**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.

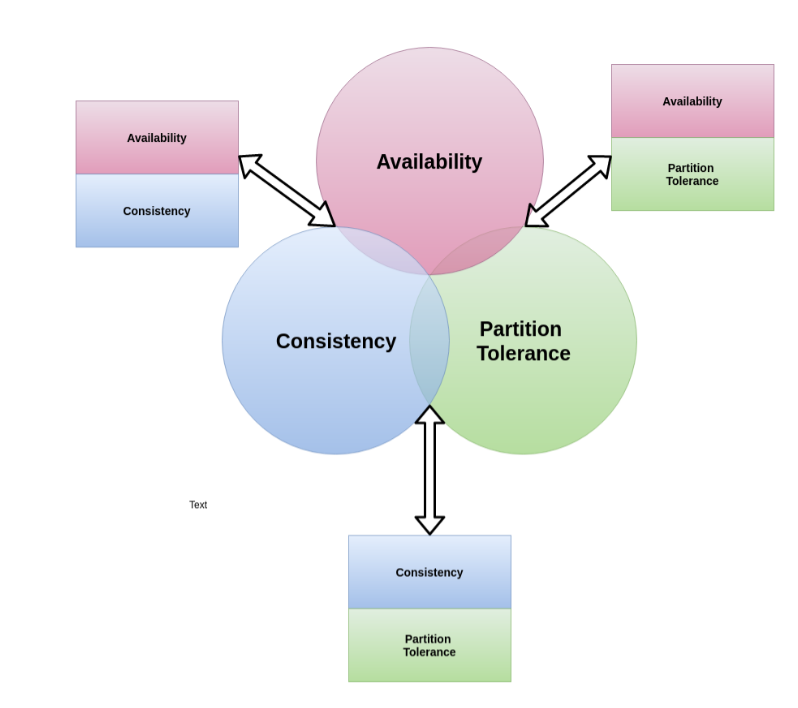
**Partition Tolerance**- The system continues to run without losing messages. If part of the system is failed, it is not affected to the entire system.

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**



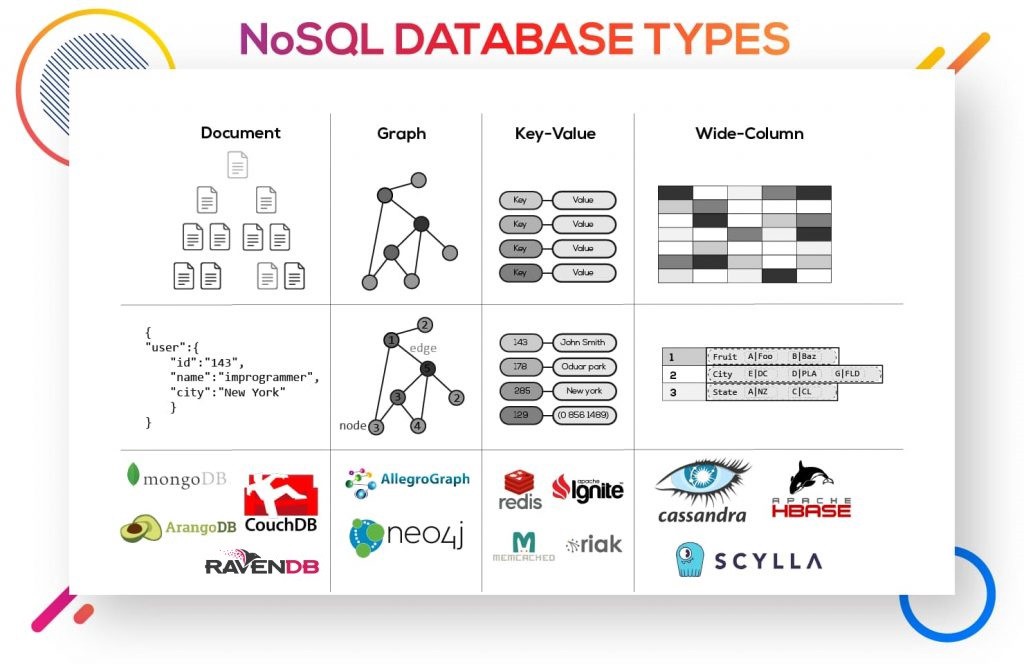
**No-SQL**

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:**

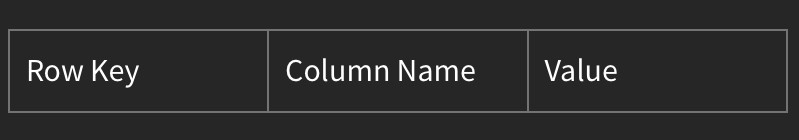
A database that stores data in multiple columns. It is similar to a two-dimensional key-value store. One of the most popular examples is Cassandra.

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.

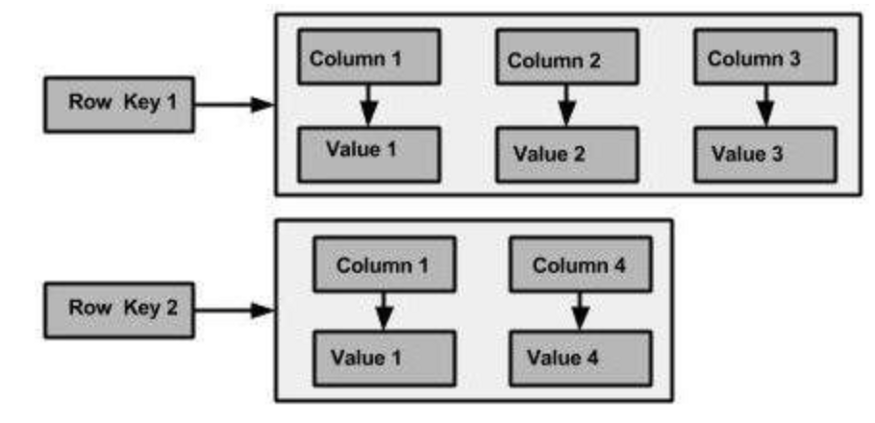
**Cassandra Data Model**

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.

