**Cassandra Architecture + Terminologies + Internal Working**

**CAP Theorem**

What is the CAP Theorem?

CAP theorem is also named as Brewer’s theorem, which was proposed by Eric Brewer. The distributed databases are based on the CAP theorem.

The theorem states that “**Though it’s desirable to have consistency, high availability and partition tolerance in every system, unfortunately, no system can achieve all three at the same time.**”

• **Consistency**: — All database clients will read the same value for the same query, even given concurrent updates.

• **Availability**: — All database clients will always be able to read and write data.

• **Partition** Tolerance: — The database can be split into multiple machines; it can continue functioning in the face of network segmentation breaks.

**OR**

CAP theorem is Consistency, Availability and partition tolerance. This theorem was proposed by Eric Brewer in 1999. CAP is basis of distributed databases.

CAP theorem is a concept that distributed database system can have two of above three. Not possible to guarantee all three concepts simultaneously in the distributed databases.

**Consistency**- Same data must be available in all nodes in the cluster at the same time.

**Availability**- Every request must get response whether it is succeeded or failed.

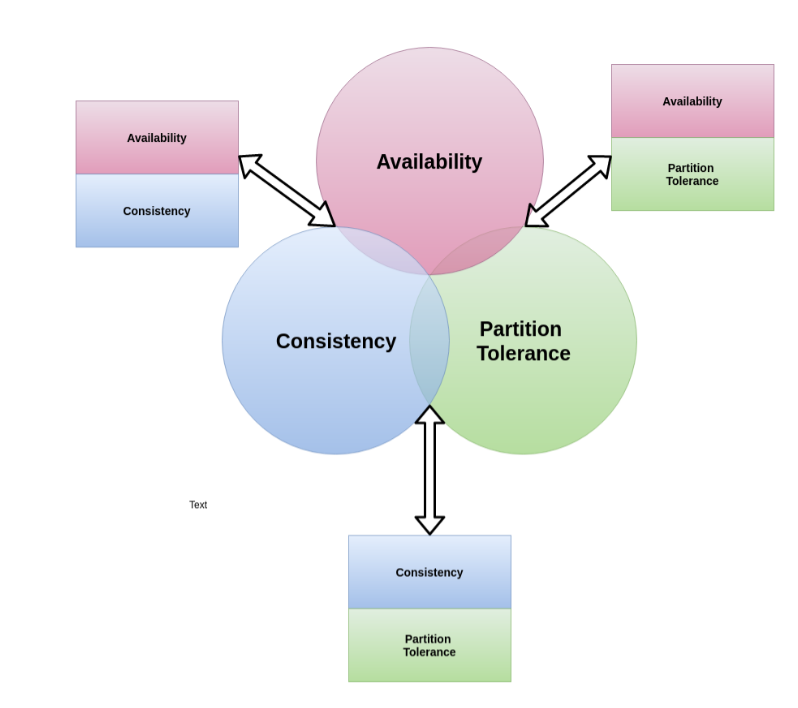
**Partition Tolerance**- The system continues to run without losing messages. If part of the system is failed, it is not affected to the entire system.

All the distributed databases must be come up with partition tolerance aspect. It is mandatory. Without partition tolerance, entire system is down.

When we consider Cassandra, it is mainly satisfied only Partition-tolerance and Availability. But we can improve consistency also.

