








# INTRODUCTION TO CASSANDRA

# History of the Cassandra Database

- Cassandra was born out of the need for a massive, globally-distributed, always-on, highly available database that could scale to the size of modern web applications and social media.
- Avinash Lakshman, who was co-inventor of Amazon's Dynamo database collaborated with Prashant Malik to develop the initial internal database, which was first announced to the world in the 2009 Cassandra whitepaper.

- 
- 
- While it drew from many of Dynamo's design principles, Cassandra also adopted other features similar to Google's 2006 [Bigtable](#) [whitepaper](#).
  - Cassandra was designed to handle the "Inbox Search" problem for Facebook.
  - Eventually, Facebook replaced Cassandra with Hbase for their Inbox Search project, but they continue to use Cassandra in their Instagram division.


- 
- 
- Cassandra is a popular NoSQL database.
  - Apache Cassandra was born at Facebook.
  - It does not follow master slave architecture.
  - It is a highly scalable, high performance distributed database.
  - It distributes and manages gigantic volumes of data across commodity servers.
  - It is a column oriented database designed to support peer-to-peer symmetric nodes



- 
- 
- A few companies that have deployed Cassandra and have benefitted immensely include
    - Twitter
    - NetFlix
    - Cisco
    - Adobe
    - eBay

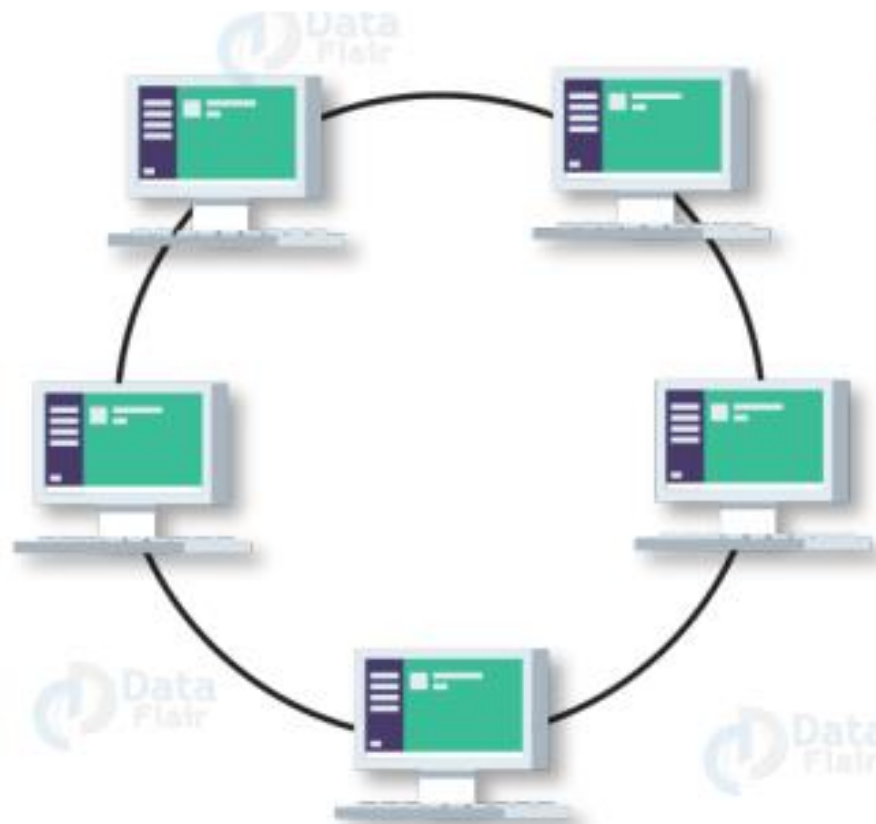


# Features of Cassandra:

## Peer-to-peer Network:

- A node in Cassandra is structurally identical to any other node.
  - It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.
  - In Cassandra, each node is independent and at the same time interconnected to other nodes.
- 

- 
- 
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
  - In the case of failure of one node, Read/Write requests can be served from other nodes in the network.





