BITS Pilani
Pilani | Dubai | Goa | Hyderabad

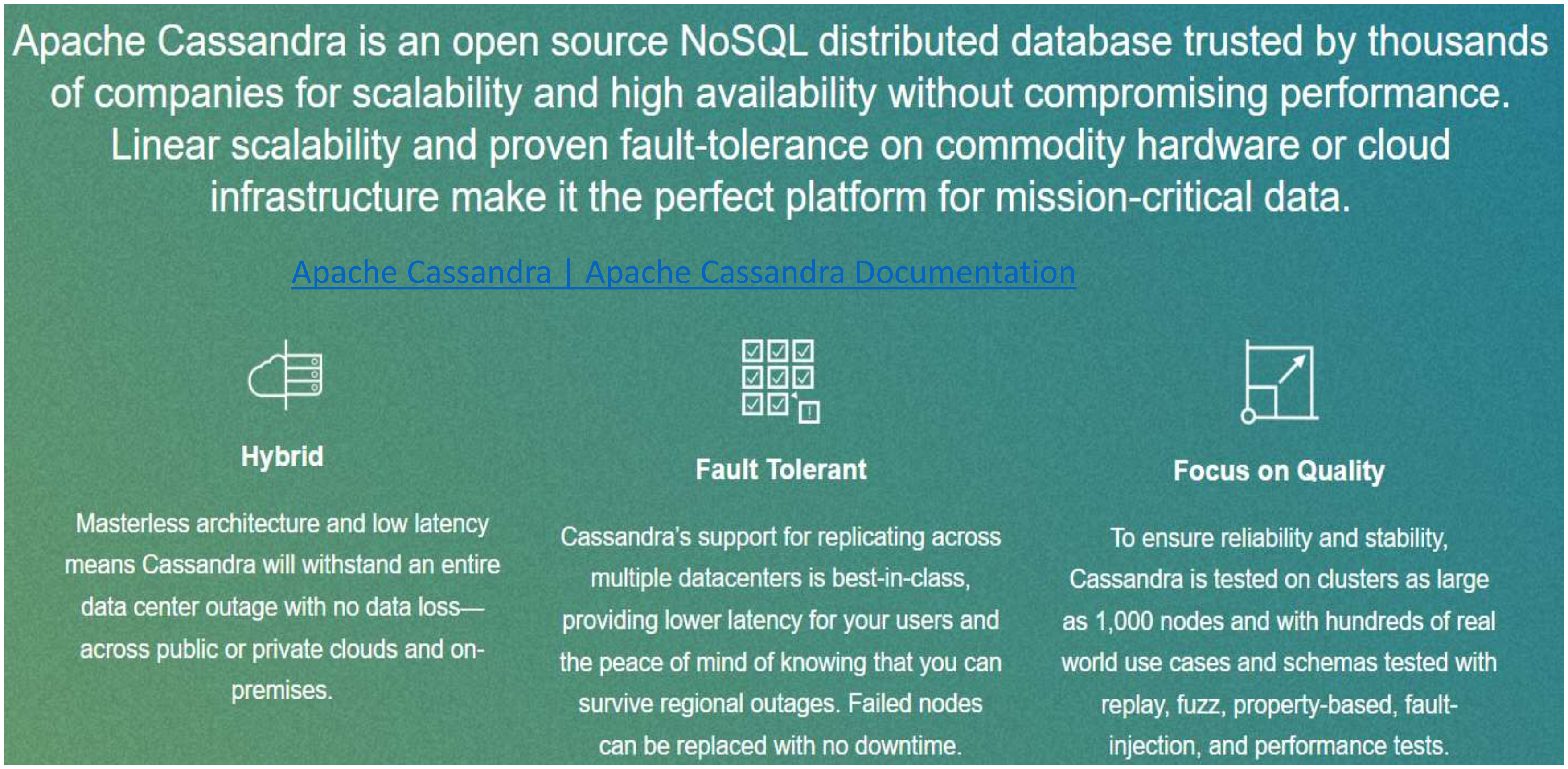
DSECL ZG 522: Big Data Systems

Session 5-2: NoSQL Database - Cassandra

Janardhanan PS

janardhanan.ps@wilp.bits-pilani.ac.in

What is Apache Cassandra



Apache Cassandra is an open source NoSQL distributed database trusted by thousands of companies for scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data.

[Apache Cassandra | Apache Cassandra Documentation]

Hybrid

Masterless architecture and low latency means Cassandra will withstand an entire data center outage with no data loss—across public or private clouds and on-premises.

Fault Tolerant

Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages. Failed nodes can be replaced with no downtime.

Focus on Quality

To ensure reliability and stability, Cassandra is tested on clusters as large as 1,000 nodes and with hundreds of real world use cases and schemas tested with replay, fuzz, property-based, fault-injection, and performance tests.

Cassandra

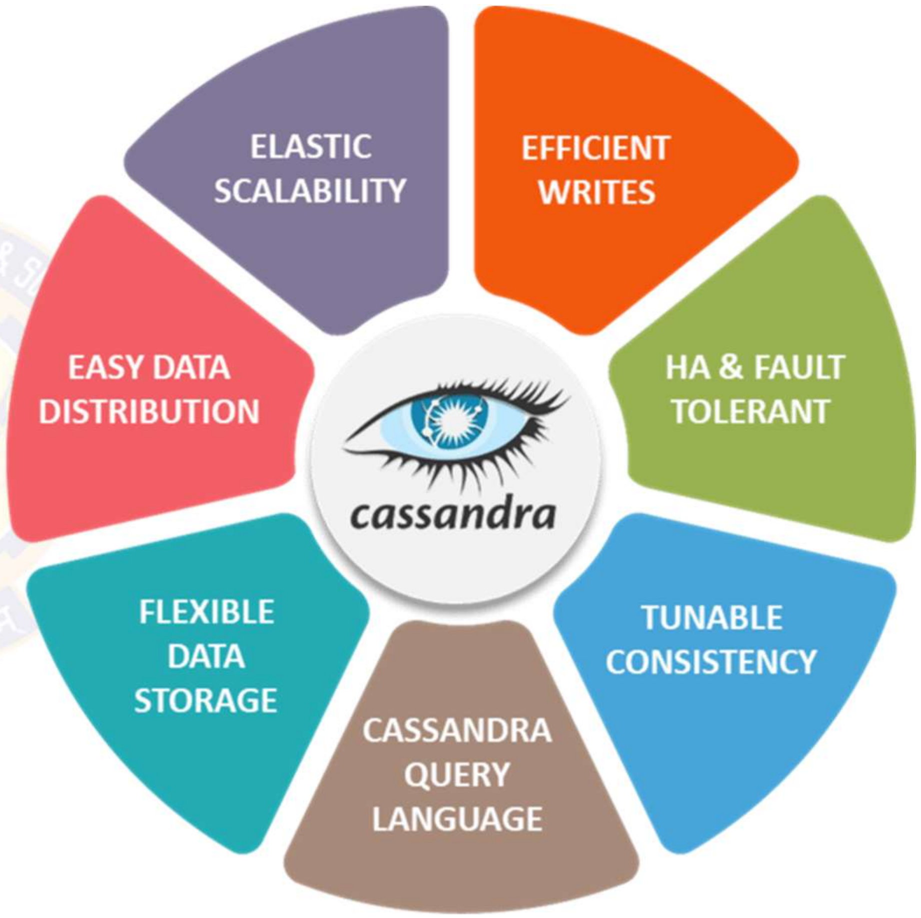
- Built on Amazon Dynamo and Google Big Table concepts
- Based AP in CAP (Consistency is sacrificed) - BASE
- Ideal for High performance, high availability applications that can sacrifice consistency
  - Built for peer-to-peer symmetric nodes (instead of primary-secondary architecture as in MongoDB)
  - Performs well for write intensive applications
- Column oriented DB
  - Create keyspace (like a DB)
  - Within keyspace create column family (like a table)
  - Within CF create attributes / columns with their types