**Consistency + Partition Tolerance- mongoDB, Reddis, Apache HBASE**

**Availability + Partition Tolerance- Cassandra**



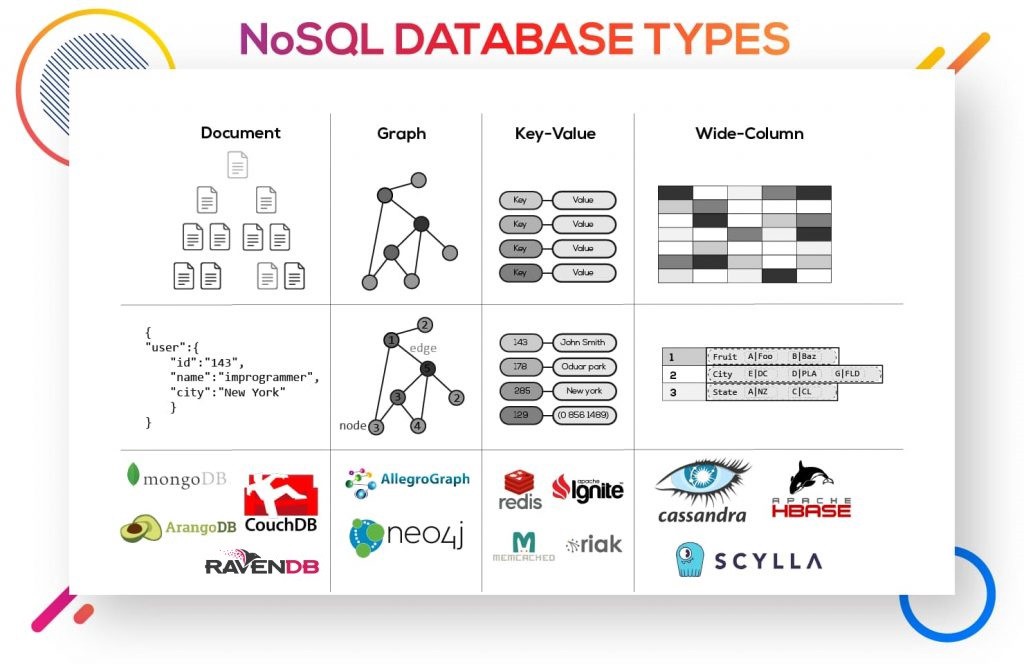
**No-SQL**

We often hear the term ‘non-relational database’ which is nothing but NoSQL. Some say the term “NoSQL” stands for “non-SQL” while others say it stands for “not only SQL.”

* NoSQL data models allow related data to be nested within a single data structure, so related data doesn’t have to be split between tables. They can easily process structured, semi-structured, and unstructured data.
* NoSQL database with a masterless, peer-to-peer architecture with all nodes being the same and guaranteed consistency to ensure constant availability. This offers easy scaling to adapt to the data volume on the applications. And also, Zero downtime because data will be distributed with multiple copies across different nodes.
* Instead of the ACID (Atomicity, Consistency, Isolation, and Durability) properties, NoSQL systems are said to have BASE (Basically Available, Soft state, and eventually consistent) properties.

**Types of NoSQL Databases**

1. **Document databases** store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. MongoDB has been considered as the world’s most popular Document database according to DB- engines.



1. **Key-value databases** where each item contains keys and values. A value can typically only be retrieved by referencing its value. Common use cases include storing user preferences or caching. Redis and DynanoDB are popular key-value databases.
2. **Wide-column databases** store data in tables, rows, and dynamic columns. Wide-column stores provide a lot of flexibility over relational databases because each row is not required to have the same columns. Commonly used for storing Internet of Things data and user profile data. Cassandra and HBase are two of the most popular wide-column stores.
3. **Graph databases** store data in nodes and edges. Commonly used when we need to traverse relationships to look for patterns such as social networks, fraud detection, and recommendation engines. Neo4j and JanusGraph are examples of graph databases.

Nodes → store information about people, places, and things

Edges → store information about the relationships between the nodes

**Basic data structure**

Cassandra is classified as a column-based database which means that its basic structure to store data is based on a set of columns which is comprised by a pair of column key and column value.

Every row is identified by a unique key, a string without size limit, called partition key. Each set of columns are called column families, similar to a relational database table.

**Columnar Database:**

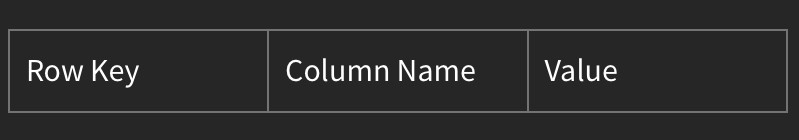
A database that stores data in multiple columns. It is similar to a two-dimensional key-value store. One of the most popular examples is Cassandra.

Cassandra was developed and open-sourced by Facebook and currently is being heavily used by the likes of Netflix, Apple, Spotify, and many more.

Columnar databases like Cassandra have the advantage that each column is stored in a separate file on disk, so if you query only certain columns, you will only need to read them instead of parsing the whole row with columns that are not part of the query. Furthermore, duplicate sequential data in the columns can be compacted to improve storage efficiency.

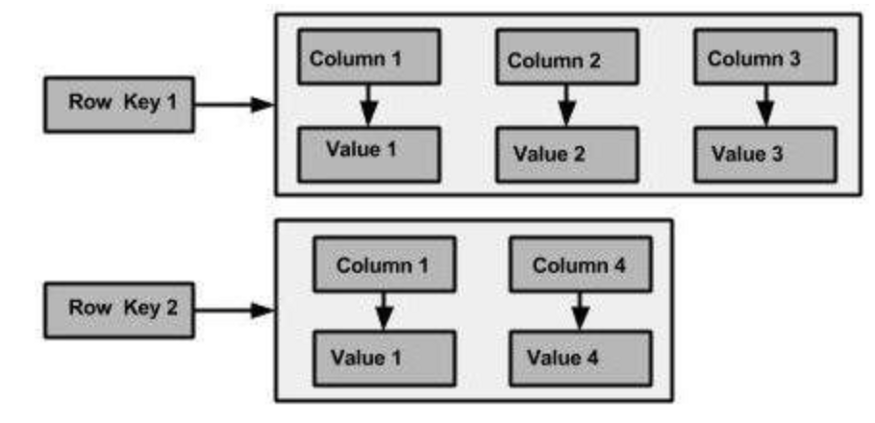
**Cassandra Data Model**

The smallest form of data stored in Cassandra can be visualized as:



1. Column: Name and Data Type
2. Column Family: List of columns with a primary key(can be one of the columns or a composite key). It has some attributes:

* key\_cached: Holds location of keys in memory.
* row\_cahed: Holds entire rows in memory.
* pre\_load\_row\_cache: Option to preload row cache



1. Super Column Family: It is a subset of the column family of a row. It can be used to bind rows which are used frequently, but:

* Cassandra doesn’t index columns in the super column family.
* Out of memory error could happen.

1. One column family or bunch of column families becomes a keyspace.

Key Space is the outermost place in the Cassandra datastore where we define replication and replica placement strategy.

**What’s Apache Cassandra?**

Apache Cassandra is a free, open-source, NoSQL, distributed data storage system, so instead of the database only being able to live on one server it can be spread across multiple ones, this allows the database to grow almost infinitely.

It’s used to create databases that are spread across nodes in more than one data center for high availability.

Cassandra is being used in production by some of the biggest companies on the web, including Facebook, Twitter, and Netflix, for various use cases: Sensor data, messaging, gaming, fraud detection, location-based services …

OR

Apache Cassandra is an open-source, distributed, NoSQL database. It presents a partitioned wide column storage model with eventually consistent semantics.

Let's start with the basic components of Cassandra, so that makes our learning easy and interesting.

Node: The data will get stored in a place called Node. We can say in other words, a single Cassandra instance is called a node.

Datacenter: The group of nodes is called a Datacenter.

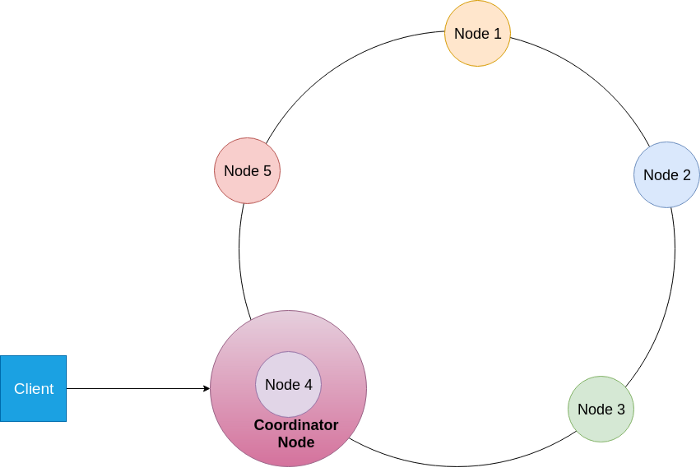
Cassandra can scale both horizontally (adding more datacenters) or vertically (adding more nodes).

Cluster: The group of Datacenters is called a cluster.

**What is the coordinator?**

When Cassandra client is hit a write and read request on the node in the cluster. That node is called as coordinator. According to the below picture, coordinator node may be changed every time.

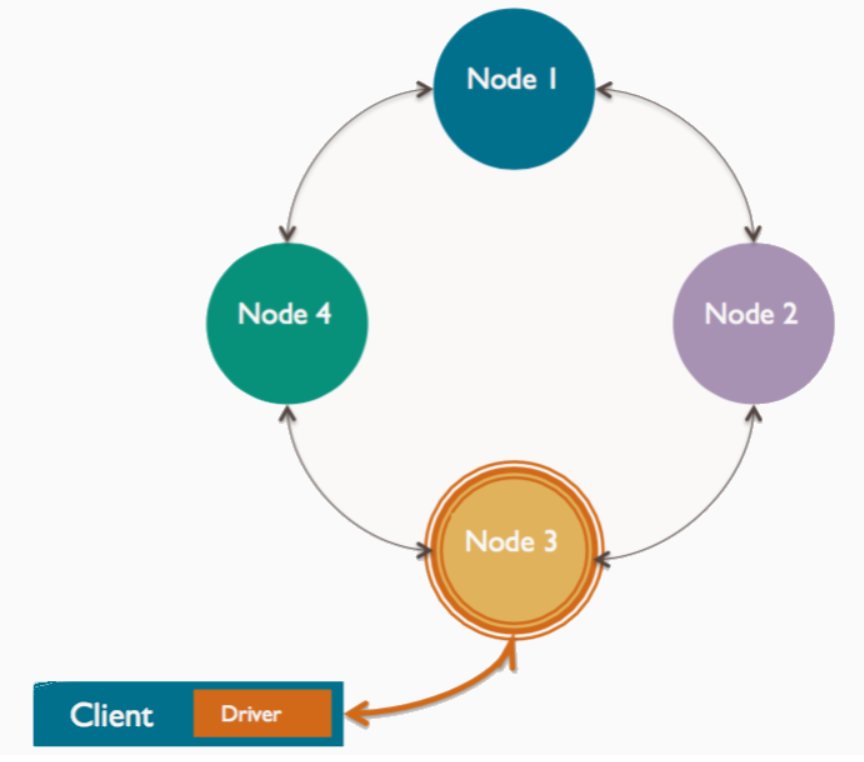
Coordinator is selected by the cassandra driver based on the policy, you have set. Most common policies are **DCAwareRoundRobinPolicy** and **TokenAwarePolicy**.



