Cassandra

Analysis Document

Overview

- Apache Cassandra™, is an Apache Software Foundation project, is a
    1. Massively scalable
    2. NoSQL database.
    3. Designed to handle big data workloads across multiple data centers
    4. With no single point of failure,
    5. Providing enterprises with extremely high database performance and availability.

- Cassandra does not use a master/slave architecture, but instead uses a peer-to-peer implementation, which avoids the pitfalls, latency problems, single point of failure issues, and performance headaches associated with master/slave setups.

- Cassandra's architecture make it perfect for full cloud deployments as well as hybrid implementations that store some data in the cloud and other data on-premises.
- A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

Architecture Level

Cassandra has been architecture for consuming large amounts of data as fast as possible. To accomplish this, Cassandra first writes new data to a commit log to ensure it is safe. After that, the data is then written to an in-memory structure called a memtable. Cassandra deems the write successful once it is stored on both the commit log and a memtable, which provides the durability required for mission-critical systems.

Once a memtable memory limit is reached, all writes are then written to disk in the form of an SSTable (sorted strings table). An SSTable is immutable, meaning it is not written to ever again. If the data contained in the SSTable is modified, the data is written to Cassandra in an upsert fashion and the previous data automatically removed.

Because SSTables are immutable and only written once the corresponding memtable is full, Cassandra avoids random seeks and instead only performs sequential IO in large batches, resulting in high write throughput.

Cassandra is architected in a peer-to-peer fashion and uses a protocol called "gossip" to communicate with other nodes in a cluster. The gossip process runs every second to exchange information across the cluster.

Gossip only includes information about the cluster itself (e.g., up/down, joining, leaving, version, schema) and does not manage the data. Data is transferred node-to-node using a message-passing like protocol on a distinct port from what client applications connect to. The Cassandra partitioner turns a column family key into a token, the replication strategy picks the set of nodes responsible for that token (using information from the snitch) and Cassandra sends messages to those replicas with the request (read or write).

Links

- http://www.tutorialspoint.com/cassandra/cassandra_architecture.htm
- http://www.datastax.com/resources/faq#intro-1
- http://cassandra.apache.org/

Key Terms and Concepts

Node - It is the place where data is stored.
Data center - It is a collection of related nodes.
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 to, from 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.
Compaction - The process of freeing up space by merging large accumulated data files is called compaction. During compaction, the data is merged, indexed, sorted, and stored in a new SSTable. Compaction also reduces the number of required seeks.

Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica; in case of a failure, replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of Keyspace in Cassandra are:
Replication factor - It is the number of machines in the cluster that will receive copies of the same data.
Replica placement strategy - It is nothing but the strategy to place replicas in the ring. We have strategies such as

1. simple strategy (rack aware strategy),
2. old network topology strategy (rack-aware strategy), and
3. network topology strategy (datacenter-shared strategy).

Column families

When comparing Cassandra to a relational database, the column family is similar to a table in that it is a container for columns and rows. However, a column family requires a major shift in thinking for those coming from the relational world.

In a relational database, you define tables, which have defined columns. The table defines the column names and their data types, and the client application then supplies rows conforming to that schema: each row contains the same fixed set of columns.
In Cassandra, you define column families. Column families can (and should) define metadata about the columns, but the actual columns that make up a row are determined by the client application. Each row can have a different set of columns.

Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Super Column

A Cassandra column family can contain regular columns (key/value pairs) or super columns. Super columns add another level of nesting to the regular column family column structure. Super columns are comprised of a (super) column name and an ordered map of sub-columns. A super column is a way to group multiple columns based on a common lookup value.

Cassandra query language shell (cqlsh)

Cassandra 0.8 is the first release to introduce Cassandra Query Language (CQL), the first standardized query language for Apache Cassandra. CQL pushes all implementation details to the server in the form of a CQL parser. Clients built on CQL only need to know how to interpret query result objects. CQL is the start of the first officially supported client API for Apache Cassandra. CQL drivers for the various languages are hosted within the Apache Cassandra project.
CQL syntax in based on SQL (Structured Query Language), the standard for relational database manipulation. Although CQL has many similarities to SQL, it does not change the underlying Cassandra data model. There is no support for JOINs, for example.

Links

· http://www.tutorialspoint.com/cassandra/cassandra_architecture.htm

Version Details

Cassandra DB can be provided by 2 places

1) Cassandra official Site (http://cassandra.apache.org/download/)
    a. Latest Version 3.11

2) DataStax – Third party distributor

a. DataStax Community Edition Apache Cassandra
b. DataStax Enterprise 4.7 (Paid)

Links

· http://docs.datastax.com/en/cassandra/2.1/cassandra/features2.html

Java Client Driver

The Java driver is a modern, feature-rich and highly tunable Java client library for Apache Cassandra (1.2+) and DataStax Enterprise (3.1+) using exclusively Cassandra's binary protocol and Cassandra Query Language v3.
Use this driver in production applications to pass CQL statements from the client to a cluster and retrieve, manipulate, or remove data. Cassandra Query Language (CQL) is the primary language for communicating with the Cassandra database. Documentation for CQL is available in CQL for Cassandra 2.x. DataStax also provides DataStax DevCenter, which is a free graphical tool for creating and running CQL statements against Apache Cassandra and DataStax Enterprise. Other administrative tasks can be accomplished using OpsCenter.
The driver is compatible with all versions of Cassandra 1.2 and later.