Cassandra

- Initially developed by Facebook
  - Open-sourced in 2008
- Used by 1500+ businesses, e.g., Comcast, eBay, GitHub, Hulu, Instagram, Netflix, Best Buy, …
- Column-family store
  - Supports key-value interface
  - Provides an SQL-like CRUD interface: CQL
  - Gossip protocol (constant communication) to maintain cluster health
  - Ring-based replication model

Cassandra Features

- Massively scalable
- Master-less architecture
- Linear scale performance
- No single point of failure
- Automated Replication
- Peer-to-peer
- Written in Java
- Tunable consistency
- Clients - CQL and/or Thrift



Case study - eBay

- Marketplace has 100 million active buyers with 200+ million items
- 2B page views, 80B DB calls, multi-PB storage capacity
- No transactions, joins, referential integrity
- Multi-DC deployment
- 400M+ writes and 200M+ reads
- 3 Use cases
  - Social signal on product pages (read latency is not important but write performance is key)
  - Connecting users and items via buy, sell, bid, watch events
  - Many time series analysis cases, e.g. fraud detection

https://www.slideshare.net/jaykumarpatel/cassandra-at-ebay-13920376/2-eBay_Marketplaces_97_million_active

Case study - AdStage (from AWS use cases)

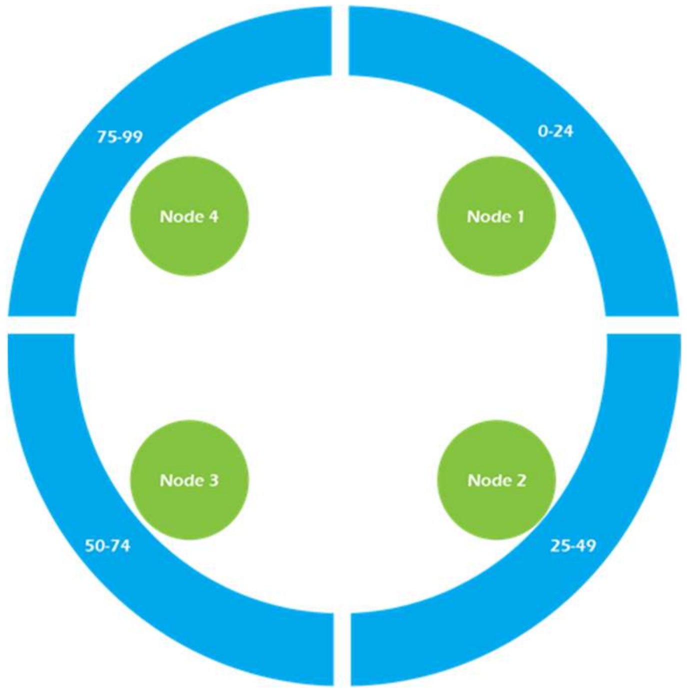
- Sector AdTech
- Online advertising platform to manage multi-channel ad campaigns on Google, FB, Twitter, Bing, LinkedIn
- 3 clusters with 80+ nodes on AWS
- Vast amount of real-time data from 5 channels
- Constantly monitor trends and optimize campaigns for advertisers
- High performance and availability - consistency is not critical as it is read mainly
- Cassandra cluster can scale as more clients are added with no SPOF

Cassandra Data Model: Partitions

- Partitions are groupings of data that will be co-located on storage
- The partition key is responsible for distributing data among nodes.
- A partition key is the same as the primary key when the primary key consists of a single column.
- Partition keys belong to a node. Cassandra is organized into a cluster of nodes, with each node having an equal part of the partition key hashes.

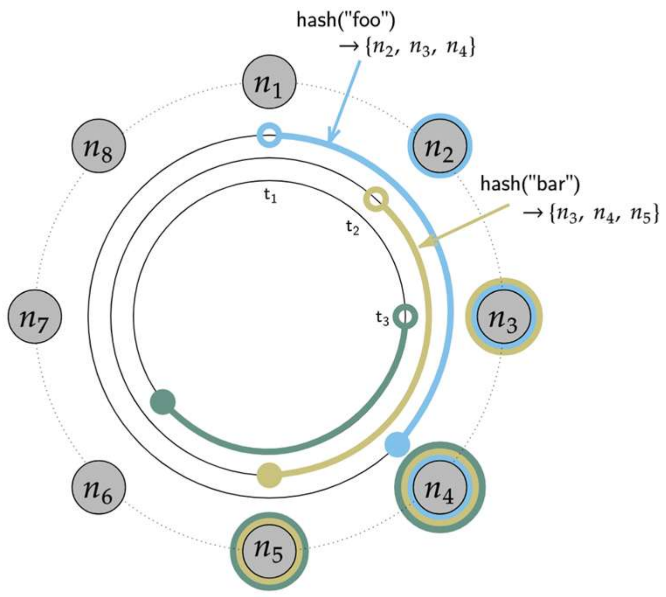
Cassandra Data Model: Partition Keys

- Nodes in a Cassandra cluster owns a range of Keys (Tokens)
- Partition key is responsible for distributing data among nodes
- Partitions are groupings of data that will be co-located on storage of the node owning the key
- Partition keys are assigned to the nodes of the cluster
- Cassandra is organized into a cluster of nodes, with each node having an equal part of the partition key hashes.
- Imagine we have a four node Cassandra cluster.
- In the example cluster (ring):
- Node 1 is responsible for partition key hash values 0-24
- Node 2 is responsible for partition key hash values 25-49
- Node 3 is responsible for partition key hash values 50-74
- Node 4 is responsible for partition key hash values 75-99



Partitioners

- Partitions data based on hashing to distribute data blocks from a column among nodes*
- Random
  - Crypto hash (MD5)- more expensive
- Murmur3
  - Non-crypto consistent hash (MU-Multiple / R - Rotate operations but easier to reverse compared to Crypto hash)
  - 3-5x faster and overall 10% performance improvement
- Byteorder
  - Lexical order



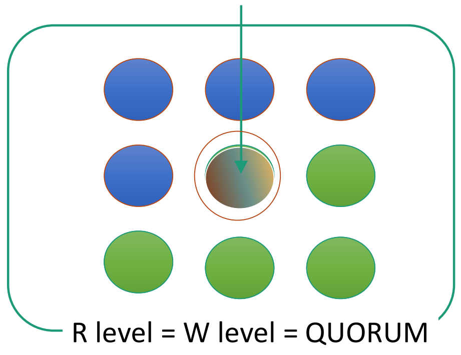
Cassandra consistency semantics

- Key-value store with columnar storage
- No primary replica - high partition tolerance and availability and levels of consistency
- Support for light transactions with “linearizable consistency”
- A Read or Write operation can pick a consistency level
    - ONE, TWO, THREE, ALL - 1,2,3 or all replicas respectively have to ack
    - QUORUM - majority have to ack
    - LOCAL_QUORUM - majority within same datacenter have to ack
    - …
- For “causal consistency” pick Read consistency level = Write consistency level = QUORUM
- Why ? At least one node will be common between write and read set so a read will get the last write of a data item
- What happens if read and write use LOCAL_QUORUM ?
- If no overlap between read and write sets then “Eventual consistency”

https://cassandra.apache.org/doc/latest/architecture/dynamo.html

https://cassandra.apache.org/doc/latest/architecture/guarantees.html#

Read latest written value from common node



Cassandra Write Consistency levels

- *ALL*. Data is written on all replica nodes in the cluster before the coordinator node acknowledges the client. (Strong Consistency, high latency)
- *QUORUM*. Data is written on a given number of replica nodes in the cluster before the coordinator node acknowledges the client. This number is called the quorum. (Eventual Consistency, low latency)
- *LOCAL_QUORUM*. Data is written on a quorum of replica nodes in the same data center as the coordinator node before this last one acknowledges the client. (Eventual Consistency, low latency)
- *ONE*, *TWO*, *THREE*. Data is written in to specified no of replica node. (Eventual Consistency, low latency)
- ANY – Data is written to any replica node
- *LOCAL_ONE*. Data is written in at least one replica node in the same data center as the coordinator node. (Eventual Consistency, low latency)