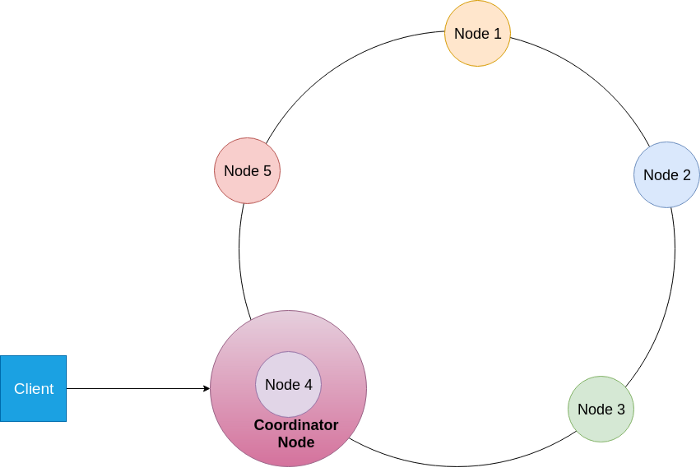
Cassandra can scale both horizontally (adding more datacenters) or vertically (adding more nodes).

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.

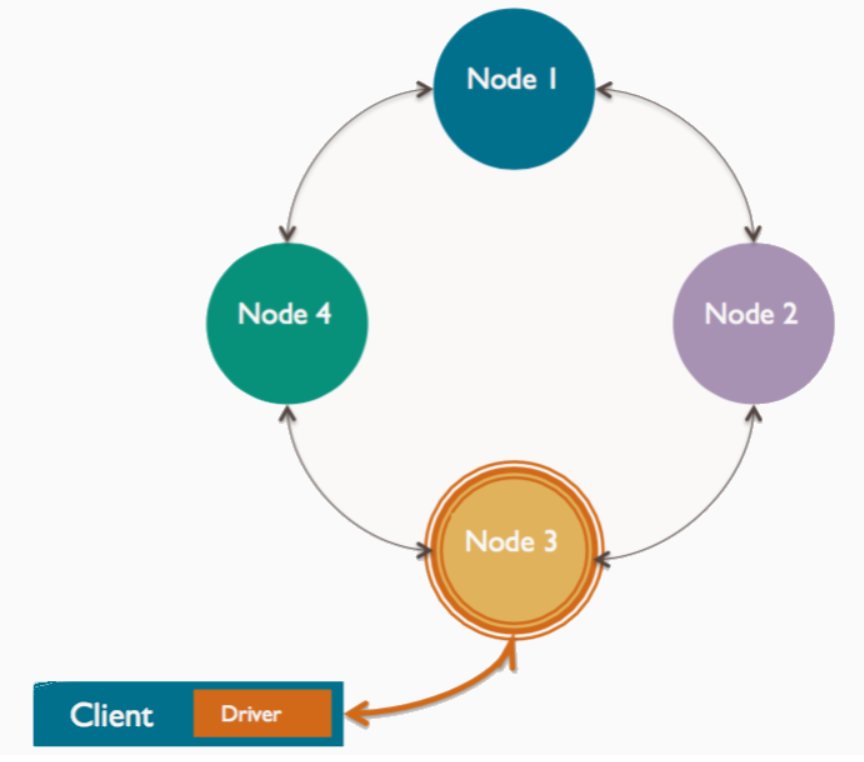
Coordinator is selected by the cassandra driver based on the policy, you have set. Most common policies are **DCAwareRoundRobinPolicy** and **TokenAwarePolicy**.



**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**

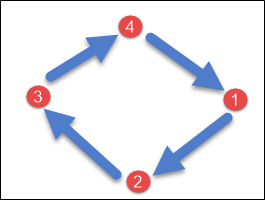
**The remainder of the replicas is placed by Cassandra on specific nodes using a *replica placement strategy*.**

**Replication Strategy**

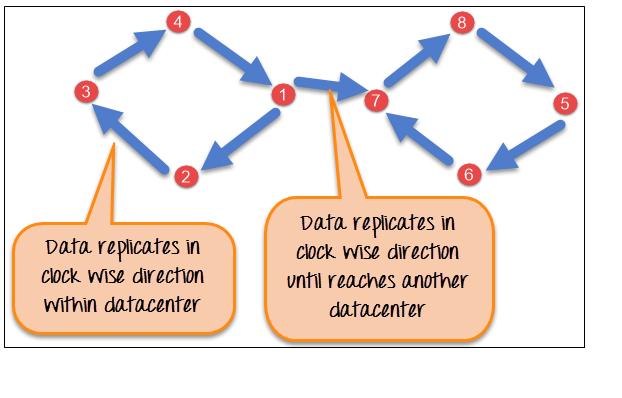
**Basically, the coordinator uses the Replication Strategy to find out which nodes will be the replica nodes for a given request.**

**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.**

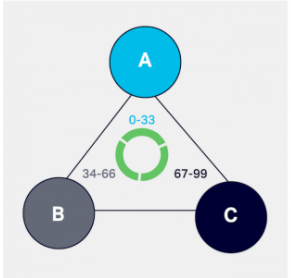


**Data Partitioning**

**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**

**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.**

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.**

**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).**