- **Cassandra 2.1 support**
  - User-defined types (UDT)
  - Tuple type
- **New features**
  - Simple object mapping API

Links

· Maven Link:
http://mvnrepository.com/artifact/com.datastax.cassandra/cassandra-driver-core/2.1.5
· Documentation Guide:
http://docs.datastax.com/en/developer/java-driver/2.1/java-driver/whatsNew2.html
· Source Code: https://github.com/datastax/java-driver

Data Types

CQL data types to Java types

Java classes to CQL data types

CQL3 data type	Java type
ascii	java.lang.String
bigint	long
blob	java.nio.ByteBuffer
boolean	boolean
counter	long
decimal	java.math.BigDecimal
double	double
float	float
inet	java.net.InetAddress
int	int
list	java.util.List<T>
map	java.util.Map<K, V>
set	java.util.Set<T>
text	java.lang.String
timestamp	java.util.Date
timeuuid	java.util.UUID
tuple	com.datastax.driver.core.TupleType
uuid	java.util.UUID
varchar	java.lang.String
varint	java.math.BigInteger

Links

.

http://docs.datastax.com/en/developer/java-driver/2.1/java-driver/reference/javaClass2Cql3Datatypes_r.html

Installation

We have flowed following link to install Cassandra 2.0 using below link
http://exponential.io/blog/2015/01/19/install-cassandra-2_1-on-ubuntu-linux/

1. Download most stable release of Apache Cassandra is 2.0.15 from below link
   http://cassandra.apache.org/download/
2. Make directory in Linux system
   mkdir -p ~/opt/packages && cd $_
3. Copy apache-cassandra-2.0.14-bin.tar.gz file at this location using WinSCP
4. Unzip the file using following command

gzip -dc apache-cassandra-2.0.14-bin.tar.gz | tar xf -
ln -s ~/opt/packages/apache-cassandra-2.0.14 ~/opt/cassandra
5, Create log directory
mkdir -p ~/opt/cassandra/data/data
mkdir -p ~/opt/cassandra/data/commitlog
mkdir -p ~/opt/cassandra/data/saved_caches
mkdir -p ~/opt/cassandra/logs

The installation of Cassandra is completed

Commands to start/stop the DB

Server

Start Server:
~/opt/cassandra/bin/cassandra –f
Stop Server:
Control + C

Client

Start Client:
~/opt/cassandra/bin/cqlsh
Stop Client:
exit

Security

Authentication

Introduction

Internal authentication is based on Cassandra-controlled login accounts and passwords. Internal authentication equates to having user login accounts and their passwords being managed inside Cassandra.Internal authentication stores usernames and bcrypt-hashed passwords in the system_auth.credentials table.
Authentication backend, implementing IAuthenticator; used to identify users Out of the box, Cassandra provides org.apache.cassandra.auth.

Types of Authentication

AllowAllAuthenticator performs no checks - set it to disable authentication.

PasswordAuthenticator relies on username/password pairs to authenticate users. It keeps usernames and hashed passwords in system_auth.credentials table. Please increase system_auth keyspace replication factor if you use this authenticator.

Configuring Authentication

1.Change the authenticator option in the cassandra.yaml to PasswordAuthenticator.
By default, the authenticator option is set to AllowAllAuthenticator.
authenticator: PasswordAuthenticator

2. Increase the replication factor for the system_auth keyspace to N (number of nodes).
If you use the default, 1, and the node with the lone replica goes down, you will not be able to log into the cluster because the system_auth keyspace was not replicated.

3. Restart the Cassandra client.
The default superuser name and password that you use to start the client is stored in Cassandra.
<client startup string> -u cassandra -p Cassandra

4. Start cqlsh using the superuser name and password.
./cqlsh -u cassandra -p Cassandra

CQL Commands

Creating Super User

```
cqlsh> create user admin with password 'admin' superuser;
CREATE ROLE admin WITH PASSWORD = admin
AND SUPERUSER = true
 AND LOGIN = true;
```

Creating Normal User

```
cqlsh> create user laura with password 'newhire';
```

Getting list of Login Account

```
cqlsh> list users;
name    | super
----------+-------
cassandra | True
laura     | False
robin     | True
```

Dropping User

```
cqlsh> drop user laura;
```

Login using cqlsh

Typically, after configuring authentication, you log into cqlsh using the -u and -p options to the cqlsh command. To avoid having enter credentials every time you launch cqlsh, you can create a cqlshrc file in the .cassandra directory, which is in your home directory. When present, this file passes default login information to cqlsh.