**OR**

The node is chosen by the client to receive a particular read or write request to its cluster.

* Any node can coordinate any request
* Each client request may be coordinated by a different node
* The coordinator manages the Replication Factor — onto how many nodes should write be copied?
* The coordinator also applies the Consistency Level — how many nodes must acknowledge a read or write request.



**Data Replication**

**Apache Cassandra’s structure is “built-for-scale” and can handle large amounts of data and concurrent users across a system. With no single point of failure, the system offers true continuous availability, avoiding downtimes and data loss by Data Replication.**

**Data replication is defined per keyspace in terms of**

**→ replication factor per data center**

**→ the replication strategy**

**Replication Factor**

**The total number of replicas placed on different nodes is determined by the replication factor.**

**1 replication factor = only a single copy of data**

**3 replication factor = three copies of the data on three different nodes**

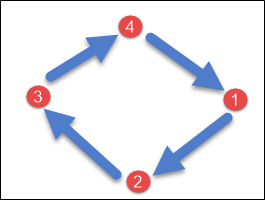
**The remainder of the replicas is placed by Cassandra on specific nodes using a *replica placement strategy*.**

**Replication Strategy**

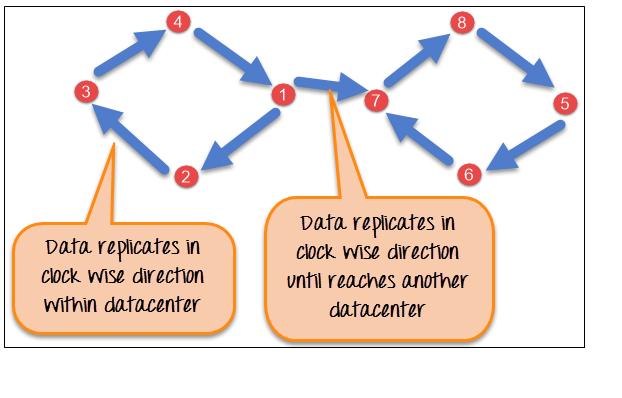
**Basically, the coordinator uses the Replication Strategy to find out which nodes will be the replica nodes for a given request.**

**There are two replication strategies available:**

**SimpleStrategy: used for a single data center deployment (not recommended for production environment). It doesn’t consider the network topology. Basically, it just takes the partitioner’s decision (that is, the node that will handle the request first based on the token range) and places the remaining replicas clockwise in relation to this node. For example, in Figure 3, if the table replication factor was 3, which nodes would have been chosen by the SimpleStrategy to act as replicas (besides node2, which was already chosen by the partitioner)? That’s correct, node3 and node4! What if the replication factor was 4? Well, then node5 would also be included.**



**NetworkTopologyStrategy: used for multiple data centers deployment (recommended for production environment). It also takes the partitioner’s decision and places the remaining replicas clockwise, but it also takes into consideration the rack and data centers configuration.**

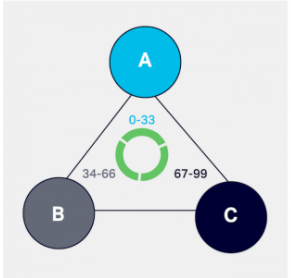


**Data Partitioning**

**Cassandra organizes data into partitions. This is a common concept of distributed data systems. All the data is distributed into chucks called partition. Partitioning is very important for performance and scalability.**

**Basically, for each node in the Cassandra cluster (Cassandra ring) is assigned a range of tokens.**

**The *partitioner* is the factor responsible for deciding how the data can be spread across cluster nodes. Cassandra distributes data across the cluster using a *Consistent Hashing algorithm*, given the partition key of a row.**



**Let’s consider a simple example: suppose a request is issued to node B (that is, node B is the coordinator for this request) with a row containing the partition key “data science”. Suppose the partitioner applies the hash function to the partition key “data science and gets the token 87. Node C token ranges include 87, so this node will be the one handling the request.**