**peer to peer**





## **Gossip and failure detection:**

- The communication between nodes is often like peer-to-peer communication, where every node talks to the other.
  - In Cassandra, when one node talks to another, the node which is expected to respond, not only provides information about its status, but also provides information about the nodes that it had communicated with before.
- 

- 
- An important feature of Gossip Protocol is Failure Detection.
  - When two nodes communicate with one another; for instance, Node A to Node B,
  - then Node A sends a message 'gossipdigestsynmessage', which is very similar to TCP protocol to Node B.
  - Here, Node B, once receives the message, sends an acknowledgement message 'ack',
  - and then Node A responds with an acknowledgement message to Node B's 'ack' message. This is known as the 3 way handshake.




## Partitioner:

- A partitioner takes a call on how data is distributed on the various nodes in a cluster.
- It also determines the node on which the first copy of data is placed.




## Replication factor:

- Replication factor determines the number of copies of data that will be stored across nodes in a cluster.
  - If one wishes to store only one copy of data of each row on one node then the replication factor is set to 1. The number of copies of each row of data determines the replication factor. However, the replication factor should be more than one but less than the number of nodes in a cluster.
- 




# What is CQL?

- Cassandra CQL, a simple alternative to Structured Query Language (SQL), is a declarative language developed to provide abstraction in accessing Apache Cassandra.
- 

# Basic Cassandra (CQL) constructs

- **Keyspace** — Similar to an RDBMS database, a keyspace is a container for application data that must have a name and a set of associated attributes.
- **Column Families/Tables** — A keyspace consists of a number of Column Families/Tables. A Cassandra column family is a SQL table.
- **Primary Key / Tables** — A Primary Key consists of a Row/Partition Key and a Cluster Key, and functions to enable users to uniquely identify internal rows of data. A Row/Partition Key determines the node on which data is stored.

- 
- The Partition Key is responsible for data distribution accross your nodes.
  - The Clustering Key is responsible for data sorting within the partition.
  - The Primary Key is equivalent to the Partition Key in a single-field-key table.
  - The Composite/Compund Key is just a multiple-columns key

