What is Apache Cassandra?

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra −

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon’s Dynamo and its data model on Google’s Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful “column family” data model.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- **Elastic scalability** − Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- **Always on architecture** − Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- **Flexible data storage** − Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- **Easy data distribution** − Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.

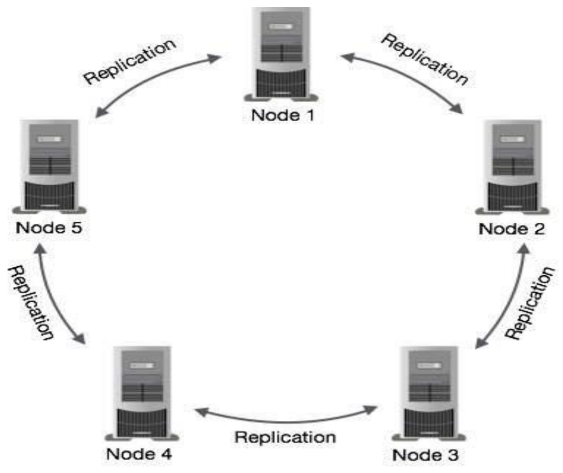
Cassandra - Architecture

Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- All the nodes in a cluster play the same role. Each node is independent and at the same time interconnected to other nodes.
- Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- When a node goes down, read/write requests can be served from other nodes in the network.

Data Replication in Cassandra

In Cassandra, one or more of the nodes in a cluster act as replicas for a given piece of data. If it is detected that some of the nodes responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a **read repair** in the background to update the stale values.



The following figure shows a schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.

Note − Cassandra uses the **Gossip Protocol** in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.

Components of Cassandra

The key components of Cassandra are as follows −

- **Node** − It is the place where data is stored.
- **Data center** − It is a collection of related nodes.
- **Cluster** − A cluster is a component that contains one or more data centers.
- **Commit log** − The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- **Mem-table** − A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable** − It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter** − These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Difference between Cassandra and RDBMS:

CASSANDRA	RDBMS
Cassandra is a high performance and highly scalable distributed NoSQL database management system.	RDBMS is a Database management system or software which is designed for relational databases.
Cassandra is a NoSQL database.	RDBMS uses SQL for querying and maintaining the database.
It deals with unstructured data.	It deals with structured data.
It has a flexible schema.	It has fixed schema.
Cassandra handles high volume incoming data velocity.	RDBMS handles moderate incoming data velocity.
In Cassandra the outermost container is Keyspace.	In RDBMS the outermost container is database.
Cassandra follows decentralized deployments.	RDBMS follows centralized deployments.
In Cassandra, relationships are represented using collections.	In RDBMS relationships are represented using keys and join etc.

1. What are clusters in Cassandra? What is a YAML file in Cassandra?

Clusters in Cassandra:

- A cluster in Cassandra refers to a group of nodes that work together to store data and serve client requests. Cassandra is designed to be distributed, meaning data is partitioned and replicated across multiple nodes in a cluster for fault tolerance and scalability.
- Each node in a Cassandra cluster communicates with other nodes using a peer-to-peer protocol, sharing information about data distribution, replication, and membership within the cluster.

- Clusters can span multiple data centers and geographic regions, providing high availability and resilience against failures.

YAML File in Cassandra

- In Cassandra, configuration settings are often specified in YAML (YAML Ain't Markup Language) files. YAML is a human-readable data serialization format commonly used for configuration files.
- The main YAML file in Cassandra is typically named `cassandra.yaml`. This file contains various configuration options that control the behavior of the Cassandra server, such as:
  - Cluster and node settings: Including parameters related to the node's role within the cluster, like its name, IP address, and data directory.
  - Memory and storage configuration: Options for configuring memory usage, disk settings, and caching.
  - Security settings: Configuration for authentication, authorization, encryption, and other security-related features.
  - Performance tuning: Settings for tuning Cassandra's performance based on hardware resources and workload characteristics.
- Administrators can modify the `cassandra.yaml` file to customize the Cassandra deployment according to their specific requirements and environment.

2. **What are partitions and tokens in Cassandra? Explain**

Partitions

- A partition in Cassandra is a unit of data organization within a table. It represents a subset of data that shares the same partition key. Each partition is stored on a specific node in the cluster.

- The partition key is a primary component of the data model in Cassandra. It determines how data is distributed across the cluster and is used to locate the appropriate node for reading or writing data.

- When data is inserted into a Cassandra table, it is partitioned based on the partition key. Cassandra's partitioning scheme ensures that data with the same partition key is stored together, making it efficient to retrieve all related data in a single query.

- Partitions can vary in size depending on the amount of data associated with a particular partition key. Cassandra automatically manages the distribution of partitions across nodes to achieve load balancing and fault tolerance.

Tokens:

- In Cassandra, tokens are used to determine the placement of data within the cluster. Each node in the cluster is assigned a range of tokens that define its position in the token ring, which is a virtual continuum of all possible token values.

- Tokens are generated based on the partition key values using a hashing algorithm, such as Murmur3 or MD5. This deterministic process ensures that similar partition keys map to nearby positions on the token ring.

