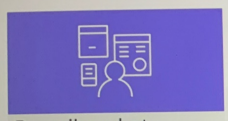
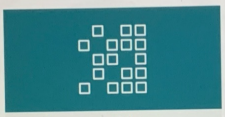
Apache Cassandra Overview

What you will learn



Describe what Apache Cassandra is

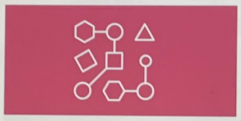


Understand how Apache Cassandra fits in the NoSQL space

Contrast Apache Cassandra with MongoDB

What you will learn

Understand Apache Cassandra's key capabilities



Identify Apache Cassandra's best usage scenarios

Describe Apache Cassandra common use cases

What is Apache Cassandra?

"Apache Cassandra is an ***open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tunable, and consistent database*** that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable.

Created at Facebook, it is now used at some of the most popular sites on the Web."

↳ Netflix
↳ Spotify

Source: Carpenter, Jeff, and Eben Hewitt. *Cassandra: The Definitive Guide*. O'Reilly, 2020.

Apache Cassandra in the NoSQL space

available (immediately)

MongoDB

- Search use cases, ecommerce websites
- Consistency
- Primary-Secondary architecture

Apache Cassandra

- "Always available" type of services (Netflix, Spotify)
- Fast writes – capture all data
- Availability (all time) & scalability
- Peer-to-peer architecture (a simpler)

Key features of Apache Cassandra

single peer-to-peer make it be one of a friendliest NoSQL

- Distributed and decentralized
- Always available with tunable consistency
- Fault-tolerant
- High write throughput (fast)
- Fast and linear scalability
- Multiple data center support
- SQL-like query language

without the need to restart or reconfigure the service →

↳ useful for service that need to be accessed worldwide

What is Apache Cassandra?

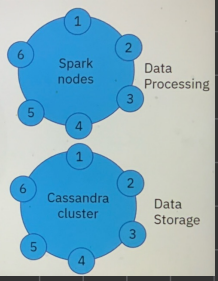
A reliable, performant, scalable database for data storage

Not a drop-in replacement for a relational database

- Does not support joins
- Limited aggregation support
- Limited support of transactions

no concept of referential integrity or foreign keys

For joins and aggregations: Cassandra and Spark



Usage scenarios for Cassandra

- When writes exceed read requests
  - For example, storing all the clicks on your website or all the access attempts on your service
- When using append-like type of data
  - Not many updates and deletes
- When you can predefine your queries and your data access is by a known primary key
  - Data can be partitioned via a key that allows the database to be spread evenly across multiple nodes
- When there is no need for joins or aggregations

Common use cases for Cassandra

Online services

- Users' authentication for access to services
- Tracking users' activity in the application

eCommerce websites

- Storing transactions
- Website interactions (clicks) for prediction of customer behavior
- Status of orders/users' transactions
- Users' profiles and shopping history

Time series

- Monitoring servers' access logs
- Weather updates from sensors
- Tracking packages

Key Features of Apache Cassandra

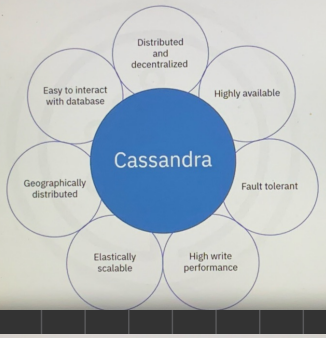
What you will learn

- Explain distributed and decentralized
- Describe how replication works in Cassandra
- List the difference between availability and consistency

What you will learn

- Describe the scalability of Cassandra
- Explain how Cassandra provides high write throughput
- Explain what CQL language is

Key features



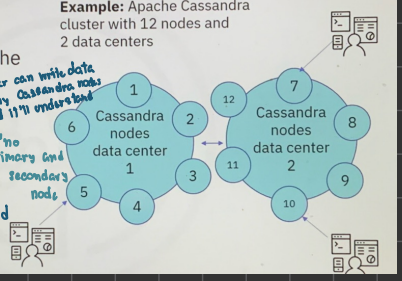
Distributed and decentralized

not many NoSQL distributed databases that are both decentralized and centralized

- Cluster runs on multiple distributed machines
- Users address the cluster in the seamless to the number of nodes in the cluster
- All nodes perform the same functions (server symmetry)
- Peer-to-peer communication