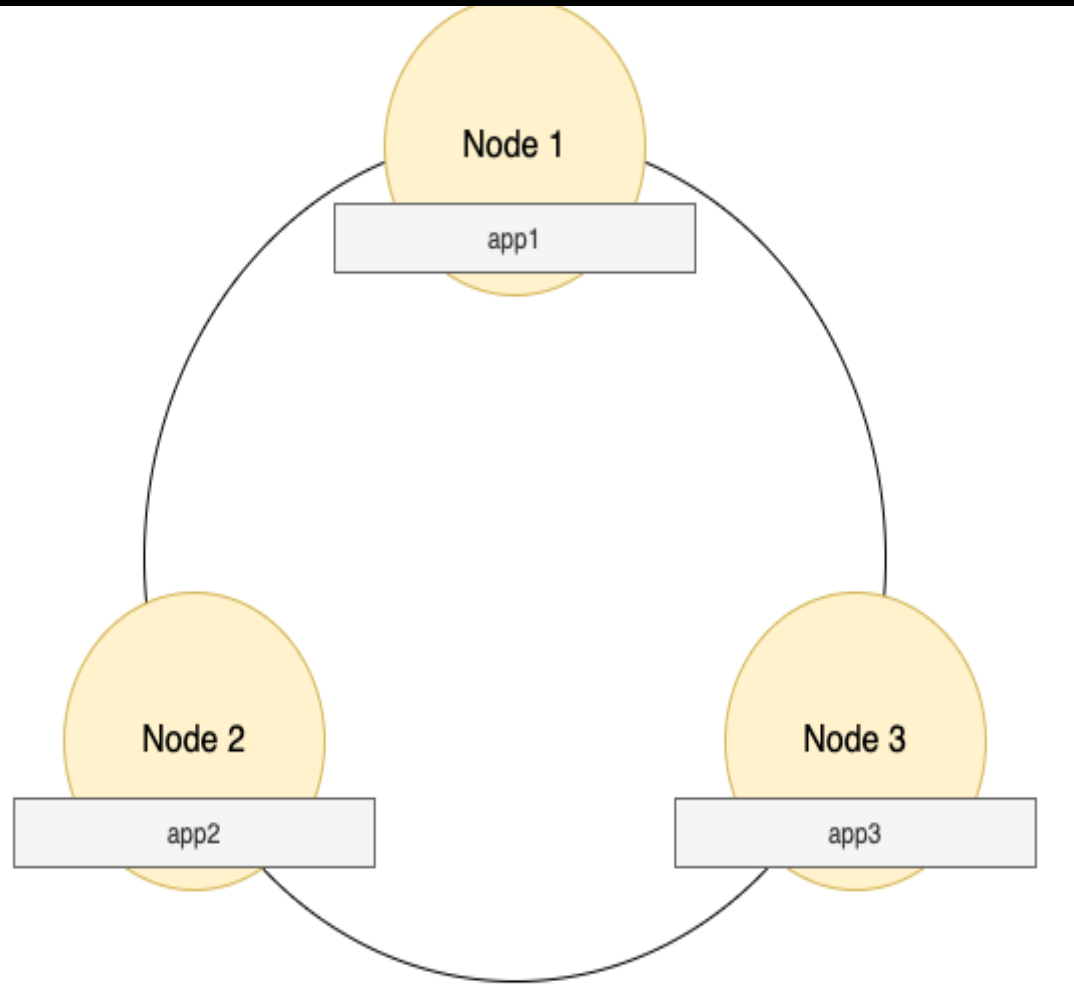
# Partition Key

- The primary goal of a partition key is to distribute the data evenly across a cluster and query the data efficiently.

```
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) );
```



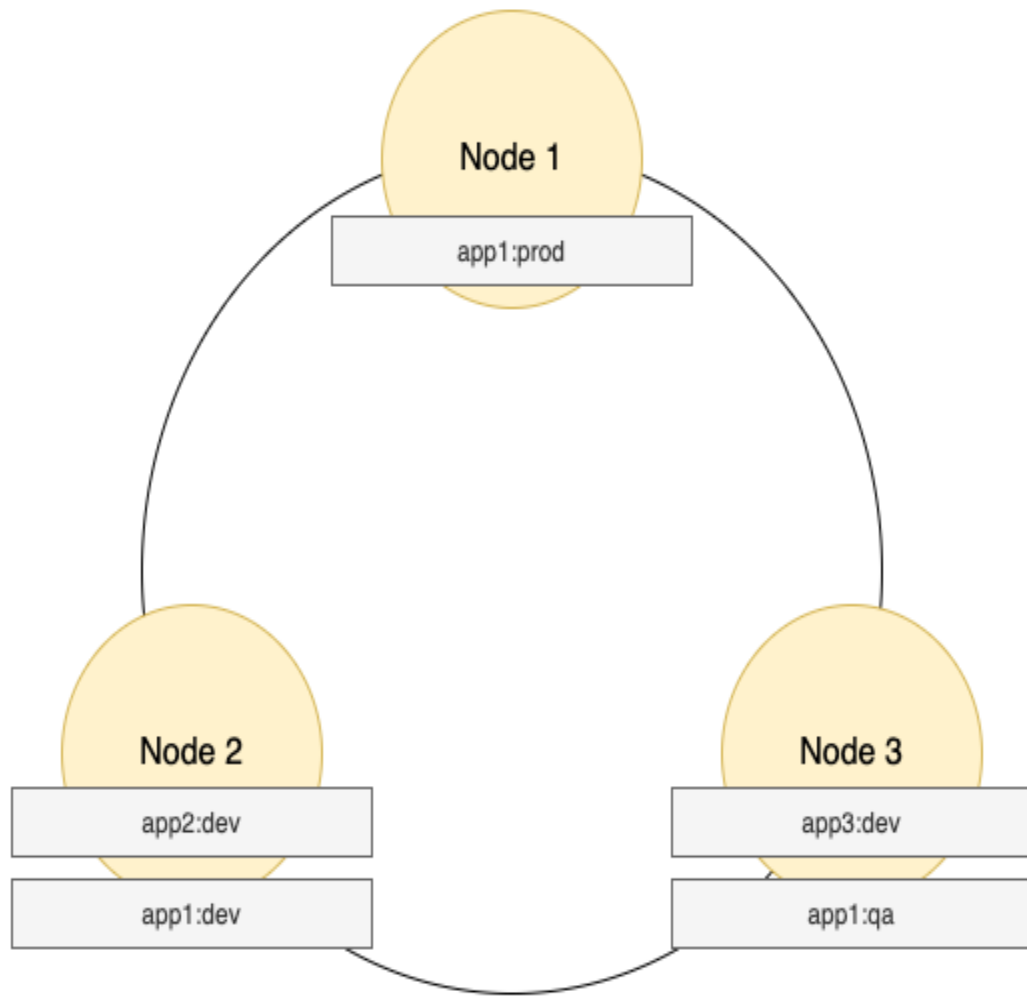
| id | app_name | env  | hostname | log_datetime                 | log_level | log_message       |
|----|----------|------|----------|------------------------------|-----------|-------------------|
| 1  | app1     | prod | host1    | 2021-08-15 04:05:00.000+0000 | INFO      | app1 INFO message |
| 2  | app2     | dev  | host2    | 2021-08-14 12:30:00.000+0000 | INFO      | app2 log message  |
| 3  | app3     | dev  | host2    | 2021-08-13 16:45:50.000+0000 | WARN      | app3 log message  |