- When data is inserted into Cassandra, the partition key is hashed to determine its corresponding token value. The token value is then used to determine which node in the cluster will store the data.

- By distributing data based on tokens, Cassandra achieves a decentralized architecture where data is evenly spread across nodes in the cluster. This approach provides fault tolerance and scalability, as data can be replicated and distributed across multiple nodes.

Keyspace

In Apache Cassandra, a keyspace is a high-level logical container for data. It serves as a namespace that defines how data is organized within the database. Keyspaces in Cassandra are analogous to databases in traditional relational database management systems (RDBMS).

Given below is the syntax for creating a keyspace using the statement **CREATE KEYSPACE**.

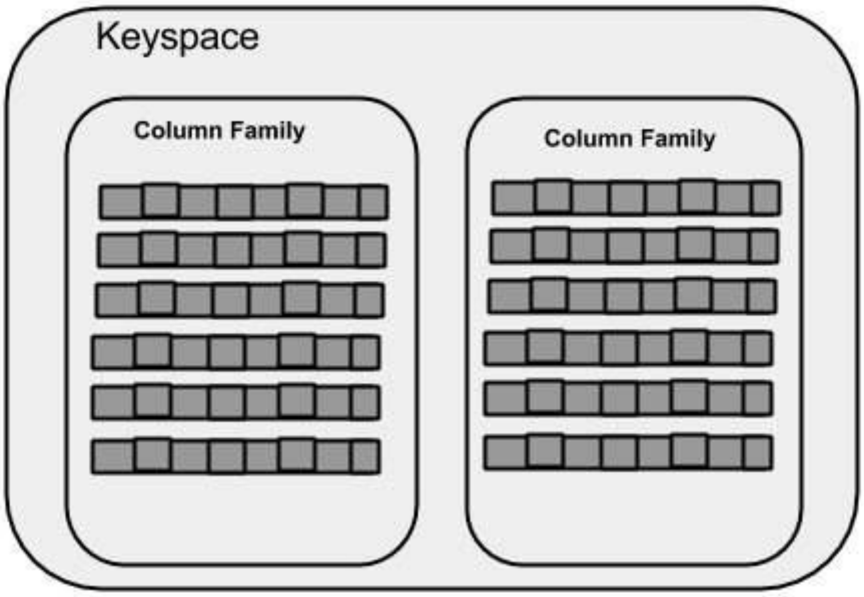
Syntax

CREATE KEYSPACE <identifier> WITH <properties>

=

```
CREATE KEYSPACE tutorialspoint
WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};
```

The following illustration shows a schematic view of a Keyspace.



Replication

The replication option is to specify the **Replica Placement strategy** and the number of replicas wanted. The following table lists all the replica placement strategies.

Strategy name	**Description**
Simple Strategy	Specifies a simple replication factor for the cluster.
Network Topology Strategy	Using this option, you can set the replication factor for each data-center independently.
Old Network Topology Strategy	This is a legacy replication strategy.

Using this option, you can instruct Cassandra whether to use **commitlog** for updates on the current KeySpace. This option is not mandatory and by default, it is set to true.

Cassandra Query Language

CQL stands for Cassandra Query Language. It is the primary language used to interact with Apache Cassandra databases. CQL is designed to resemble SQL (Structured Query Language) in its syntax, making it relatively easy for developers familiar with SQL to work with Cassandra.

Here are some key aspects of CQL:

Data Definition Language (DDL):CQL provides commands for defining and managing database schema elements such as keyspaces, tables, columns, indexes, and user-defined types.

Example:

CREATE KEYSPACE my_keyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};

CREATE TABLE my_keyspace.my_table (id UUID PRIMARY KEY, name TEXT);

Data Manipulation Language (DML): CQL supports commands for inserting, updating, deleting, and querying data in Cassandra tables.

Example:

INSERT INTO my_keyspace.my_table (id, name) VALUES (uuid(), 'John');

SELECT * FROM my_keyspace.my_table WHERE id = uuid();

Data Control Language (DCL): CQL allows managing permissions and access control for users and roles in Cassandra.

Example:

CREATE ROLE my_role WITH PASSWORD = 'password' AND LOGIN = true;

```
GRANT ALL PERMISSIONS ON KEYSPACE my_keyspace TO my_role;
```

Data Query Language (DQL): CQL supports a wide range of query operations, including filtering, sorting, aggregating, and paging data.

Example:

```
SELECT * FROM my_keyspace.my_table WHERE name = 'John' ALLOW FILTERING;
```

Batch Statements: CQL enables executing multiple DML statements as a single batch for atomicity and performance.

Example:

```
BEGIN BATCH

INSERT INTO my_keyspace.my_table (id, name) VALUES (uuid(), 'Alice');

DELETE FROM my_keyspace.my_table WHERE id = uuid();

APPLY BATCH;
```

CQL simplifies the process of interacting with Cassandra databases, allowing developers to define schemas, manipulate data, and perform queries using a familiar SQL-like syntax.