Cassandra Read Consistency levels

- *ALL*. The coordinator node returns the requested data to the client only after all replicas have responded. (Strong consistency, less availability)
- *QUORUM*. The coordinator node returns the requested data to the client only after a quorum of replicas has responded. (Eventual consistency, high-availability)
- *LOCAL QUORUM*. The coordinator node returns the requested data to the client only after a quorum of replicas has responded from the same datacenter as the coordinator. (Eventual consistency, high-availability)
- *ONE, TWO, THREE*. The coordinator node returns the requested data to the client from the closest replica nodes. (Eventual consistency, high availability)
- *LOCAL ONE*. The coordinator node returns the requested data to the client from the closest replica node in the local datacenter. (Eventual consistency, high availability)

Tunable Consistency

Cassandra guarantees strong consistency if

(nodes_**W**ritten + nodes_**R**ead) > replication_factor **N**

$$\boxed{\mathbf{R + W > N}}$$

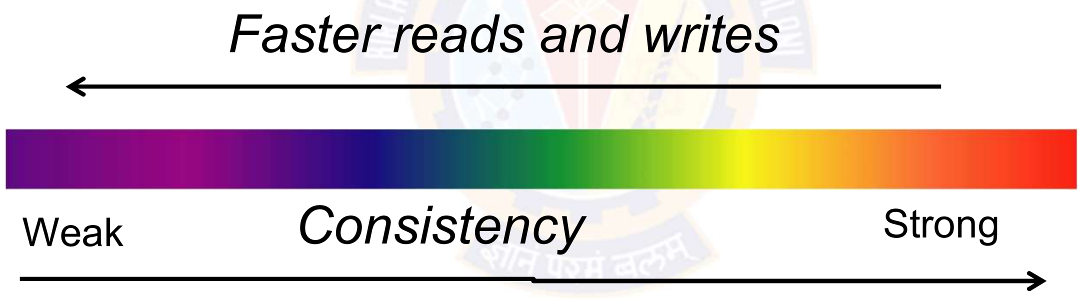
Tuning done by controlling the number of nodes (replicas)

- Selected for Write
- Selected for reads
- Replica sets for Writes and Reads should overlap for strong consistency
- What happens if the consistency level for Write and Read is Quorum ?

This simple form (https://www.ecyrd.com/cassandracalculator/) allows you to try out different parameters for your Apache Cassandra cluster and see the impact of them for your application.

Cassandra Consistency Spectrum

- Cassandra has C A P
- But Consistency is tunable
- Give up a little A and P to get more C

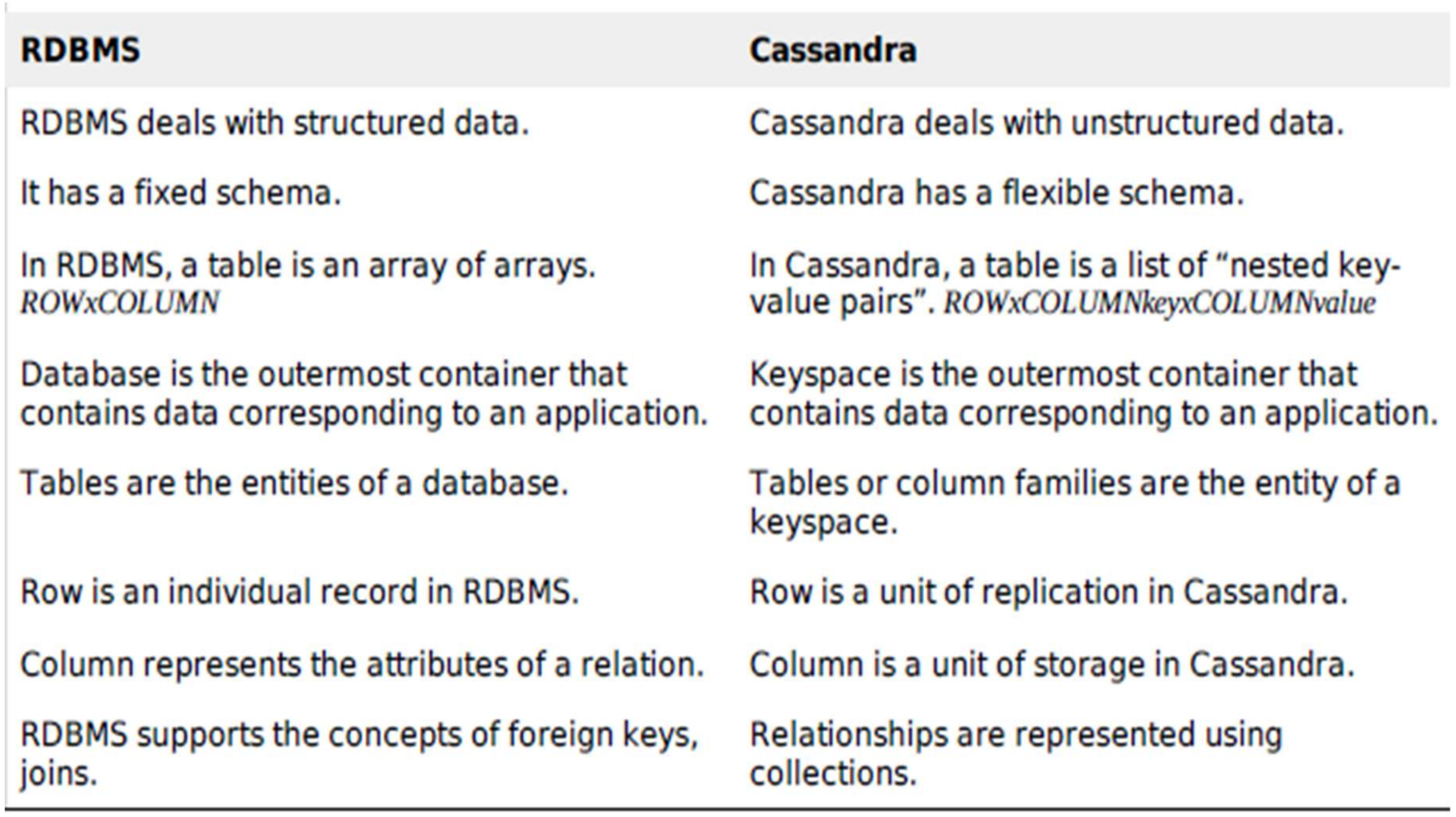


- The lower the level of consistency, you get faster reads and writes
- The higher the consistency level, less chance of getting stale data during read
- Pay for this with latency
- Consistency selection depends on your situational needs

How applications deal with eventual consistency of AP

- Application must be ready to deal with multiple versions of data
- Application must handle stale data
- Expect NULL values instead of data
- Application must be able to fix inconsistent state
- Read again, if you do not get the expected data during the first read

Cassandra Data model: Comparison with RDBMS



RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. *ROWxCOLUMN*	In Cassandra, a table is a list of "nested key-value pairs". *ROWxCOLUMNkeyxCOLUMNvalue*
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Cassandra Vs MongoDB

	Cassandra	MongoDB
Data model	Cassandra uses a wide-column data model more closely related to relational databases.	MongoDB moves completely away from the relational model by storing data as documents.
Basic storage unit	Sorted string tables (SSTables).	Serialized JSON documents.
Indexing	Cassandra supports Primary and Secondary indexes	MongoDB indexes at a collection level and field level and offers multiple indexing options.
Query language	Cassandra uses CQL.	MongoDB uses MQL.
Concurrency	Cassandra achieves concurrency with row-level atomicity and tunable consistency.	MongoDB uses document-level locking to ensure concurrency.
Availability	All Cassandra nodes are equally capable and can function in the role of a coordinator. Data replication by keys offer high availability.	MongoDB uses a single primary node and multiple replica nodes. Combined with sharding, MongoDB provides high availability and scalability.
Partitioning	Consistent hashing algorithm, less control to users.	Users define sharding keys and have more control over partitioning.