**Cassandra offers three partitioners:**

* **Murmur3Partitioner (default)**
* **RandomPartitioner**
* **ByteOrderedPartitioner**

**Virtual nodes**

* **There is another concept implemented as virtual nodes (vnodes), where each node owns a large number of small token ranges in order to improve token reorganization and avoid hotspots in the cluster, that is, some nodes storing much more data than the others.**
* **Virtual nodes also allow to add and remove nodes in the cluster more easily and manages the token assignment automatically for you so that you can enjoy a nice coffee when adding or removing a node instead of calculating and assigning new token ranges for each node (which is a very error-prone operation, by the way).**

**Cassandra Keys**

***“A primary key in Cassandra consists of one or more partition keys and zero or more clustering key components. The order of these components always puts the partition key first and then the clustering key”.***

**The primary key consists of two parts:**

1. **Partition Key: It is the key that determines which on which node our data would lie:**

* **Partition Key is hashed with a hash function and the token thus formed decides on which node (having the token range) data would lie.**
* **Rows with the same primary key will lie on the same node, hence while querying it is essential to specify the entire partition key.**

***“The partition key is responsible for distributing data among nodes. In Cassandra, we can only access data from the partitioning key”.***



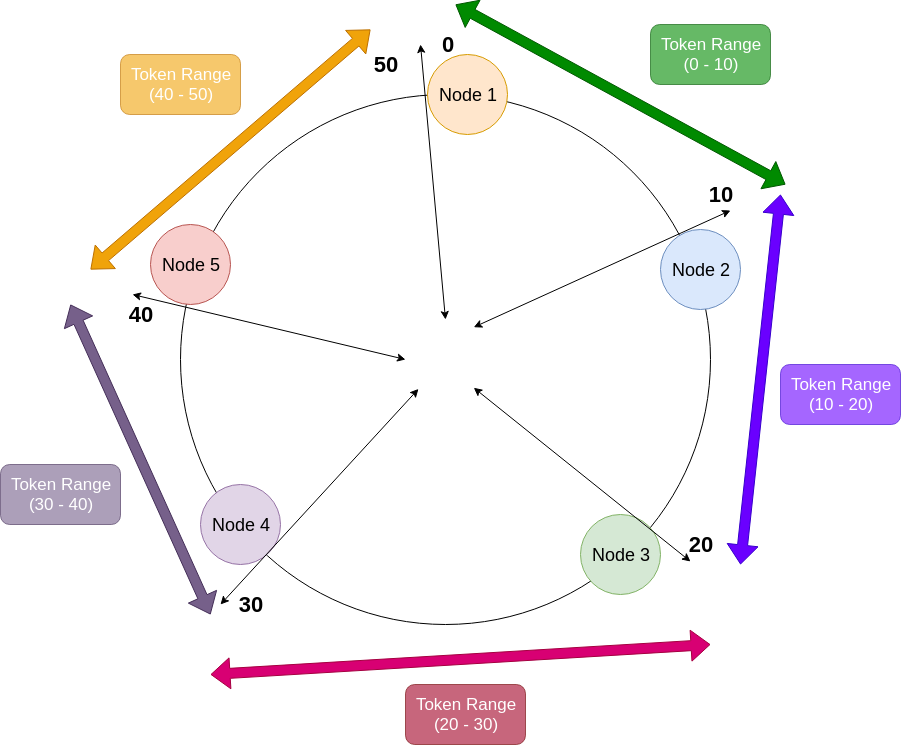
**In above table, Car name is a partitioning key. When data is coming to the cassandra, it gets token of that incoming data by providing partitioning key to the hash function.**