Cassandra architecture is built such way on combination of client and server provide sufficient info to route the user request to end user

Example: Apache Cassandra cluster with 12 nodes and 2 data centers



user can write data to any Cassandra node and it'll understand

no primary and secondary nodes

keep all node sync and it's called gossip

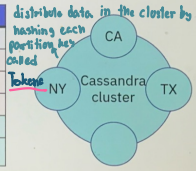
Data distribution starts with a query

Table (Partition key= State)

Initial data → **Partition key** → **Data distribution**

UserID	Name	State
1	John	TX
2	Elaine	CA
3	Alex	NY
4	Jay	CA
5	Julio	NY
6	Elen	CA

State	Name	UserID
TX	John	1
CA	Elaine	2
CA	Elen	6
CA	Jay	4
NY	Alex	3
NY	Julio	5



distribute data in the cluster by hashing each partition key called Tokens

Query: All users in a state =>
PartitionKey = State

grouped state column
based on partition key

Data replication and multiple DC support

- Replicas
  - How many nodes contain a certain piece of your data (partition)
- Data replication takes cluster topology into consideration
  - Racks and data centers distribution of nodes

Ex.
- 8-node cluster, 2 DCs, Rep = 5 (DC1 RF3, DC2 RF2)
- Nodes 1 and 2, 3 and 4, 5 and 6, 7 and 8 in the same rack

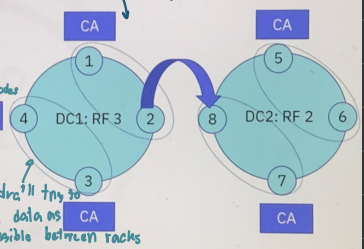
Data replication is done clockwise in the cluster

is done according to the set replication factor (RF)

specify number of nodes that will hold the replicas for each partition

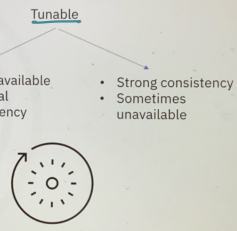
two data center

cassandra'll try to distribute data as much as possible between racks



Availability versus consistency

- Always available
- Tunable consistency
  - Per operation set consistency (read/write)
- CAP theorem: Cassandra favors availability over consistency
  - Tunable: Strong or eventual consistency
  - Consistency conflicts solved during reads

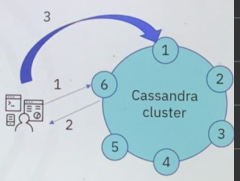


distributed system can not according to CAP theorem be consistent and available at the same time

if it lost, there will still be nodes available to answer, though return data might be in consist

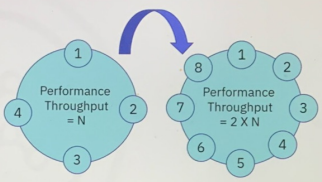
High availability and fault tolerance

- Peer-to-Peer architecture
- The other nodes in the cluster immediately recognize nodes' temporary/permanent failures
- Nodes reconfigure the data distribution once nodes are taken out of the cluster
- Failed requests can be retransmitted to other nodes



Fast and linear scalability

- Scales horizontally by adding new nodes in the cluster
- Performance increases linearly with the number of added nodes
- New nodes are automatically assigned tokens from existing nodes
- Adding and removing nodes is done seamlessly

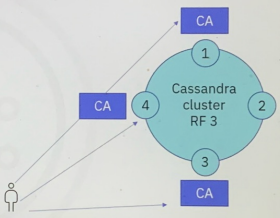


Double performance by doubling the number of nodes

High write throughput

can handle large number of write

At the cluster level:
- Writes can be distributed in parallel to all nodes holding replicas

