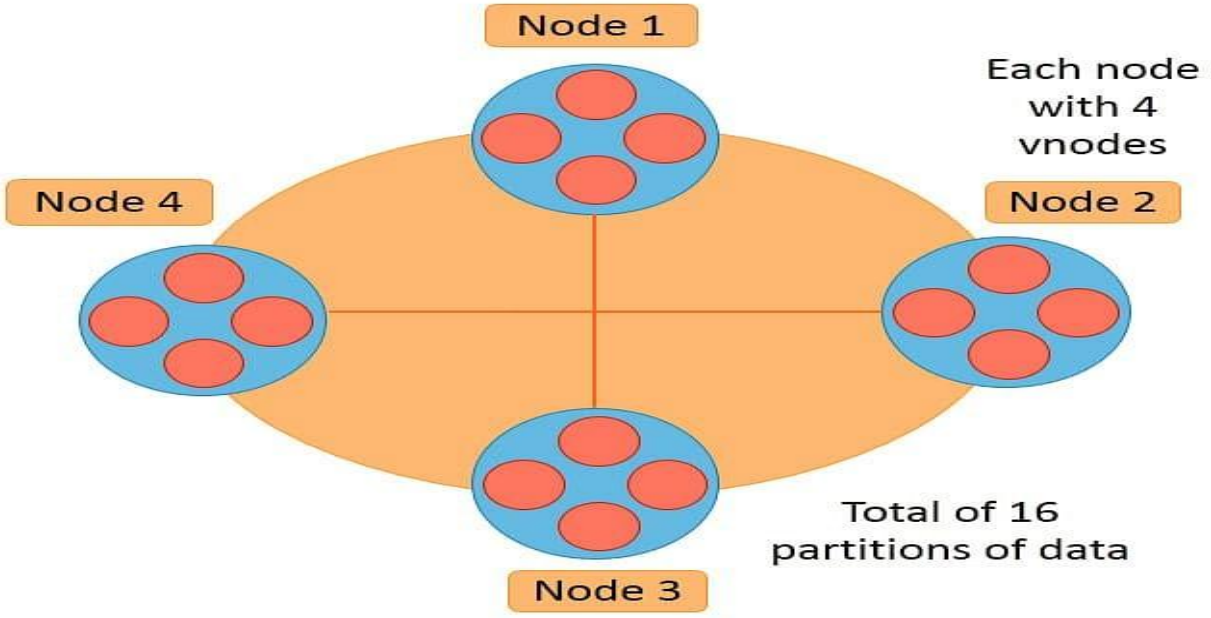
Configuration

The main configuration file in Cassandra is the Cassandra.yaml file. We will look at this file in more detail in the lesson on installation. Right now, let us remember that this file contains the name of the cluster, seed nodes for this node, topology file information, and data file location. This file is located in /etc/Cassandra in some installations and in /etc/Cassandra/conf directory in others.

Virtual Nodes

Virtual nodes in a Cassandra cluster are also called vnodes. Vnodes can be defined for each physical node in the cluster. Each node in the ring can hold multiple virtual nodes. By default, each node has 256 virtual nodes.

Virtual nodes help achieve finer granularity in the partitioning of data, and data gets partitioned into each virtual node using the hash value of the key. On adding a new node to the cluster, the virtual nodes on it get equal portions of the existing data. So there is no need to separately balance the data by running a balancer. The image depicts a cluster with four physical nodes.



Each physical node in the cluster has four virtual nodes. So there are 16 vnodes in the cluster.

If 32TB of data is stored on the cluster, each vnode will get 2TB of data to store. If another physical node with 4 virtual nodes is added to the cluster, the data will be distributed to 20 vnodes in total such that each vnode will now have 1.6 TB of data.

Token Generator

The token generator is used in Cassandra versions earlier than version 1.2 to assign a token to each node in the cluster. In these versions, there was no concept of virtual nodes and only physical nodes were considered for distribution of data.

The token generator tool is used to generate a token for each node in the cluster based on the data centers and number of nodes in each data center. A token in Cassandra is a 127-bit integer assigned to a node. Data partitioning is done based on the token of the nodes as described earlier in this lesson. Starting from version 1.2 of Cassandra, vnodes are also assigned tokens and this assignment is done automatically so that the use of the token generator tool is not required.