Cassandra Installation

Download (Datastax or Open source apache version)

- Stable Apache Cassandra from https://cassandra.apache.org/_/download.html
- You can get apache-cassandra-4.0.9-bin.tar.gz (Latest GA version)

- Untar the binary .gz file
  - tar -zxvf apache-cassandra-4.0.9-bin.tar.gz
- Cassandra Configuration file - /conf/cassandra.yaml
  - You may change cluster name and leave all other values to default for a single node Cassandra
- Set JAVA_HOME and PATH in ENV
  - export JAVA_HOME=/usr/local/java/jdk1.8.0_111
  - export PATH=$JAVA_HOME/bin:$PATH
- Set CASSANDRA home and PATH in .bash_profile
  - export CASSANDRA_HOME=/home/user/apache-cassandra-4.0.9
  - export PATH=$CASSANDRA_HOME/bin:$PATH
- Start cassandra
  - ./bin/cassandra -f

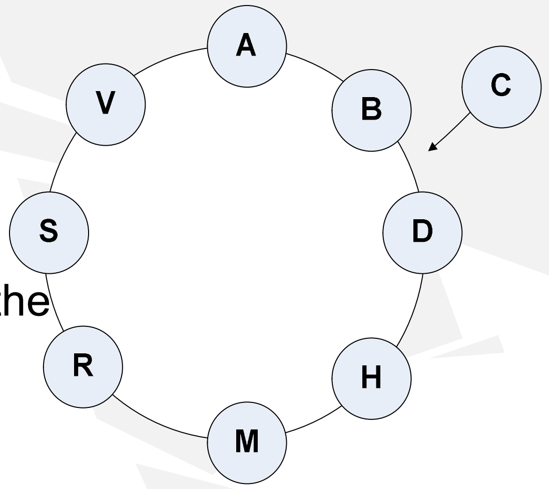
  (Watch for Startup complete)

CQL – Cassandra Query Language

- CQL - Cassandra Query Language
  - DDL and Query language for Cassandra
  - Very similar to SQL, but not SQL
- SELECT * FROM users;
- SELECT does not allow JOINS (involving more than 1 table)
- Copy - imports/exports data from .csv files
- The native types supported by CQL are:
- ASCII | BIGINT | BLOB | BOOLEAN | COUNTER | DATE |
- DECIMAL | DOUBLE | DURATION | FLOAT |
- INET | INT | SMALLINT |
- TEXT | TIME | TIMESTAMP | TIMEUUID | TINYINT |
- UUID | VARCHAR | VARINT

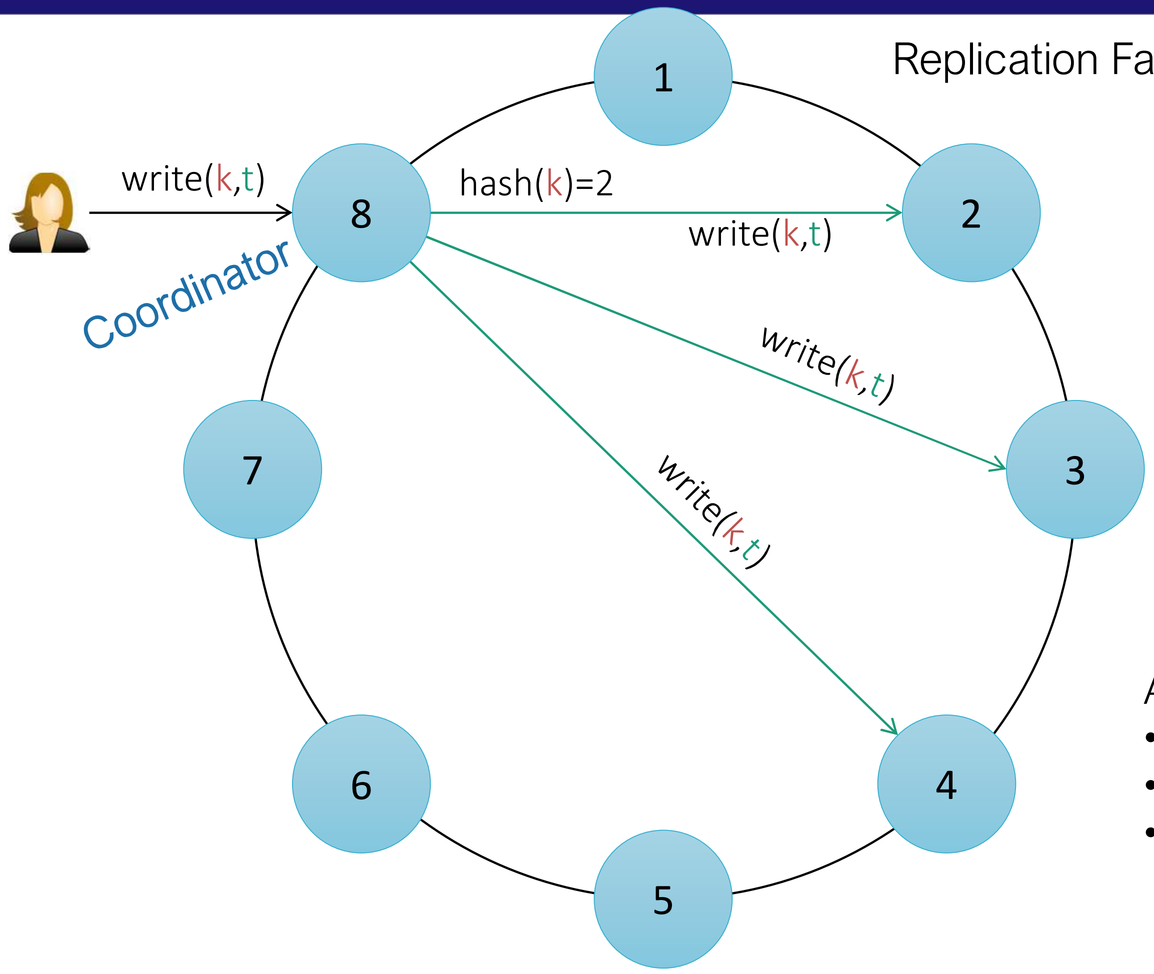
https://cassandra.apache.org/doc/latest/cassandra/cql/types.html

Cluster: Ring



- Cassandra Ring - Clustering for scaling
- Clients can connect to any node of the cluster
- The connected node becomes the **coordinator** for the client
- The coordinator is to act as a proxy between the client application and the nodes (or replicas) that own the data being requested
- Each node is assigned different ranges of data - Token ranges
- *(1) nodetool describecluster (2) nodetool ring* (3) *nodetool describering table*
- The coordinator routes the data to the destination node based on the token
- The partitioner does the job of creating the tokens
- Nodes can be added to the cluster online (redistributes token ranges)
- Nodes can be removed from a live cluster - decommissioning