**BMW -> Hash Function -> 9**

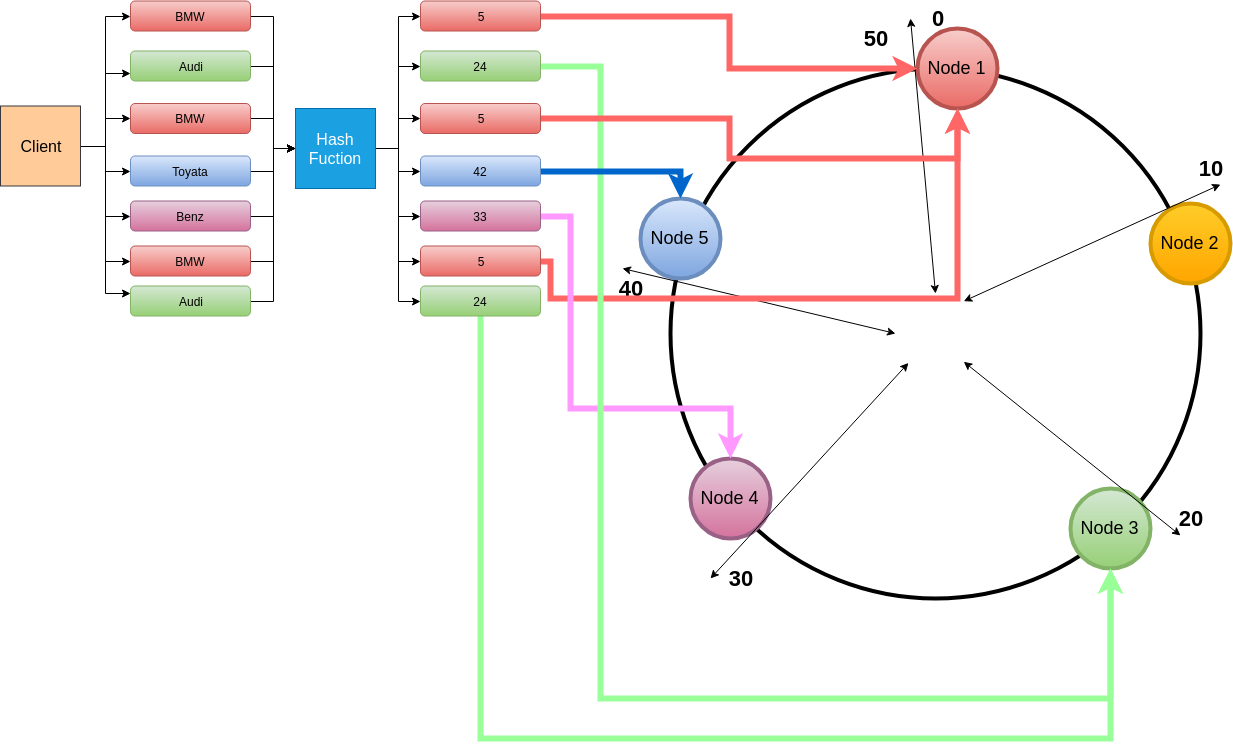
**Toyata -> Hash Function -> 17**

**Audi -> Hash Function -> 31**

**Benz -> Hash Function -> 25**



**below diagram is displayed how the data is distributed among the cluster by using partition key.**



1. **Clustering Key: It is the order in which data is sorted amongst the node with the same partition key. Hence range queries are possible here.**

* **Clustering columns order data within a partition. Each primary key column after the partition key is considered a clustering key.**
* **The database uses the clustering information to identify where the data is within the partition.**

**Example:**

**CREATE TABLE application\_logs (**

**Id INT,**

**app\_name VARCHAR,**

**hostname VARCHAR,**

**log\_datetime TIMESTAMP,**

**env VARCHAR,**

**log\_level VARCHAR,**

**log\_message TEXT,**

**PRIMARY KEY ((app\_name, env), hostname, log\_datetime)**

**);**

**Here (app\_name, env) = composite partition key**

**hostname, log\_datetime = clustering key**

**Note: For querying using any other column (than the primary key) we need to make it a secondary index. It does that with SS Table Attached Secondary Index (SASI).**

**//Write Operation**

**//Read Operation**

**Snitch**

**The job of a snitch is to determine which data centers and racks are to be written to and read from (relative host proximity/host is relatively nearer).**

**Snitch gathers information about topology (simple Strategy or Network Topology Strategy).**

**There are several types of snitches. Depending on Snitches, It can efficiently route the read and write requests among the cluster.**

1. **Simple Snitch**
   1. **This is the default Snitch in Cassandra. Simple Snitch assumes all the nodes are in same data center and same rack (only for single data center).**
2. **Property File Snitch**
   1. **This Snitch is using cassandra-topology.properties file. That snitch can be used to determine proximity by using data center and rack.**

**Every nodes in the cluster must be listed in the casssandra-topology.properties file.**

**This cassandra-topology.properties file must be same on every node in the cluster.**

**# datacenter One**

**175.56.12.105=DC1:RAC1**

**175.50.13.200=DC1:RAC1**

**175.54.35.197=DC1:RAC1**

**120.53.24.101=DC1:RAC2**

**120.55.16.200=DC1:RAC2**

**120.57.102.103=DC1:RAC2**

**# datacenter Two**

**110.56.12.120=DC2:RAC1**

**110.50.13.201=DC2:RAC1**

**110.54.35.184=DC2:RAC1**

**50.33.23.120=DC2:RAC2**

**50.45.14.220=DC2:RAC2**

**50.17.10.203=DC2:RAC2**

**If we add new node into the cluster we must add it as well into that cassandra-topology.properties file by manually.**

1. **Gossiping Property File Snitch**

**This snitch is using file called cassandra-rackdc.properties. Every node has this file and we define rack and data center information in that property file.**