Steps:

1. Open a text editor and create a file that specifies a user name and password.
2. [authentication]
3. username = fred

password = !!bang!!$

4. Save the file in your home/.cassandra directory and name it cqlshrc.
5. Set permissions on the file.

To protect database login information, ensure that the file is secure from unauthorized access.

Note: Sample cqlshrc files are available in:

- Packaged installations

/usr/share/doc/dse-libcassandra

- Binary installations

install_location/conf

Java Client to access Database with Credentials

Invalid Credentials:
Exception in thread "main" com.datastax.driver.core.exceptions.AuthenticationException: Authentication error on host /10.44.189.225:9042: Username and/or password are incorrect

Authorization

Introduction

· Internal authorization is where a user's permissions – what actions they can perform inside the database are enforced and managed inside of Cassandra.
· GRANT/REVOKE paradigm to grant or revoke permissions to access Cassandra data.
· A superuser grants initial permissions, and subsequently a user may or may not be given the permission to grant/revoke permissions.
· Authorization backend, implementing IAuthorizer; used to limit access/provide permissions Out of the box, Cassandra provides org.apache.cassandra.auth

Types of Authorization

· AllowAllAuthorizer allows any action to any user - set it to disable authorization.
· CassandraAuthorizer stores permissions in system_auth.permissions table. Please increase system_auth keyspace replication factor if you use this authorizer.

Configuring internal authorization

1. In the cassandra.yaml file, comment out the default AllowAllAuthorizer and add the CassandraAuthorizer.

```
authorizer: CassandraAuthorizer
```

You can use any authenticator except AllowAll.

1. Configure the replication factor for the system_auth keyspace to increase the replication factor to a number greater than 1.
2. Adjust the validity period for permissions caching by setting the permissions_validity_in_ms option in the cassandra.yaml file.

Alternatively, disable permission caching by setting this option to 0.

CQL Commands

Grant Command

Grant permission ON resource TO user

Revoke Command

Revoke permission ON resource FROM user;

Permission and Resource

- ALL
- ALTER
- AUTHORIZE
- CREATE
- DROP
- MODIFY
- SELECT

The various permission/resource combinations currently are:

Permission		CQL Statements
ALTER		ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX
AUTHORIZE		GRANT, REVOKE
CREATE		CREATE KEYSPACE, CREATE TABLE
DROP		DROP KEYSPACE, DROP TABLE
MODIFY		INSERT, DELETE, UPDATE, TRUNCATE
SELECT		SELECT

Example

```
cqlsh> create user laura with password 'newhire';
cqlsh> grant all on dev.emp to laura;
cqlsh> revoke all on dev.emp from laura;
cqlsh> grant select on dev.emp to laura;
```

The new ‘laura’ user account can login, read data from the emp table, but cannot perform other actions such as reading data from another table for which it has not been granted rights:

```
robinsmac:bin robin$ ./cqlsh -3 localhost -u laura -p newhire
Connected to Test Cluster at localhost:9160.
[cqlsh 2.2.0 | Cassandra 1.1.8.1-SNAPSHOT | CQL spec 3.0.0 | Thrift protocol 19.33.0]
Use HELP for help.
cqlsh> use dev;
```

```
cqlsh:dev> select * from emp;
empid | empname
------+---------
1     |  robin
2     |  laura

cqlsh:dev> select * from emp_bonus;

Bad Request: User laura has no SELECT permission on <table dev.emp_bonus> or any of its
parents
```

Finding out what login accounts possess what permissions on what objects and what objects have permissions granted on them is performed through various list commands (as long as the user has the authority to use them):

```
cqlsh> list all permissions of laura;
username | resource       | permission
---------+----------------+------------
laura    | <table dev.emp> |    SELECT

cqlsh> list all permissions on dev.emp;
username | resource       | permission
---------+----------------+------------
laura    | <table dev.emp> |    SELECT
```

Java Client to Test Authorization

Permission to Laura is given for demo. Playlits Table

```
grant all on demo.playlists to laura;
```

Output:

```
Exception in thread "main"
com.datastax.driver.core.exceptions.UnauthorizedException: User laura has no
SELECT permission on <table demo.users> or any of its parents
```

SSL Encryption

Client-to-node encryption protects data in flight from client machines to a database cluster using SSL (Secure Sockets Layer). It establishes a secure channel between the client and the coordinator node.

Prerequisites

All nodes must have all the relevant SSL certificates on all nodes.

To enable client-to-node SSL, you must set the client_encryption_options in the cassandra.yaml file.