Cassandra’s Ring Model



Replication Factor = 3

Advantages:

- Flexibility
- Ease of cluster scaling
- Auto Sharding

Distributed Architecture: Snitch

- Snitch controls the node placement in data centers
- Determines/declares each node's rack and data center
- Defines the topology of the cluster
- Several different types of snitches (Eg: SimpleSnitch)
    - Ec2Snitch - for cloud
    - Property File Snitch (for big clusters)
    - Gossiping Property File Snitch
        - In cassandra-rackdc.properties file you put the data center this node belongs to and the rack that it belongs to.
- Configured in cassandra.yaml
- ***nodetool describecluster***

Fault Tolerance and High availability

Cassandra allows to :

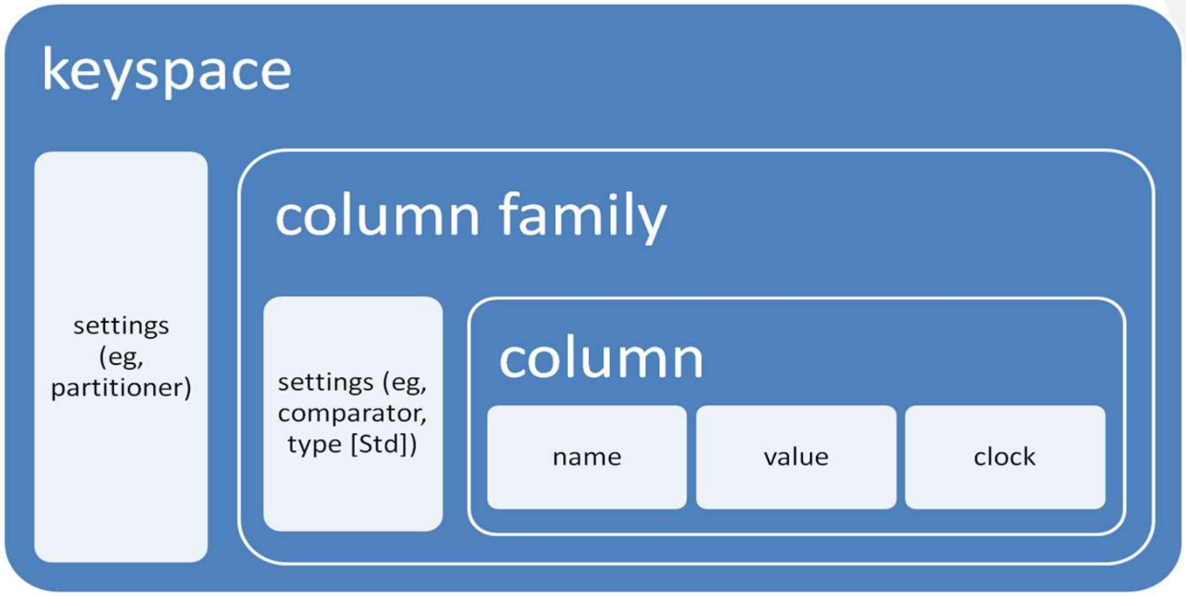
- Replace failed nodes with no downtime
    - Steps to replace a node that has died for some reason, such as hardware failure.
    - Extra steps are required for replacing dead seed nodes.

  https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/operations/opsReplaceNode.html

- Replicate data to multiple data centers to ensure HA for data

Cassandra Data Model

- Keyspace ~= database , typically one per application
- Some settings are configurable only per keyspace
- Row oriented - Each row is uniquely identified by a key

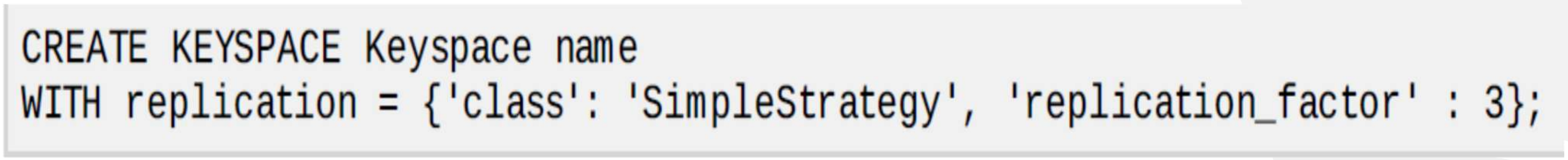

keyspace
settings (eg, partitioner)
column family
settings (eg, comparator, type [Std])
column
name
value
clock

Keyspaces

Key spaces are

- Outermost container for **data** in Cassandra
- Top-level Namespace / Containers
- A bunch of attributes which define keyspace-wide behavior
- Stores the replication information
- Within the Keyspaces you have Table / Columnfamily
- CQL command ***use*** can be used to switch between different keyspaces
- Closest match is database in Relational Databases

Creating Keyspaces



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

- Replication factor = N and data is stored in N nodes of cluster
- Replica placement strategy – Strategy to place replicas in the ring.
  - Simplestrategy - *rack – awarestrategy*
  - Network topology strategy - *datacenter – sharedstrategy*.
- NetworkTopology
  - Specify replication factor per DC where we want reliability from DC failures
  - e.g. CREATE KEYSPACE bds2 WITH replication = {'class': 'NetworkTopologyStrategy', 'eastDC' : 2, ‘westDC' : 3};

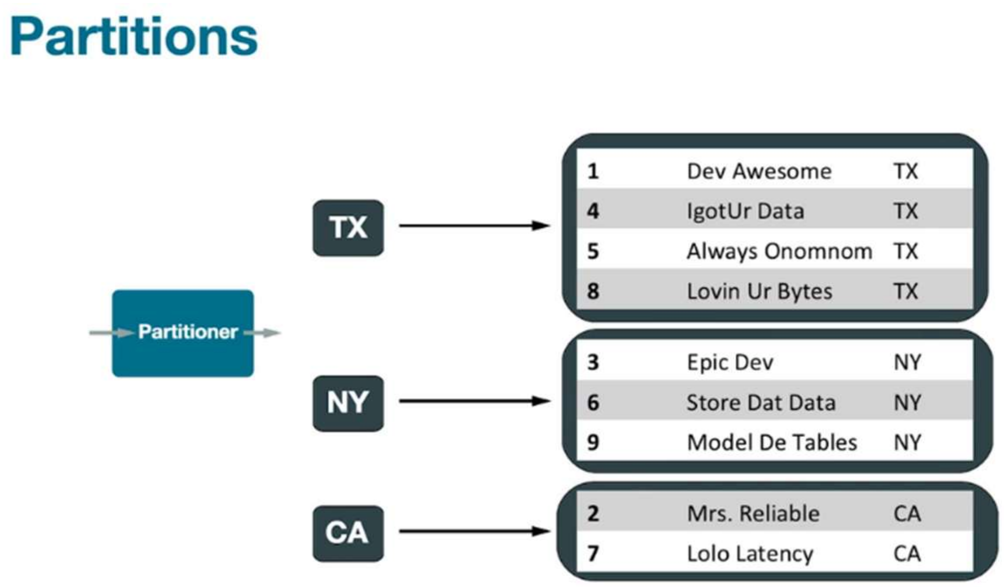
Cassandra Data Model – High level goals

- **Spread data evenly around the cluster** – For Cassandra to work optimally, data should be spread as evenly as possible across cluster nodes.
  - Distributing data evenly depends on selecting a good partition key and Partitioner.
- **Minimize the number of partitions to read** – When Cassandra reads data, it's best to read from as few partitions as possible since each partition potentially resides on a different cluster node.
  - If a query involves multiple partitions, the coordinator node responsible for the query needs to interact with many nodes, thereby reducing performance.
- **Anticipate how data will grow, and think about potential bottlenecks in advance** – A particular data model might make sense when you have a few hundred transactions per user, but what would happen to performance if there were millions of transactions per user?
  - Always "think big" when building data models for Cassandra and avoid knowingly introducing bottlenecks.