**dc=DC1**

**rack=RAC1**

**Those mentioned information of nodes are distributed among other nodes via gossiping. datacenter and rack names are case-sensitive.**

1. **Dynamic Snitch**
2. **Rack Inferring Snitch**

**Gossip Protocol**

**gossip is communication protocol. It is exchanged all the information about themselves and other nodes in the cluster. This process runs in every second. So all the nodes quickly know about status of other nodes.**

**What kind of data is exchanged between nodes,**

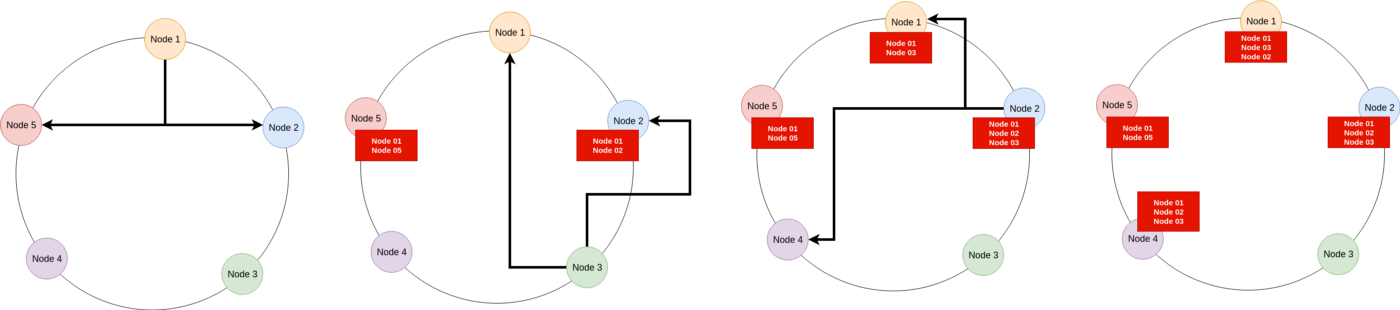
**Heartbeat State-**

* **When the node started.**
* **When gossiping session sent.**

**Application State-**

* **Node status (Normal/Leaving/Joining)**
* **Data center**
* **Rack**
* **Schema**
* **Load(Performance of the node)**
* **Severity(I/O)**

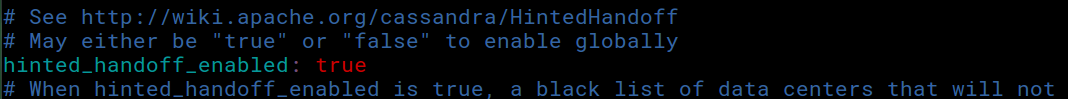
**Load and Severity give us good indication of current performance of the node.**



**Hinted Handoff**

**When a node becomes down or unresponsive, hinted handoff allows Cassandra to continue its write operation without any problems.**

**Now we can enabled and disabled hinted handoff by using cassandra.conf.**



**when node is down, all the writes data which belongs to that down node key range are stored for a period of time.**

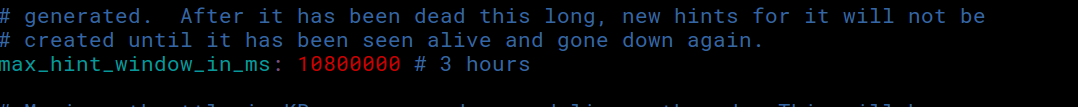
**The hint consists as below,**

* **target Id- downed node**
* **hint Id- time uuid**
* **message id- cassandra version**
* **data is stored as a blob.**

**Hints are flushed to disk every 10 seconds.**

**When gossip knows that down node is up and running again, all the remaining hint are written to the new node. After that hint file is deleted.**

**There is an another configuration called “max\_hint\_window\_in\_ms” in cassandra.conf file.**



**If node is down for longer than max\_hint\_window\_in\_ms, stops writing new hints.**

**Tombstones (Soft delete)**

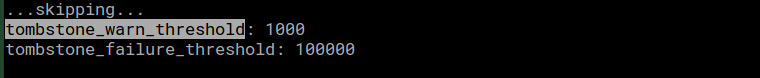
**Cassandra is not deleted data from the disk immediately. If Cassandra do it, it takes lots of time. That is why Cassandra comes up with Tombstones. Tombstones are a mechanism which allows Cassandra to write fast.**

**So Cassandra use marker as special value called tombstones to determine which data is deleted.**

**Tombstones prevent deleted data from being returned during reads.**

**Tombstone is generated by,**

* **DELETE statement**
* **TTL (time-to-live)**
* **INSERT or UPDATE with null values**
* **Update with collection column**