# Composite Partition Key


- If we need to combine more than one column value to form a single partition key, we use a composite partition key.

```
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)) );
```






# Create, Read, Update, and Delete (CRUD) Operations

- Cassandra supports basic database operations, which can be categorized into
  - Create, Read, Update, and Delete (CRUD) operations,
  - as well as some specialized operations due to its distributed nature.
- 





# Create (Insert)

- To create data in Cassandra, use the INSERT statement.
  - Data is inserted into a table, and each insert typically requires specifying a primary key that determines the partition key and clustering columns,
  - as Cassandra is optimized for queries based on these keys.
  - You can insert single or multiple rows at once.
- 



# Read (select)

- The SELECT statement is used for reading data from Cassandra tables.
  - You can perform various types of queries, such as selecting specific rows,
  - filtering based on conditions,
  - and retrieving aggregated data.
- 

- 
- **Get by Primary Key:** Cassandra excels at retrieving data using the primary key or a combination of partition key and clustering columns.
  - **Batch Read:** You can perform batch reads to retrieve multiple rows using the BATCH statement, which is useful for optimizing read operations.





**Syntax:**

**BEGIN BATCH**

**<insert-stmt>/ <update-stmt>/ <delete-stmt>**

**APPLY BATCH**





# Cassandra CQL Shell

```
cqlsh:javatpoint> SELECT * FROM student;
```

| student_id | student_fees | student_name |
|------------|--------------|--------------|
| 1          | 5000         | Ajeet        |
| 2          | 3000         | Kanchan      |
| 3          | 2000         | Shivani      |

```
<3 rows>
```

```
cqlsh:javatpoint>
```

- 
- 
- In this example, we will perform the BATCH (Insert, Update and Delete) operations
  - Insert a new row with the following details (4, 4000, Sonoo).
  - Update the student\_fees of student with row id 3 to 8000.
  - Delete student\_fees of the employee with row id 2.

## Cassandra CQL Shell

CH 11 INSERT...>

cqlsh:javatpoint> BEGIN BATCH

... INSERT INTO student(student\_id, student\_fees, student\_name) va  
lues (4, 4000, 'Sonoo');

... UPDATE student SET student\_fees = 8000 WHERE student\_id = 3;

... DELETE student\_fees FROM student WHERE student\_id =2;

... APPLY BATCH;

cqlsh:javatpoint>

# Cassandra CQL Shell

```
cqlsh:javatpoint> SELECT * FROM student;
```

| student_id | student_fees | student_name |
|------------|--------------|--------------|
| 1          | 5000         | Ajeet        |
| 2          | null         | Kanchan      |
| 4          | 4000         | Sonoo        |
| 3          | 8000         | Shivani      |

<4 rows>

```
cqlsh:javatpoint> _
```

# Update:


- **Update:** To modify existing data, Cassandra supports the UPDATE statement. You can update one or more columns in a row.
- **Conditional Updates:** Cassandra allows conditional updates, where you can specify conditions that must be met for the update to occur.
- `UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;`

# Delete:

- **Delete:** The DELETE statement is used to remove data from Cassandra tables. You can delete specific rows or columns.
- DELETE FROM <identifier> WHERE <condition>;
- DELETE emp\_sal FROM emp WHERE emp\_id=3;