Reference: Data Modeling in Apache Cassandra™ - Five Steps to an Awesome Data Model

Cassandra Data Model: Partitions

- If only one primary key, it becomes the partition key
- Partitioner finds out a numeric Token value from the key



1	Dev Awesome	TX
4	IgotUr Data	TX
5	Always Onomnom	TX
8	Lovin Ur Bytes	TX
3	Epic Dev	NY
6	Store Dat Data	NY
9	Model De Tables	NY
2	Mrs. Reliable	CA
7	Lolo Latency	CA

Partitioners

Murmur3Partitioner (default): Uniformly distributes data across the cluster based on MurmurHash hash values.

RandomPartitioner: Uniformly distributes data across the cluster based on MD5 hash values.

ByteOrderedPartitioner: Keeps an ordered distribution of data lexically by key bytes

Access Keys for Cassandra Tables

Partition Key – Single Column

- These types of tables have primary keys that are also partition keys.
- Eg: **PRIMARY KEY** (State)

Partition Key – Multiple Columns

- These types of tables have primary keys composed of partition and clustering keys
- Consists of one or more partition keys and zero or more clustering key components
- Always puts the partition key first and then the clustering keys
- Eg: **PRIMARY KEY** (State, AccountType, AccountNo)

Composite Partition keys

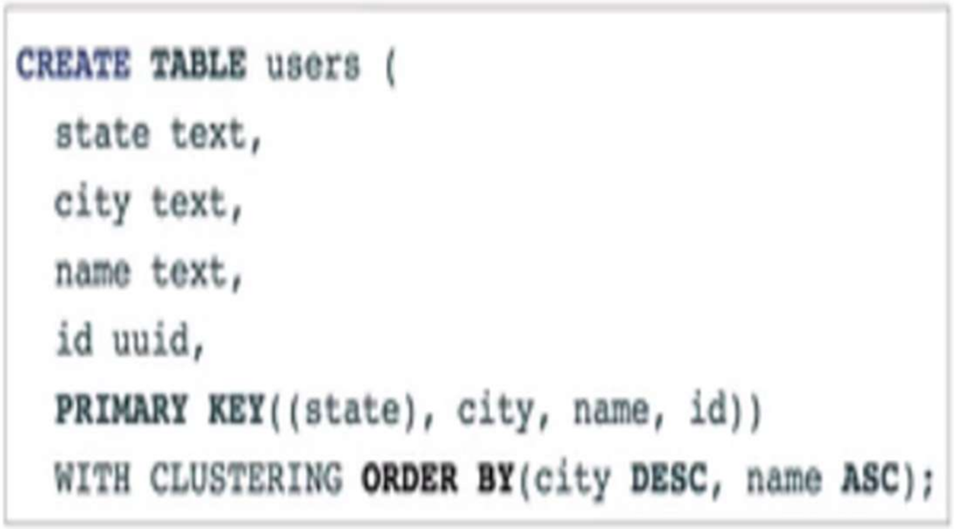
- A partition key in Cassandra can be made up of multiple columns.
- **Eg: PRIMARY KEY** ((State,AccountType), AccountNo)

Cassandra Data Model: Clustering Columns

- Clustering is the process of storing the data within a partition and is based on the columns defined as the clustering keys.
- The selection of clustering key columns depends on how we want to use the data in our application.
- All data within a partition is stored in continuous storage, sorted by clustering key columns
- Clustering columns follow ascending order by default
- Clustering allows us to change default ordering using ORDER BY
- Change ordering direction via WITH CLUSTERING ORDER BY
- Ordering is done when data is stored

Allow Filtering

- Relaxes the querying on partition key constraint
- Causes Cassandra to scan all partitions
- Do not use it
  - Unless you really have to
  - Best on small data sets



```
CREATE TABLE users (
  state text,
  city text,
  name text,
  id uuid,
  PRIMARY KEY((state), city, name, id))
  WITH CLUSTERING ORDER BY(city DESC, name ASC);
```

Replication/Consistency: Hinted Handoff

- When Node is offline, what happens to the data we go to write there
- Cassandra is Always writable: Hinted Handoff mechanism
- If any replica node is down, the coordinator writes to all other replicas, and keeps the write locally until down replica node comes back up.

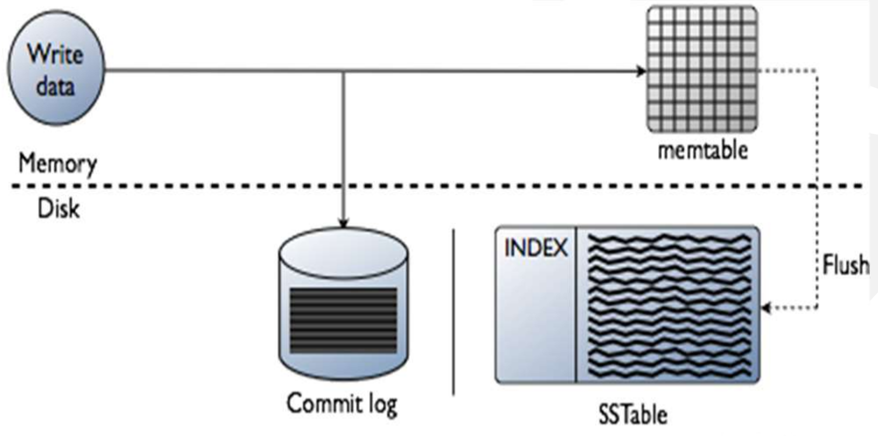
  "*I have the write information that is intended for node B. I'm going to hang onto this write, and I'll notice when node B comes back online; when it does, I'll send it the write request*"
- When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours ~ 3 hours).
- When the node comes back it can write the hints to that node
- If we want to reboot a node, we are free to do that

Memtables, SSTables, and Commit Logs

- Commit Log for Durability
  - Once written never lost
  - Commit logs :all writes go in for recovery
- Memtable
  - memory-resident data structure
  - When contents become too big. Flushed into SStable
- SSTable : File in Harddisk

Internal Architecture: Write-Path

- A client issues a write request to the coordinator node in the Cassandra cluster.
- The "Partitioner" determines the nodes responsible for the data
- On receiving a Write request, log it in disk commit log
- Make changes to appropriate memtables – In-memory representation of multiple key-value pairs



Manual flush - nodetool flush

- Later, when memtable is full or old, flush them to disk , remove commitLog
- Data File: An SSTable (Sorted String Table) – list of key value pairs, sorted by key
- Index file: An SSTable – (key, position in data sstable) pairs