**tombstone\_warn\_threshold - Cassandra will display a warning, if the scanned tombstone count is exceeded tombstone\_warn\_threshold by a query.**

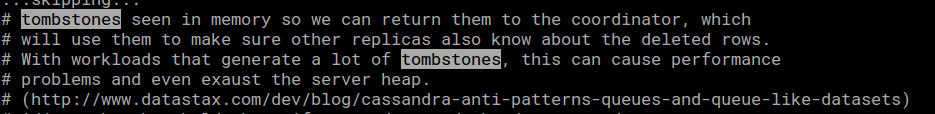
**tombstone\_failure\_threshold - Cassandra abort the query, if the scanned tombstone count is exceed tombstone\_failure\_threshold value by a query.**

**There is a setting called Garbage Collection Grace Seconds(gc\_grace\_seconds). This is the amount of time that the server will wait to garbage-collect a tombstone. default value is 10 days(864000 seconds). Tombstones will be dropped during compaction after gc\_grace\_second has passed.**

**Tombstones will not be removed until a compaction event even if gc\_grace\_seconds has elapsed.**

**Problems of Tombstones**

* **This is how cassandra.yaml is described about a lots of tombstones.**



**Lot of tombstones are come up with performance problems and server heap.**

* **Cassandra abort the query, if the scanned tombstone count is exceed tombstone\_failure\_threshold value by a query.**
* **cluster is rapidly filling up.**

**Compaction**

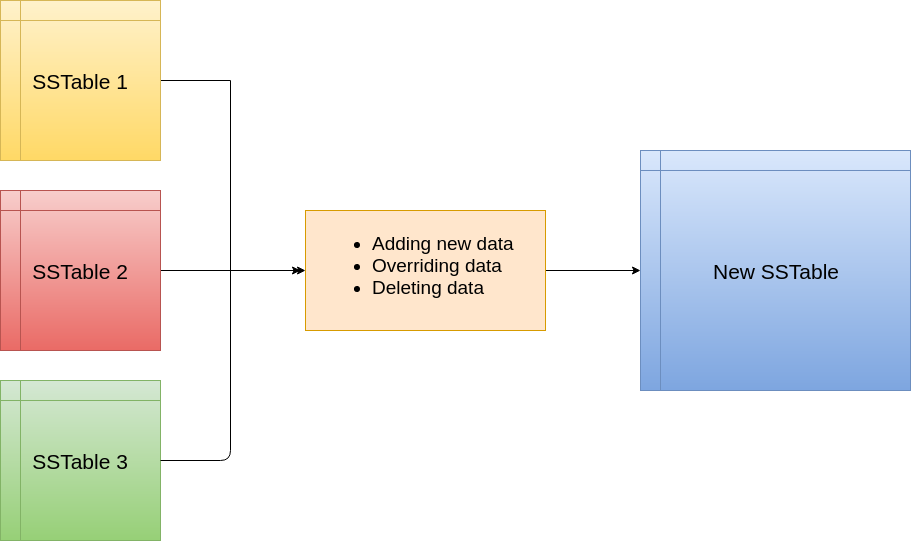
**SSTables are immutable. Mutations, adding new data, updating data, deleting data are inserted into memtable. Always adding new record to the memtable when doing above mentioned operations. After that memtable is periodically flushed to the different SSTables.**

**When we consider update operation in Cassandra, there may be old value and new value in different SSTables or same SSTable. In that case Cassandra is using timestamps to figure out which is the most recent value. In here Cassandra is using lots of disk space. In this case, we are trying to read some data from the Cassandra, query might need to read from several SSTables to get a result. So read operation may be slow. That is why Cassandra needs a operation called compaction.**

**Compaction is doing read all the existing SSTables and merge all the rows with most recent information into the one SSTable. Basic idea of compaction is created new SSTable instead of existing SSTables.**

**As we discussed earlier, SSTables are immutable. If Cassandra need to update an existing row, it will add another new row in same SSTable or different SSTable.**

**When doing compaction, all the tombstones are permanently removed.**



**There are several compaction strategies.**

1. **Size Tiered Compaction Strategy- default one**
2. **Leveled Compaction Strategy**
3. **DateTiered Compaction Strategy**
4. **Time Window Compaction Strategy**

**Bloom Filter**

**Bloom filter is a data structure. It is used to test whether an element is a member of a set. False positive matches are possible and false negative are not. This is extremely very fast.**

**Cassandra is using bloom filter to check whether requested partition key is available in any of the SSTables without reading existing data of SSTables. By using blooom filter, we can avoid expensive I/O operations.**

**There are corresponding bloom filers in memory per each SSTables.**