Procedure

On each node under client_encryption_options:

- Enable encryption.
- Set the appropriate paths to your .keystore and .truststore files.
- Provide the required passwords. The passwords must match the passwords used when generating the keystore and truststore.
- To enable client certificate authentication, set require_client_auth to true

Steps

1. Create private key and keystore

```
cd /opt/jdk1.7.0_75/jre/bin
./keytool -genkey -keyalg RSA -alias cassandraKey -keystore privateKey.keystore
```

2. Generate a certificate file

```
./keytool -export -alias cassandraKey -file cassandra.cer -keystore privateKey.keystore
```

3. Set client_encryption_options

```
client_encryption_options:
enabled: true
          keystore: /root/opt/packages/apache-cassandra-2.0.14/conf/privateKey.keystore
      keystore_password: cassandra
                  require_client_auth: false
```

4. Add certificate in cacerts of local system

```
C:\Program Files\Java\jdk1.7.0_45\jre\bin>keytool.exe -list -keystore ..\lib\security\cacerts
```

Connect with CQLSH

Java Code to connect with SSL Details

Replication structure

Steps to Configure Cluster

1. Suppose you install Cassandra on these nodes:

node0: 10.44.189.225(seed1)
node1: 10.222.188.104

Note: It is a best practice to have more than one seed node per data center.

2. If you have a firewall running in your cluster, you must open certain ports for communication between the nodes.
service iptables stop

3. Stop Cassandra Server
ps auwx | grep cassandra
$ sudo kill pid
4. Change the cassandra.yaml file
Properties to set:

- **num_tokens**: *recommended value: 256*
- **-seeds:** *internal IP address of each seed node*

Seed nodes do not bootstrap, which is the process of a new node joining an existing cluster. For new clusters, the bootstrap process on seed nodes is skipped.

- **listen_address:**

If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the listen_address.

- **endpoint_snitch:** *name of snitch* (See endpoint_snitch.) If you are changing snitches, see Switching snitches.
- auto_bootstrap: false (Add this setting **only** when initializing a fresh cluster with no data.)

Note: If the nodes in the cluster are identical in terms of disk layout, shared libraries, and so on, you can use the same copy of the cassandra.yaml file on all of them.
Example:

```
cluster_name: 'Test Cluster'
num_tokens: 256
initial_token: 0
seed_provider:
        - class_name: org.apache.cassandra.locator.SimpleSeedProvider
        parameters:
        - seeds: "10.44.189.225"
listen_address: 10.44.189.225
rpc_address: 0.0.0.0
endpoint_snitch: GossipingPropertyFileSnitch
```

5. In the cassandra-rackdc.properties file, assign the data center and rack names you determined in the Prerequisites. For example:

```
# indicate the rack and dc for this node
dc=DC1
rack=RAC1
```

6. Start Cassandra Server
~/opt/cassandra/bin/cassandra –f
7. Check status of cluster

./nodetool status

Java Code

When we add multiple nodes address, the cluster first tries to attempt first node in list. If it fails then moves to next and so on.
If the node is down by any reason then cluster will move forward.
Once the cluster is initialized with one node details, it will have the details of complete cluster. But if this connected node goes down after initializing cluster object then the further code fails.

Description of addContactPoint():

Contact points are addresses of Cassandra nodes that the driver uses to discover the cluster topology. Only one contact point is required (the driver will retrieve the address of the other nodes automatically), but it is usually a good idea to provide more than one contact point, because if that single contact point is unavailable, the driver cannot initialize itself correctly.
Note that by default (that is, unless you use the withLoadBalancingPolicy(com.datastax.driver.core.policies.LoadBalancingPolicy)) method of this builder), the first successfully contacted host will be use to define the local data-center for the client. If follows that if you are running Cassandra in a multiple data-center setting, it is a good idea to only provided contact points that are in the same datacenter than the client, or to provide manually the load balancing policy that suits your need.
When we add multiple node address, the cluster first tires to attempt first node in list. If it fails then moves to next and so on.

Different POC's

The java drivers have wide range of class and methods that allow us to provide various operations on Cassandra Database. Below are the details of some POC's that we did.

Connection to Cluster

The default cluster provided by Cassandra is Test Cluster. Whenever client connects to Cassandra it is nothing but a node in a default Test Cluster.

Command Prompt

a. Connect without Credentials
   ~/opt/cassandra/bin/cqlsh
b. Connect with Credentials
   ~/opt/cassandra/bin/cqlsh -u cassandra -p cassandra

Java Code

a. Connect without Credentials
b. Connect with Credentials

Session Object

Whenever we connect to Cassandra from command prompt, we are in one session until we exit from the Cassandra
In Java code, there are two method to maintain session object

Method 1: Cluster level

Session = cluster.connect();

Here the session object is created at cluster level.

Method 1: Keyspace level

After execution this statement session will point to a particular keyspace.
For execution of further commands, the code will search for table in ‘demo’ keyspace.