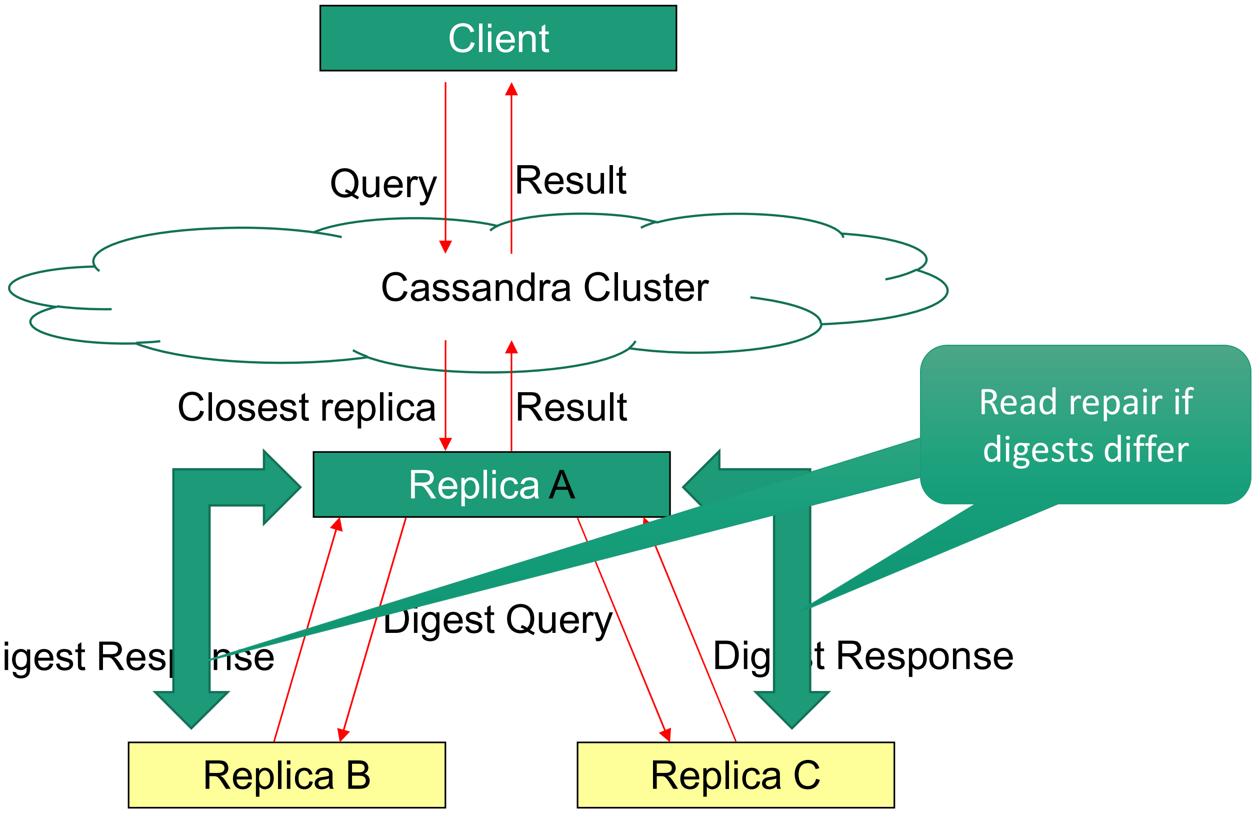
Write / Read

- Writes
  - Written to commit log sequentially and deemed successful
  - Data is indexed and put into in-memory Memtable (one or more per Column Family)
  - Memtable is flushed to disk SSTable file
  - SSTable is immutable and append only
  - Partitioning and replication happens automatically
- Reads
  - Client connects to any node to read data
  - Consistency level decides when a read is returned, i.e. how many replicas should contain the same copy
  - Read repair: replication via a Gossip protocol is triggered as a client issues a read and Cassandra has to meet the required consistency level

Internal Architecture: Read-Path

- Read: Similar to writes, except
  - Need to touch commit log and multiple SSTables
  - Coordinator can contact closest replica (e.g., in same rack)
  - Coordinator also fetches from multiple replicas
    - Check consistency in the background, initiating a read-repair if any two values are different
    - Makes read slower than writes (but still fast)
    - Read repair: uses data digests
  - Slower than writes
- Delete: don’t delete item right away
  - Add a tombstone to the log
  - Compaction will remove tombstone and delete item

Internal Architecture: Read Operation



Internal Architecture: Compaction

- Compaction is delayed IO on your file, Cleaning up the disk space
- Compaction operations occur periodically to re-write and combine SSTables.
- Required because SSTables are immutable (no modifications once written)
- Compactions prune deleted data and merge disparate row data into new SSTables in order to reclaim disk space and keep read operations optimized
- Multiple Compaction Strategies are included with Cassandra, and each is optimized for a different use case:

1. SizeTiered Compaction Strategy (STCS) (Default) - Good for insert-heavy and general workloads
2. Leveled Compaction Strategy (LCS) - Best for read-heavy workloads
3. DateTiered Compaction Strategy (DTCS) - Designed for use with time-series data. stores data written within the same time period in the same SSTable.

- From cqlsh , compaction strategy can be changed
- ALTER TABLE fresher.world2 WITH compaction = { 'class' : 'LeveledCompactionStrategy'};
- ALTER TABLE fresher.world2 WITH compaction = { 'class' : 'SizeTieredCompactionStrategy'};

The Art of Computer Programming - Volume 3 Sorting and Searching - Donald E Knuth

Compaction Strategies

- LeveledCompactionStrategy (LCS): (Best for Read heavy workloads)
  - The leveled compaction strategy creates SSTables of a fixed, relatively small size (160 MB by default) that are grouped into levels. Within each level, SSTables are guaranteed to be non-overlapping. Each level (L0, L1, L2 and so on) is 10 times as large as the previous. Disk I/O is more uniform and predictable on higher than on lower levels as SSTables are continuously being compacted into progressively larger levels. At each level, row keys are merged into non-overlapping SSTables in the next level. This process can improve performance for reads, because Cassandra can determine which SSTables in each level to check for the existence of row key data. This compaction strategy is modeled after Google's LevelDB implementation. Also see LCS compaction subproperties.
- SizeTieredCompactionStrategy (STCS): (Best for Write heavy workloads)
  - The default compaction strategy. This strategy triggers a minor compaction when there are a number of similar sized SSTables on disk as configured by the table subproperty, min_threshold. A minor compaction does not involve all the tables in a keyspace. Also see STCS compaction subproperties.
- TimeWindowCompactionStrategy (TWCS): (Best for Time Series Workloads)
  - TWCS compacts SSTables using a series of time windows. While with a time window, TWCS compacts all SSTables flushed from memory into larger SSTables using STCS. At the end of the time window, all of these SSTables are compacted into a single SSTable. Then the next time window starts and the process repeats. The duration of the time window is the only setting required. See TWCS compaction subproperties. This strategy is an alternative for time series data.

Types of Compaction

- Minor compaction - triggered automatically in Cassandra
  - When is a minor compaction triggered?
  - When an SSTable is added to the node through flushing/streaming
  - When autocompaction is enabled after being disabled (nodetool enableautocompaction)
  - When compaction adds new SSTable
  - Checks for new minor compactions every 5 minutes
- Major compaction - User executes a compaction over all SSTables on the node
- User defined compaction - User triggers compaction on a given set of SSTables
- Scrub - try to fix any broken SSTables. This can actually remove valid data if that data is corrupted, if that happens you will need to run a full repair on the node.