- **Secondary Indexes:** You can create secondary indexes to enable non-primary key-based queries, although this should be used sparingly as it can impact performance in a distributed system.
- **CREATE INDEX** <identifier> **ON** <tablename>  
>
- **CREATE INDEX** name **ON** student (student\_name);





```
CREATE TABLE my_table (  
    id UUID PRIMARY KEY,  
    name TEXT,  
    age INT,  
    ...  
    INDEX idx_name (name)  
);
```



## DROP INDEX

- Dropping a secondary index uses the DROP INDEX statement:
- Syntax:

DROP INDEX [ IF EXISTS ] index\_name



# Cassandra TTL (Time to Live) using Automatic Data Expiration

- functionality by which data can be automatically expired.
- During data insertion, you have to specify 'ttl' value in seconds. 'ttl' value is the time to live value for the data.
- After that particular amount of time, data will be automatically removed.
- For example, specify ttl value 100 seconds during insertion. Data will be automatically deleted after 100 seconds.

# Syntax

- Insert into  
KeyspaceName.TableName(ColumnNames)  
values(ColumnValues) using ttl  
TimeInseconds;
- insert into  
University.Student(rollno,name,dept,semest  
er) values(103,'Guru','CS',2) using ttl 100;

```
cqlsh> select * from University.Student where rollno=3;
```

| rollno                  | dept | name | semester |
|-------------------------|------|------|----------|
| -----+-----+-----+----- |      |      |          |

```
(0 rows)
```

```
cqlsh>
```

# ALTER COMMANDS

- Alter Table to Change the Data Type of a Column 1.
- Alter the schema of the table "sample".  
Change the data type of the column "sample\_id" to integer from text.

Eg: ALTER TABLE sample ALTER sample\_id  
TYPE int;



## Adding a Column:

- You can add a column in the table by using the ALTER command.
- Syntax:

**ALTER TABLE table name**

**ADD new column datatype;**

## Dropping a Column

- You can also drop an existing column from a table by using ALTER command.

### Syntax:

**ALTER table name**

**DROP column name;**

Eg:

**ALTER TABLE student**

**DROP (student\_fees, student\_phone);**





## Drop a Database

- DROP keyspace students;


## Drop a Table



1. Drop the column familytable "sample".
  2. DROP columnfamily sample;
- 



# CQL Collections

## 1. List

- When the order of elements matter, one should go for a list collection.
  - For example, when you store the preferences of places to visit by a user, you would like to respect his preferences and retrieve the values in the order in which he has entered rather than in sorted order.
  - A list also allows one to store the same value multiple times.
- 





Eg: CREATE TABLE data(name text PRIMARY  
KEY, email list<text>;  
INSERT INTO data(name, email) VALUES  
('ramu',  
['abc@gmail.com','cba@yahoo.com'])



## 2. SET

- A column of type set consists of unordered unique values.
- However, when the column is queried, it returns the value in sorted order.
- For example, for text values, it sorts in alphabetical order.

- 
- 
- CREATE TABLE data2 (name text PRIMARY KEY, phone set<varint>);
  - INSERT INTO data2(name, phone)VALUES ('rahman', {9848022338,9848022339});

- UPDATE data2 ... SET phone = phone + {9848022330} ... where name = 'rahman';
- SELECT \* FROM data2;

| name   | phone   |
|--------|---|
| rahman | {9848022330, 9848022338, 9848022339} (1 rows) |

### 3. MAP

- Map is a data type that is used to store a key-value pair of elements.
- Example:

```
CREATE TABLE data (name text PRIMARY KEY,  
address map<timestamp, text>);
```

```
INSERT INTO data (name, address) VALUES  
('robin', {'home' : 'hyderabad' , 'office' : 'Delhi'  
} );
```

- **CREATE TABLE** airplanes ( name text **PRIMARY KEY**, manufacturer **ascii**, **year** int, mach float );
- **INSERT INTO** airplanes (name, manufacturer, **year**, mach) **VALUES** ('P38-Lightning', 'Lockheed', 1937, 0.7);
- **COPY** airplanes (name, manufacturer, **year**, mach) **TO** 'temp.csv';



- 
- **COPY** airplanes (name, manufacturer, **year**, mach) **FROM** 'temp.csv';
- 