Keyspace

Create Keyspace

Command Prompt

CREATE KEYSPACE IF NOT EXISTS store WITH replication = {'class':'SimpleStrategy', 'replication_factor':3};

Java Code

Alter Keyspace

Command Prompt

ALTER KEYSPACE test WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};

Java Code

Drop Keyspace

Command Prompt

DROP KEYSPACE test

Java Code

Table

Create Table

Command Prompt

```
CREATE TABLE emp(
  emp_id int PRIMARY KEY,
  emp_name text,
  emp_city text,
  emp_sal varint,
  emp_phone varint
  );
```

Java Code

Alter Table

Command Prompt

```
ALTER TABLE emp ADD emp_email text;
```

Java Code

Truncate Table

Command Prompt

TRUNCATE emp;

Java Code

Delete Table

Command Prompt

DROP TABLE emp;

Java Code

CRUD Operations

Insert Data

Command Prompt

INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

Java Code

Read Data

Command Prompt

Select * from emp;

Java Code

Update Data

Command Prompt

UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;

Java Code

Delete Data

Command Prompt

DELETE emp_sal FROM emp WHERE emp_id=3;

Java Code

Connection Pooling Options

Java Code

Batch Processing

Batch Processing allows us to group related SQL statements into a batch and submit them with one call to the database.
When we send several SQL statements to the database at once, it reduces the amount of communication overhead, thereby improving performance.

Command Prompt

BEGIN BATCH INSERT INTO users (firstname, lastname, age, city) VALUES ('Ran', 'Gill', 46, 'LA'); UPDATE users SET age = 5999 WHERE lastname='Doe'; DELETE city FROM users WHERE lastname='Doe'; APPLY BATCH;

Java Code

Handling Collection Datatypes

Create Table with List

Command Prompt

CREATE TABLE data (name text PRIMARY KEY, email list<text>);

Java Code

Insert List in Table

Command Prompt

INSERT INTO data(name, email) VALUES ('ramu',['abc@gmail.com','cba@yahoo.com'])";

Java Code

Update List in Table

Command Prompt

UPDATE data SET email = email +['xyz@tutorialspoint.com'] where name = 'ramu';

Java Code

Read List in Table

Command Prompt

SELECT * FROM data

Java Code

Exception Handling

For all Java API there is a exception type, details are given in the Link. Below is a brief NoHostAvailableException, QueryExecutionException, QueryValidationException, and UnsupportedFeatureException all extend DriverException which is a RuntimeException which is an unchecked exception.

Java Code

For a batch stmt, if one stmt fails, rest all fails.

Links

- http://stackoverflow.com/questions/30052030/cassandra-error-handling
- http://docs.datastax.com/en/drivers/java/2.1/com/datastax/driver/core/Session.html#execute%28com.datastax.driver.core.Statement%29
- https://www.npmjs.com/package/node-cassandra-cql

Metadata Support

We have wide range of metadata method available at Keyspace, table, column level.

Result Set to XML

Java Code

Questions And Answer

1. What is Cassandra ?

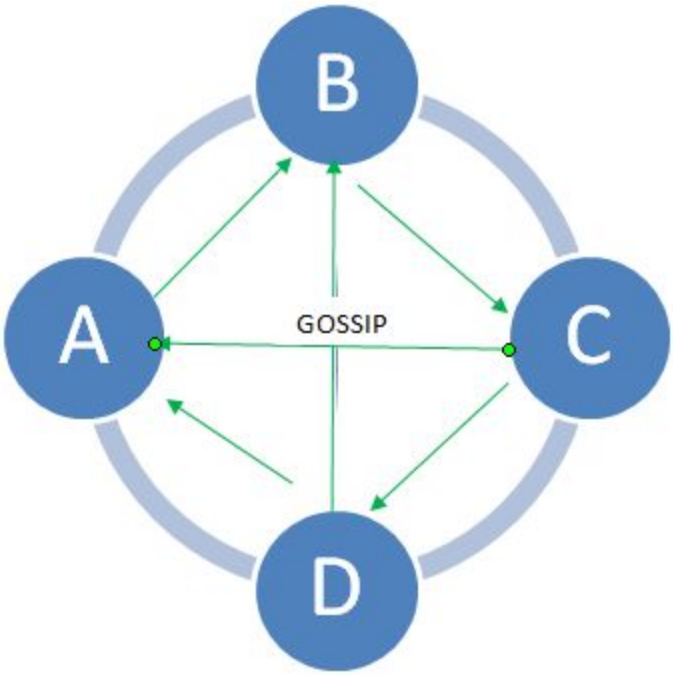
Apache Cassandra is an open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, Tuneably consistent, column-oriented database.

2. Difference between Cassandra And MongoDB ?

- https://scalegrid.io/blog/cassandra-vs-mongodb/
  https://www.upwork.com/hiring/development/mongodb-vs-cassandra/

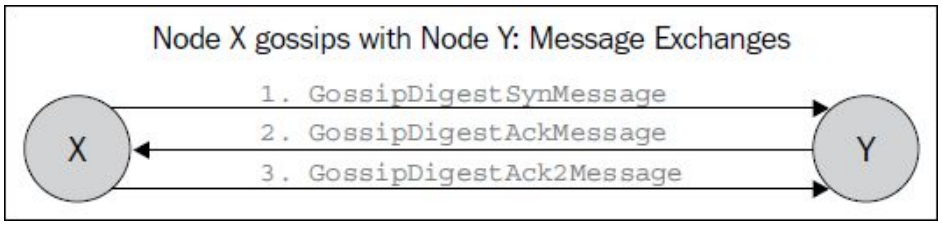
3. What is GOSSIP Protocol in Cassandra ?