Example of Token Generator

A token generator is an interactive tool which generates tokens for the topology specified. Let us now look at an example in which the token generator is run for a cluster with 2 data centers.

- Type token-generator on the command line to run the tool.
- A question is asked next: “How many data centers will participate in this cluster?” In the example, specify 2 as the number of data centers and press enter.
- Next, the question: “How many nodes are in data center number 1?” is asked. Type 5 and press enter.
- The next question is: “How many nodes are in data center number 2?” Type 4 and press enter.

The tokens are calculated and displayed below.

```
token-generator
How many datacenters will participate in this cluster? 2
How many nodes are in datacenter #1? 5
How many nodes are in datacenter #2? 4
 DC #1:
  Node #1:                                0
  Node #2:   34028236692093846346337460743176821145
  Node #3:   68056473384187692692674921486353642290
  Node #4:  102084710076281539039012382229530463435
  Node #5:  136112946768375385385349842972707284580
DC #2:
  Node #1:  169417178424467235000914166253263322299
  Node #2:   41811290829115311202148688466350243003
  Node #3:   84346586694232619135070514395321269435
  Node #4:  126881882559349927067992340324292295867
```

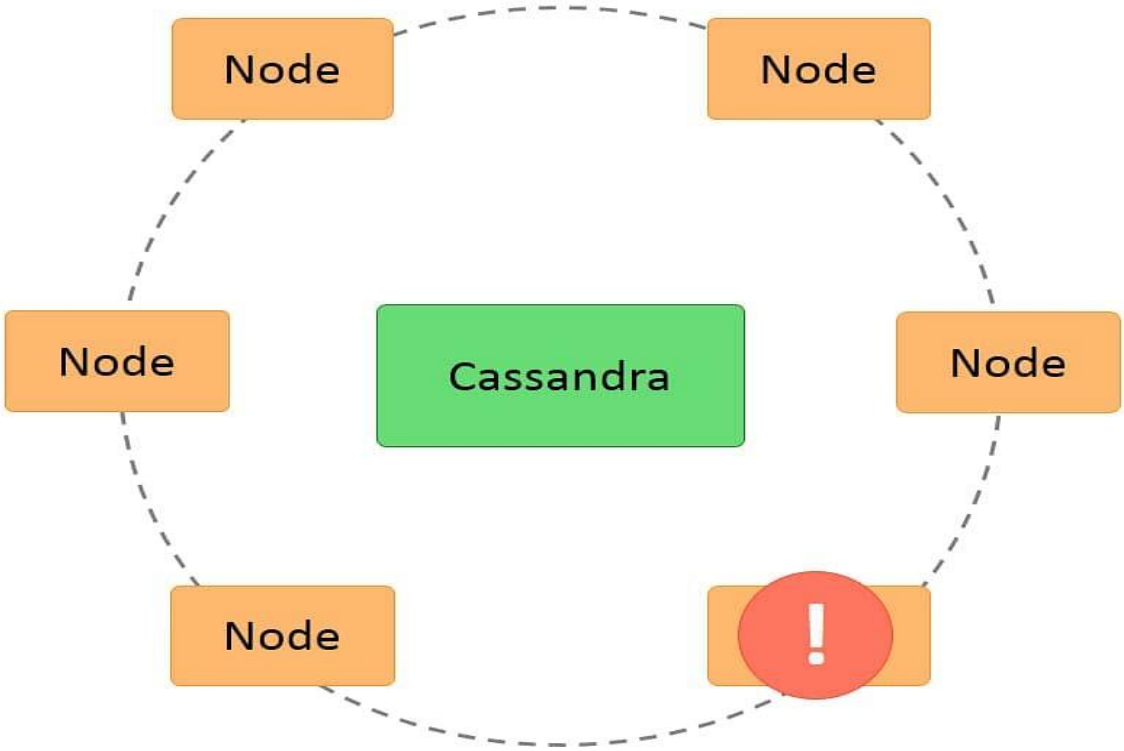
The example shows the token numbers being generated for 5 nodes in data center 1 and 4 nodes in data center 2. The first node always has the token value as 0. These token numbers will be copied to the Cassandra.yaml configuration file for each node.