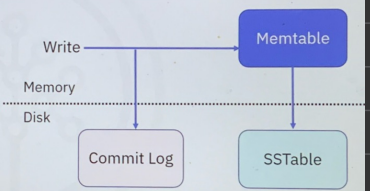


Write operations: cluster level

High write throughput

- No reading before writing (by default)
- At node level
  - Writes are done in node memory and later flushed on disk
  - All disk writes are sequential, append-like operations

no read



Write operations: node level

Cassandra query language (CQL)

Data Definition and Manipulation: CQL, an SQL-like syntax

similar to

```
CREATE TABLE test (
    groupid uuid,
    name text,
    occupation text,
    age int,
    PRIMARY KEY ((groupid), name)
);
INSERT INTO test (groupid, name, occupation, age)
        VALUES  (1001, 'Thomas', 'engineer', 24),
                (1001, 'James', 'designer', 30),
                (1002, 'Lily', 'writer', 35);
SELECT * FROM test WHERE groupid = 1001;
```

Apache Cassandra Data Model Part 1

What you will learn

- Describe the logical entities of the Cassandra data model
- Describe the role of primary keys
- Explain what partition keys are
- Describe the two types of tables supported in Cassandra

Logical entities: Tables and keyspaces

store in

Table

- Is the logical entity that organizes data storage at cluster and node level according to a declared schema

grouped in

Keyspace

- Is the logical entity that contains one or more tables
- Replication and data centers' distribution are defined at the keyspace level
- Recommended 1 keyspace per application

Logical entities: Tables

- Data is organized in tables containing rows of columns
- Tables can be created, dropped, and altered at runtime without blocking updates and queries
- To create a table, you must define a primary key and other data columns (regular columns)

```
CREATE TABLE intro_cassandra.groups(
groupid int,
group_name text STATIC,
username text,
age int,
PRIMARY KEY ((groupid), username));
```

Partition Key

Clustering Key

Primary key in Cassandra tables

- Is a subset of the declared columns
- Is mandatory: You cannot change the key once defined
- Peforms two main roles:
  - Optimizes the read performance for your queries on the table: Query-driven table design
  - Provide uniqueness to the entries
- Has two components:
  - Partition key is mandatory
  - Clustering keys are optional

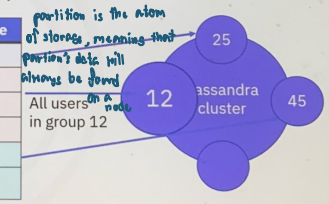
Partition keys

- When inserting data into a table, data is grouped into partitions and distributed on the cluster nodes based on the partition key
- Partition key => Hash (token) => Node
- The Partition key determines data (partition) locality in the cluster

GroupID	Group_name	Username	Age
25	Vegan cooking	Johns@gmail.com	60
12	Baking	Alaind@gmail.com	32
12	Baking	Elaine@yahoo.com	60
12	Baking	Peterd@gmail.com	32
45	Grilling	Moirad@yahoo.com	35
45	Grilling	Daveg@gmail.com	43

partition is the atom of storage, meaning that partition's data will always be found on a node.

All users in group 12



PRIMARY KEY ((groupid), username))

Table types

Two types of tables: Static and dynamic

- Static table — reason: contains only partition key, single or multiple columns, and no clustering key
  - PRIMARY KEY (username)
- Dynamic table — composed of both partition key or keys and clustering key or keys
  - PRIMARY KEY ((groupid), username))

Static tables

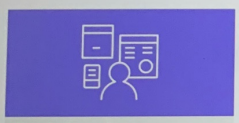
Users: Static table
Partition key = username
No Clustering key
1 partition = 1 entry

Username	Occupation	Age
Johns@gmail.com	Engineer	60
Alaind@gmail.com	Programmer	32
Elaine@yahoo.com	Engineer	60
Peterd@gmail.com	Marketer	27
Moirad@yahoo.com	None	35
Daveg@gmail.com	Pianist	43
JayZ@yahoo.com	Entertainer	46