- The Gossip protocol is the internal communication technique for nodes in a cluster to talk to each other. Gossip is an efficient, lightweight, reliable, inter-nodal broadcast protocol for diffusing data.
- It's decentralized, "epidemic", fault tolerant and a peer-to-peer communication protocol. Cassandra uses gossiping for peer discovery and metadata propagation.



PEER TO PEER DISTRIBUTION MODEL OF CASSANDRA

- The gossip process runs every second for every node and exchange state messages with up to three other nodes in the cluster. Since the whole process is decentralized, there is nothing or no one that coordinates each node to gossip. Each node independently will always select one to three peers to gossip with. It will always select a live peer (if any) in the cluster, it will probabilistically pick a seed node from the cluster or maybe it will probabilistically select an unavailable node.
-



- The Gossip messaging is very similar to the TCP three-way handshake. With a regular broadcast protocol, there could only have been one message per round, and the data can be allowed to gradually spread through the cluster. But with the gossip protocol, having three messages for each round of gossip adds a degree of anti-entropy. This process allows obtaining "convergence" of data shared between the two interacting nodes much faster.

- SYN: The node initiating the round of gossip sends the SYN message which contains a compendium of the nodes in the cluster. It contains tuples of the IP address of a node in the cluster, the generation and the heartbeat version of the node.

- ACK: The peer after receiving SYN message compares its own metadata information with the one sent by the initiator and produces a diff. ACK contains two kinds of data. One part consists of updated metadata information (AppStates) that the peer has but the initiator doesn't, and the other part consists of digest of nodes the initiator has that the peer doesn't.

- ACK2: The initiator receives the ACK from peer and updates its metadata from the AppStates and sends back ACK2 containing the metadata information the peer has requested for. The peer receives ACK2, updates its metadata and the round of gossip concludes.

An important note here is that this messaging protocol will cause only a constant amount of network traffic.

- Since the broadcasting of the initial digest is limited to three nodes and data convergence occurs through a pretty constant ACK and ACK2, there will not be much of network spike.
- Although, if a node gets UP, all the nodes might want to send data to that peer, causing the Gossip Storm.

4. So how does a new node get the idea of whom to start gossiping with?

- Well, Cassandra has many seed provider implementations that provide a list of seed addresses to the new node and starts gossiping with one of them right away. After its first round of gossip, it will now possess cluster membership information about all the other nodes in the cluster and can then gossip with the rest of them.

- Well, how do we get to know if a node is UP/DOWN?
  The Failure Detector is the only component inside Cassandra(only the primary gossip class can mark a node UP besides) to do so. It is a heartbeat listener and marks down the timestamps and keeps backlogs of intervals at which it receives heartbeat updates from each peer. Based on the reported data, it determines whether a peer is UP/DOWN.

- How does a node being UP/DOWN affect the cluster? The write operations stay unaffected. If a node does not get an acknowledgment for a write to a peer, it simply stores it up as a hint. The nodes will stop sending read requests to a peer in DOWN state and probabilistically gossiping can be tried upon since its an unavailable node, as we have already discussed early on. All repair, stream sessions are closed as well when an unavailable node is involved.

- What if a peer is responding very slowly or timing out? Cassandra has another component called the Dynamic Snitch, which records and analyses latencies of read requests to peer nodes. It ranks latencies of peers in a rolling window and recalculates it every 100ms and resets the scores every 10mins to allow for any other events(Eg: Garbage Collection) delaying the response time of a peer. In this way, the Dynamic Snitch helps you identify the slow nodes and avoid them when indulging in Gossip.

  This article gives you the necessary overview and understanding of how the nodes in clusters in Apache Cassandra behaves via the gossip protocol. For a detailed information about APIs involved and other functional aspects, please visit the links given below in the references.

- https://www.linkedin.com/pulse/gossip-protocol-inside-apache-cassandra-soham-saha

Questions on Infrastructure Setup ?

5. What is the difference between Node, Cluster, And Datacenter ?

- https://stackoverflow.com/questions/28196440/what-are-the-differences-between-a-node-a-cluster-and-a-datacenter-in-a-cassand

6. What is a Rack in Cassandra ?

- The rack configuration allows cassandra to optimize replica placement so you have better fault tolerance properties. If you have all your replicas in rack 1, and that rack goes down, you'll lose the data. If you tell Cassandra about your rack configuration it will keep replicas on different racks. If the replicas happen to be on the same rack, and the rack fail, you will suffer from data loss. Racks are susceptible to failures due to power, heat or network issues.

Questions On ACID, Transaction, Consistency etc ?