Configuring Compaction Strategies

Use CREATE TABLE or ALTER TABL to configure compaction strategy property:

- Maximum number of SSTables to compact
- Minimum SSTable size

1. Update a table to set the compaction strategy using the ALTER TABLE

```
ALTER TABLE users WITH
  compaction = { 'class' :  'LeveledCompactionStrategy’  }
```

2. Change the compaction strategy property to SizeTieredCompactionStrategy and specify the minimum number of SSTables to trigger a compaction using the CQL min_threshold attribute.

```
ALTER TABLE users
  WITH compaction = {'class' : 'SizeTieredCompactionStrategy', 'min_threshold' : 6 }
```

3. Nodetool commands

```
    nodetool compact
    nodetool compactionhistory
```

Replication/Consistency: Read-repair

- Network partitions can cause nodes to get out of sync
- Hinted-handoff works only for 3 hours
- Read repair is the process of making sure your data is consistent across all replicas
- It finds an answer to the question "*Oh, is this data consistent with the rest of the replicas?*" at the time of read
- Read repair is done by comparing the timestamps
- Read repair happens for reads with consistency level < ALL
- Choose data with latest timestamp, return that data to the client and update it on all other nodes
- Repair manually at least once in 10 days using ***nodetool repair*** command - (repairs one or more tables)

Sample queries

```
> create keyspace demo with replication={'class':'SimpleStrategy',
'replication_factor':1};

> describe keyspaces;

> use demo;              or columnfamily

> create table student_info (rollno int primary key, name text, doj
timestamp, lastexampercent double);

> describe table student_info ;

> consistency quorum

> insert into student_info (rollno,name,doj,lastexampercent) values
(4,'Roxanne', dateof(now()), 90) using ttl 30;

> select rollno from student_info where name='Roxanne' ALLOW
FILTERING;

> update student_info set lastexampercent=98 where rollno=2 IF
name='Sam';
```

Cassandra Certification Trainings

Free Cassandra courses – Datastax Academy - https://www.datastax.com/dev/academy

Multiple Learning Paths - Gain an expert understanding of Apache Cassandra™

Each Learning Path is composed of a sequence of recommended courses for your role

- Administrator Certification
  - DS201
  - DS210
- Developer Certification
  - DS201
  - DS220

Application Connectivity: Drivers

- Driver connects to Cassandra database from your application
  - Datastax driver (Token Range Aware)
  - Thrift client driver - (nodetool enablethrift)
  - Similar to JDBC for RDBMS
- Driver listens and manages changes to the cluster (Node addition/deletion etc.)
- Driver specifies multiple connection points to the cluster
- Setup
  - Create a cluster object
  - Use the cluster to obtain a session
  - Session manages connection to the cluster
  - Insert - session.execute(“ CQL command for data insertion”)
  - Select – result = session.execute(“SELECT * from mytable …..”)
  - Print result.Firstname, result.Lastname

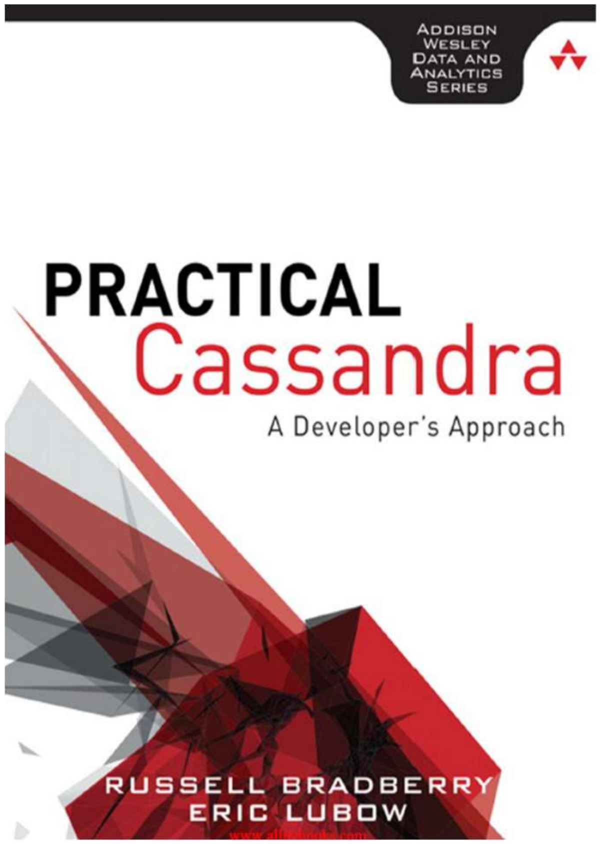
Handson with CQLSH

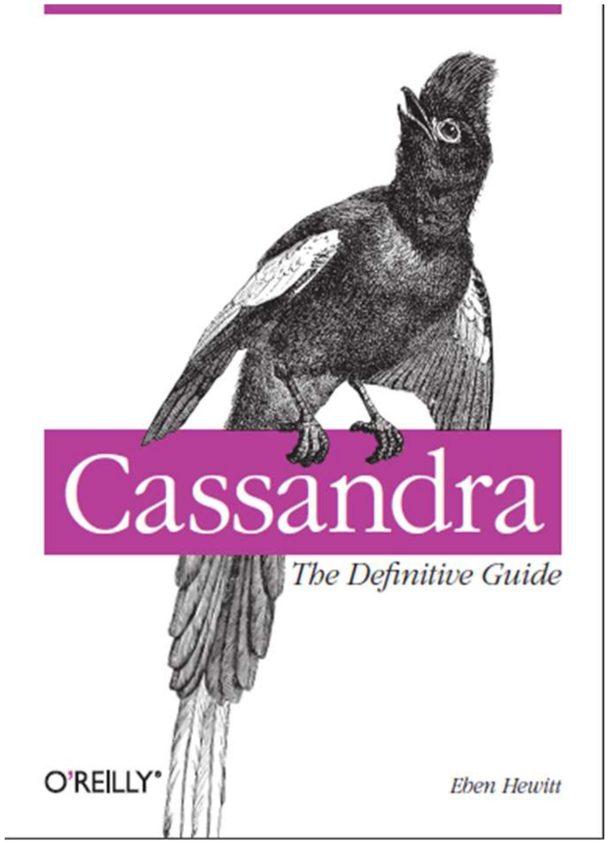
- Creating Keyspaces and Column families
- Demonstration of TTL
- Flexible schema
- Updata, Insert, Delete on tables
- Load data from .csv files
- Aggregation
- Indexing
- Dropping tables and Keyspaces

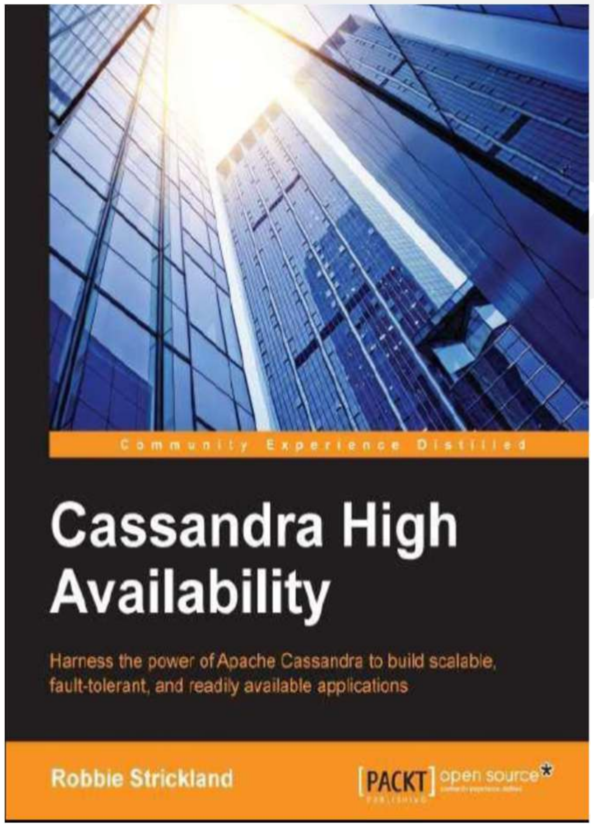
Cassandra on Cloud

- Amazon Keyspaces (for Apache Cassandra)
- Azure Managed Instance for Apache Cassandra
- DataStax Astra DB for Apache Cassandra (Google Cloud)

Cassandra Books






Community Experience Distilled
Cassandra High Availability
Harness the power of Apache Cassandra to build scalable, fault-tolerant, and readily available applications
Robbie Strickland
[PACKT] open source

Hands on with Cassandra

49

Next Session:
Introduction to Neo4j Graph database