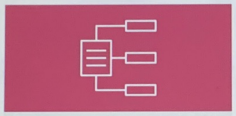
```
CREATE TABLE
intro_cassandra.users(
username text,
occupation text,
age int,
PRIMARY KEY (username));
```

Apache Cassandra Data Model Part 2

What you will learn



Explain what clustering keys are



Describe dynamic tables

Explain the basic guidelines for modeling your data

clustering column specifies the order that the data arranged in inside the partition

Clustering keys

- Store data in ascending or descending order within the partition for the fast retrieval of similar values
- Can have single or multiple columns
- Completes the primary key in dynamic tables
  - Gives uniqueness to each entry
  - Improves read query performance

```
CREATE TABLE intro_cassandra.groups(
groupid int,
group_name text STATIC,
username text,
age int,
PRIMARY KEY ((groupid), username));
```

Dynamic table

important for data locality

Partition key

means the data inside group partition is going to be stored by username in ASCS

Clustering key

Clustering keys

- Query: "return all users in groupid 12 with age 32"
- Clustering key = age, username

```
CREATE TABLE intro_cassandra.groups_by_age(
groupid int,
group_name text STATIC,
username text,
age int,
PRIMARY KEY ((groupid), age, username));
```

inside age, group by

data inside each partition is first grouped and ordered by age

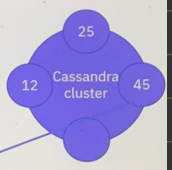
same age will be stored together

GroupID	Group_name	Age	Username
25	Vegan cooking	60	Johns@gmail.com
12	Baking	32	Alaind@gmail.com
12	Baking	32	Peterd@gmail.com
12	Baking	60	Elaine@yahoo.com
45	Grilling	35	Moirad@yahoo.com
45	Grilling	43	Daveg@gmail.com

Dynamic tables

```
INSERT INTO intro_cassandra.groups (groupid, group_name, username, age)
VALUES (45, 'Grilling', 'JayZ@yahoo.com', 46));
```

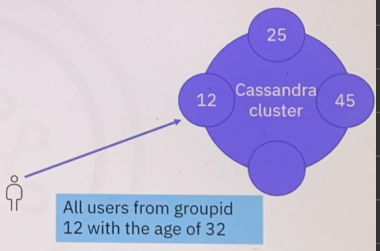
GroupID	Group_name	Username	Age
25	Vegan cooking	Johns@gmail.com	60
12	Baking	Alaind@gmail.com	32
12	Baking	Elaine@yahoo.com	60
12	Baking	Peterd@gmail.com	32
45	Grilling	Moirad@yahoo.com	35
45	Grilling	Daveg@gmail.com	43
45	Grilling	JayZ@yahoo.com	46



Basic rules of data modeling

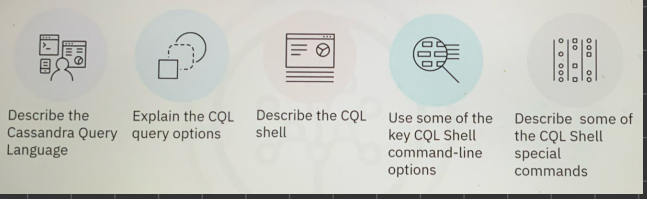
- Data modeling: Build a primary key that optimizes query execution time
- Choose a partition key: Starts answering your query and spreads the data uniformly in the cluster
- Minimize the number of partitions read to answer the query

Note: To further optimize your query model, order clustering keys according to the query



Introduction to Cassandra Query Language Shell (cqlsh)

What you will learn



- Describe the Cassandra Query Language
- Explain the CQL query options
- Describe the CQL shell
- Use some of the key CQL Shell command-line options
- Describe some of the CQL Shell special commands

Cassandra Query Language

- CQL is the primary language for communication with Cassandra clusters
- Simple yet intuitive syntax (SQL-like)
- CQL lacks grammar for relational features such as JOIN statements
- Different behavior of CQL commands vs. SQL

not support

- CREATE KEYSPACE intro_cassandra WITH ..
- CREATE TABLE test () ..
- INSERT INTO test () VALUES () ..
- SELECT * FROM test WHERE ..
- UPDATE test SET age = 25 WHERE userid=30 ..
- DELETE FROM test WHERE userid=30 ..
- DROP TABLE test;
- TRUNCATE TABLE test;