7. Define consistency in cassandra ?

8. Transaction ?

- https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlTransactionsDiffer.html

9. How are Cassandra transactions are different from RDBMS transactions?

- https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlTransactionsDiffer.html

10. Questions on NoSql And Acid ?

- https://www.javaworld.com/article/2078841/enterprise-java/datastax-ceo--let-s-clear-the-air-about-nosql-and-acid.html

11. Define Consistency or tune-able Consistency in Cassandra ?

- https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlAboutDataConsistency.html

- https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html#dml_config_consistency_c__dml-config-write-consistency

12. Define Understand the Cassandra data model ?

- Cassandra Data Replication
- http://distributeddatastore.blogspot.in/2015/08/cassandra-replication.html
- https://www.slideshare.net/davegardnerisme/cassandra-concepts-patterns-and-antipatterns?next_slideshow=1

13. How Cassandra knows your Network Topology?

- There is no magic here. Cassandra doesn't learn your topology automatically. You must define the topology such as assigning nodes to racks and data centers and give this information to Cassandra which then uses it. This mapping of nodes to racks and data center is done using Snitch, which I will discuss later.

14. Expiring and Counter columns ?

15. Dealing with tombstones ?

16. Denormalization and duplication of data is a fact of life with Cassandra ?

- https://www.techopedia.com/definition/29168/denormalization

17. Compaction

- To improve read performance as well as to utilize disk space, Cassandra periodically does compaction to create & use new consolidated SSTable files instead of multiple old SSTables.

You can configure two types of compaction to run periodically: SizeTieredCompactionStrategy and LeveledCompactionStrategy.

- SizeTieredCompactionStrategy is designed for write-intensive workloads
- LeveledCompactionStrategy for read-intensive workloads

Periodically Cassandra will run a compaction process on your existing SSTables. Compaction merges multiple SSTables into one new larger SSTable, discarding obsoleted data. After compaction has occurred Cassandra will (eventually) delete the old SSTables.

18. Indexing ?

❏ https://teddyma.gitbooks.io/learncassandra/content/model/indexing.html

19. Write Requests ?

❏ https://teddyma.gitbooks.io/learncassandra/content/client/write_requests.html

20. Can I create a table in Cassandra without having any primary key in it ?

❏ CREATE TABLE example1 (field1 int, field2 int, field3 int);
❏ No, not possible in Cassandra.
❏ InvalidRequest: Error from server: code=2200 [Invalid query] message="No PRIMARY KEY specifed (exactly one required)"

21. Can I insert new field and value dynamically ?

❏ INSERT INTO example (field1, field3 , field4) VALUES (4,6,8);
❏ InvalidRequest: Error from server: code=2200 [Invalid query] message="Undefined column name field4"
❏ NetworkTopologyStrategy requires a snitch to be able to determine rack and datacenter locations of a node. For more information about replication placement strategy, see Data replication.

22. How to define Column Family ?
23. Counter Column ?

❏ https://docs.datastax.com/en/cql/3.1/cql/cql_using/use_counter_t.html

24. How a row key is different than primary key ?

1. Recommendation Vs Primary Key

25. What is Token in Cassandra ?

26. Data placement procedure ?

- https://www.onsip.com/blog/intro-to-cassandra-and-networktopologystrategy

27. What are different types of Column Family ?

Static Column Family

Dynamic Column Family

Dynamic Column family

- A dynamic column family (or column family with wide rows) is one where each internal row may contain completely different sets of cells. The typical example of such a column family is time series. For example, keeping for each user a timeline of all the links on which the user has clicked. Such a column family would be defined in thrift by (definition 3):
- ```
  create column family clicks
   with key_validation_class = UTF8Type
    and comparator = DateType
    and default_validation_class = UTF8Type
  ```
-
- And for a given user, its clicks will be internally stored as:
-
- 

28. Clustering column ?

- In the table definition, a clustering column is a column that is part of the compound primary key definition, but not the first column, which is the position reserved for the
  ```

partition key. Columns are clustered in multiple rows within a single partition. The clustering order is determined by the position of columns in the compound primary key definition.

29. Static Column ?

- ❏ A special column that is shared by all rows of a partition. Cassandra 2.0.6 and later.

30. What is a Partition ?
31. What are different partitioning schemes ?

- ❏ https://10kloc.wordpress.com/2012/12/27/cassandra-chapter-4-data-partitioning/

32. Partition key ?

- ❏ The first column declared in the PRIMARY KEY definition, or in the case of a compound key, multiple columns can declare those columns that form the primary key.
- ❏ https://googleweblight.com/i?u=https://blog.knoldus.com/2016/10/18/cassandra-data-modeling-primary-clustering-partition-compound-keys/&grqid=mVG6ueCs&hl=en-IN
- ❏ https://10kloc.wordpress.com/2012/12/27/cassandra-chapter-4-data-partitioning/

33. Single And Multi Row Partition ?

- ❏ https://pandaforme.gitbooks.io/introduction-to-cassandra/content/understand_the_cassandra_data_model.html

34. How are primary key, partition key, and clustering columns defined?

- ❏ https://pandaforme.gitbooks.io/introduction-to-cassandra/content/understand_and_use_the_ddl_subset_of_cql.html
- ❏ Simple partition key, no clustering columns
- ❏ PRIMARY KEY (partition_key_column)
- ❏ Composite partition key, no clustering columns
- ❏ PRIMARY KEY ((partition_key_col1, ..., partition_key_colN))
- ❏ Simple partition key and clustering columns
- ❏ PRIMARY KEY (partition_key_column,clustering_column1, ..., clustering_columnM)
- ❏ Composite partition key and clustering columns
- ❏ PRIMARY KEY ((partition_key_col1, ..., partition_key_colN), clustering_column1, ...,clustering_columnM))
- ❏

35. Super Column Issues ?

- *It is very strongly recommended that you not use super columns, especially in new design. They have never been problem-free, and now they are deprecated and much more capably replaced by composite keys. Your data could be nicely represented like this in CQL 3, for example:*