Failure Scenarios: Node Failure

Cassandra is highly fault tolerant. The effects of node failure are as follows:

- Other nodes detect the node failure.
- Request for data on that node is routed to other nodes that have the replica of that data.
- Writes are handled by a temporary node until the node is restarted.
- Any memtable or sstable data that is lost is recovered from commitlog.
- A node can be permanently removed using the nodetool utility.

The following image shows the concept of node failure:



Failure Scenarios: Disk Failure

When a disk becomes corrupt, Cassandra detects the problem and takes corrective action. The effects of Disk Failure are as follows:

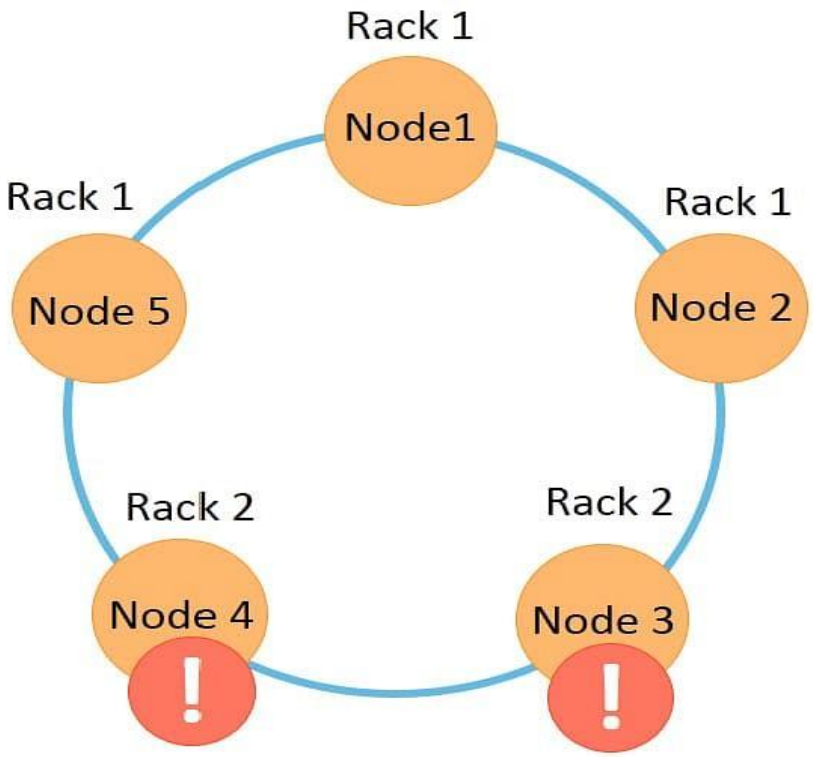
- The data on the disk becomes inaccessible
- Reading data from the node is not possible
- This issue will be treated as node failure for that portion of data
- Memtable and sstable will not be affected as they are in-memory tables
- Commitlog has replicas and they will be used for recovery

Failure Scenarios: Rack Failure

Sometimes, a rack could stop functioning due to power failure or a network switch problem. The effects of Rack Failure are as follows:

- All the nodes on the rack become inaccessible
- Reading data from the rack nodes is not possible
- The reads will be routed to other replicas of the data
- This will be treated as if each node in the rack has failed

The following figure shows the concept of rack failure:



Failure Scenarios: Data Center Failure

Data center failure occurs when a data center is shut down for maintenance or when it fails due to natural calamities. When that happens:

- All data in the data center will become inaccessible.
- All reads have to be routed to other data centers.
- The replica copies in other data centers will be used.
- Though the system will be operational, clients may notice slowdown due to network latency. This is because multiple data centers are normally located at physically different locations and connected by a wide area network.

Summary

Here’s a quick summary of the Apache Cassandra architecture tutorial:

- Cassandra has a ring-type architecture.
- Cassandra has no master nodes and no single point of failure.
- Cassandra supports network topology with multiple data centers, multiple racks, and nodes.
- Cassandra read and write processes ensure fast read and write of data.
- Cassandra partitions the data in a transparent way by using the hash value of keys.
- Replication in Cassandra can be done across data centers.
- Cassandra uses the gossip protocol for inter-node communication.
- Cassandra can handle node, disk, rack, or data center failures.