Cassandra Query Language

- CQL keywords are case-insensitive
- Identifiers in CQL are case-insensitive unless enclosed in double quotation marks
- Names for identifiers created using uppercase are stored in lowercase
- Commented text (//) is ignored by CQL

```
Select * from users;
SELECT * from users;
SELECT * FROM USERS;
//all commands are similar

CREATE TABLE USERS(..);
CREATE table users(..);
//Stored name of the table: users
```

Running CQL queries

- Run using Cassandra client drivers
  - Java, Python, Ruby, Nodejs, PHP, Scala, Clojure ...
  - Default = open source Datastax Java Driver
- Run using the cqlsh client
  - Python-based command line shell for interacting with Cassandra through CQL
  - Shipped with every Cassandra package
  - Connects to a single node (default node or one specified on the command line)
- Other CQL client editors are available

CQL shell (cqlsh)

Using cqlsh, you can:

- Create, alter, drop keyspaces
- Create, alter, drop tables
- Insert, update, delete data

```
[cqlsh> use intro_cassandra:;
[cqlsh:intro_cassandra> select * from users;
```

username	age	occupation
Daveg@gmail.com	43	engineer
JayZ@yahoo.com	46	entertainment
Elaine@yahoo.com	60	producer
Johns@gmail.com	60	actor
Alaind@gmail.com	32	actor
Peterd@gmail.com	32	producer
Moirad@yahoo.com	35	dj

(7 rows)

Command-line options for cqlsh

> cqlsh [options] [host [port]]

Options:
--help shows help about the cqlsh command options
--version shows the version of cqlsh being used
-u -user specifies the username to authenticate to Cassandra with
-p -password specifies the password to authenticate to Cassandra with
-k -keyspace specifies a keyspace to authenticate to
-f -file enables execution of commands from a given file
--request-timeout specifies the request timeout in seconds (defaults to 10s)

An example of cqlsh

Commands used in cqlsh:

USE intro_cassandra

SELECT * FROM groups where groupid=12;

INSERT INTO groups(groupid,username,group_name,age) VALUES (12,'aland@gmail.com','baking',32);

SELECT * FROM groups;

An example of cqlsh

```
[cqlsh> use intro_cassandra:;
[cqlsh:intro_cassandra> select * from groups where groupid=12;
```

groupid	username	group_name	age
12	Elaine@yahoo.com	Baking	46
12	Peterd@gmail.com	Baking	32

(2 rows)

```
cqlsh:intro_cassandra> insert into groups (groupid, username, group_name, age) values (12, 'Aland@gmail.com', 'Baking',32) ; cqlsh:intro_cassandra> select * from groups where groupid=12;
```

groupid	username	group_name	age
12	Aland@gmail.com	Baking	32
12	Elaine@yahoo.com	Baking	46
12	Peterd@gmail.com	Baking	32

(3 rows)

cqlsh special commands

CAPTURE - Captures the output of a command and adds it to a file

CONSISTENCY - Shows the current consistency level and sets a new one

COPY - Copies data to and from Cassandra

DESCRIBE - Describes the current cluster of Cassandra and its objects

EXIT - Terminates the cqlsh session

PAGING - Enables or disables paging of the query results

TRACING - Enables or disables request tracing

cqlsh CONSISTENCY

- Cassandra is tunable consistent
- Sets the **consistency level** for the operations to follow
- **Consistency** refers to the number of nodes (out of the total replicas) that should respond to a query (write/read) to consider the query successful

```
cqlsh: intro_cassandra_keyspace> CONSISTENCY

ONE
TWO
THREE
QUORUM - majority of nodes from the entire cluster
ALL
LOCAL_QUORUM - majority of nodes from the local data center
(according to the specified data center set replication for the keyspace)
```

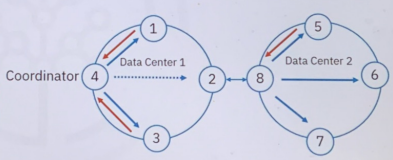
Consistency example - QUORUM

```
cqlsh: intro_cassandra_keyspace> CONSISTENCY QUORUM
cqlsh: intro_cassandra_keyspace> INSERT INTO groups (groupid,
group_name,username,age) VALUES (45, 'Grilling', 'JayZ@yahoo.com', 46);
```

Multi DC cluster of 8 nodes

DC1 RF 2, DC2 RF 3

- Set consistency QUORUM (3)
- Write operations go to all replicas (5)
- Minimum of 3 nodes (from both DCs) need to answer for a successful operation



cqlsh COPY (import/export data)

```
cqlsh: intro_cassandra_keyspace> COPY
```

CQL shell commands that import and export CSV (comma-separated values or delimited text files). Not suitable for bulk loading.

- COPY TO exports data from a table into a CSV file. Each row is written to a line in the target file with fields separated by the delimiter.
- COPY FROM imports data from a CSV file into an existing table.
  - Each line in the source file is imported as a row.
  - All rows in the data set must contain the same number of fields and have values in the PRIMARY KEY fields.
  - The process verifies the PRIMARY KEY and imports data accordingly.

Term	Definition
BSON	Binary JSON, or BSON, is a binary-encoded serialization format used for its efficient data storage and retrieval. BSON is similar to JSON but designed for compactness and speed.
Aggregation	Aggregation is the process of summarizing and computing data values.
Availability	In the context of CAP, availability means that the distributed system remains operational and responsive, even in the presence of failures or network partitions. Availablity is a fundamental aspect of distributed systems.
CAP	CAP is a theorem that highlights the trade-offs in distributed systems, including NoSQL databases. CAP theorem states that in the event of a network partition (P), a distributed system can choose to prioritize either consistency (C) or availability (A). Achieving both consistency and availability simultaneously during network partitions is challenging.
Cluster	A group of interconnected servers or nodes that work together to store and manage data in a NoSQL database, providing high availability and fault tolerance.
Clustering key	A clustering key is a primary key component that determines the order of data within a partition.
Consistency	In the context of CAP, consistency refers to the guarantee that all nodes in a distributed system have the same data at the same time.
CQL	Cassandra Query Language, known as CQL, is a SQL-like language used for querying and managing data in Cassandra.
CQL shell	The CQL shell is a command-line interface for interacting with Cassandra databases using the CQL language.
Decentralized	Decentralized means there is no single point of control or failure. Data is distributed across multiple nodes or servers in a decentralized manner.
Dynamic table	A dynamic table allows flexibility in the columns that the database can hold.
Joins	Combining data from two or more database tables based on a related column between them.
Keyspace	A keyspace in Cassandra is the highest-level organizational unit for data, similar to a database in traditional relational databases.
Partition Key	The partition key is a component of the primary key and determines how data is distributed across nodes in a cluster.
Partitions	Partitions in Cassandra are the fundamental unit of data storage. Data is distributed across nodes and organized into partitions based on the partition key.
Peer-to-peer	The term peer-to-peer refers to the overall Cassandra architecture. In Cassandra, each node in the cluster has equal status and communicates directly with other nodes without relying on a central coordinator. If a primary node fails, another node automatically becomes the primary node.
Primary key	The primary key consists of one or more columns that uniquely identify rows in a table. The primary key includes a partition key and, optionally, clustering columns.
Replication	Replication involves creating and maintaining copies of data on multiple nodes to ensure data availability, reduce data loss, fault tolerance (improve system resilience), and provide read scalability.
Scalability	Scalability is the ability to add more nodes to the cluster to handle increased data and traffic.
Static table	A static table has a fixed set of columns for each row.
Table	A table is a collection of related data organized into rows and columns.
Transactions	Transactions are sequences of database operations (such as reading and writing data) that are treated as a single, indivisible unit.