```
CREATE TABLE Timestep (
   hardware ascii,
   when timestamp,
   metric1 double,
   metric2 double,
   PRIMARY KEY (hardware, when)
);
```

Or, depending on exactly what you expect to have, it may make more sense to use:

```
CREATE TABLE Timestep (
   hardware ascii,
   metricname ascii,
   when timestamp,
   value double,
   PRIMARY KEY (hardware, metricname, when)
) WITH COMPACT STORAGE;
```

36. Cassandra Data Types?

- CQL provides a rich set of built-in data types, including collection types. Along with these data types, users can also create their own custom data types. The following table provides a list of built-in data types available in CQL.

Data Type	Constants	Description
ascii	strings	Represents ASCII character string
bigint	bigint	Represents 64-bit signed long
blob	blobs	Represents arbitrary bytes
Boolean	booleans	Represents true or false
counter	integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	strings	Represents an IP address, IPv4 or IPv6

int	integers	Represents 32-bit signed int
text	strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4 UUID
varchar	strings	Represents uTF8 encoded string
varint	integers	Represents arbitrary-precision integer

Collection Types

Cassandra Query Language also provides a collection data types. The following table provides a list of Collections available in CQL.

Collection	Description
list	A list is a collection of one or more ordered elements.
map	A map is a collection of key-value pairs.
set	A set is a collection of one or more elements.

User-defined datatypes:

Cqlsh provides users a facility of creating their own data types. Given below are the commands used while dealing with user defined datatypes.

- CREATE TYPE - Creates a user-defined datatype.
- ALTER TYPE - Modifies a user-defined datatype.
- DROP TYPE - Drops a user-defined datatype.
- DESCRIBE TYPE - Describes a user-defined datatype.
- DESCRIBE TYPES - Describes user-defined datatypes.

37. How Cassandra chooses the coordinator node and the replication nodes?

❏ The coordinator node is typically chosen by an algorith which takes "network distance" into account. Any node can act as the coordinator, and at first requests will be sent to

the nodes which your driver knows about. But once it connects and understands the topology of your cluster, it may change to a "closer" coordinator.

- ❏ The coordinator only stores data locally (on a write) if it ends up being one of the nodes responsible for the data's token range.

```
/*
  Cluster.Builder clusterBuilder =
Cluster.builder().addContactPoints("10.xx.xx.245","10.xx.xx.143")
    .withPort(9042).withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
    .withReconnectionPolicy(new ConstantReconnectionPolicy(100L)
    .withPoolingOptions(poolingOptions)
    .withLoadBalancingPolicy(new TokenAwarePolicy(new DCAwareRoundRobinPolicy()));

    Cluster cluster = clusterBuilder.build();
    Session session=cluster.connect("tp");

    */
```

38. Load Balancing Strategy ?

- ❏ http://docs.datastax.com/en/drivers/java/2.1/com/datastax/driver/core/policies/DCAwareRoundRobinPolicy.html

39. What Are seed Nodes ?

- ❏ Seed nodes serve two purposes in cassandra:

1. They allow the instance to find the cluster on the very first startup, and
2. They assist in gossip convergence.

The first should be pretty self explanatory - when you add a new node to a cluster, it needs to find the cluster; this is the most easily understood purpose of seeds.
The second is slightly less obvious. Each instance will gossip with other nodes, in three distinct steps:

- Gossip with a random live node
- Gossip with a random dead node
- Maybe gossip with a seed

- ❏ Gossiping with a live node allows each instance to ensure that updates to state (new nodes, etc) are propagated.

Gossiping with a dead node allows each instance to detect when dead nodes come back to life (we expect gossip to fail if it's still dead, but if it succeeds, we can mark the node UP).

Gossiping with a seed is probabilistic, and the primary purpose is to assist in convergence: by having all of the nodes in the cluster gossip regularly with the same set of seeds, we can guarantee schema/state changes propagate regularly.
It's not strictly required that all instances have the same seed list, and it's not required that all of your seed lists remain exactly the same over time, but it does help with convergence. If your

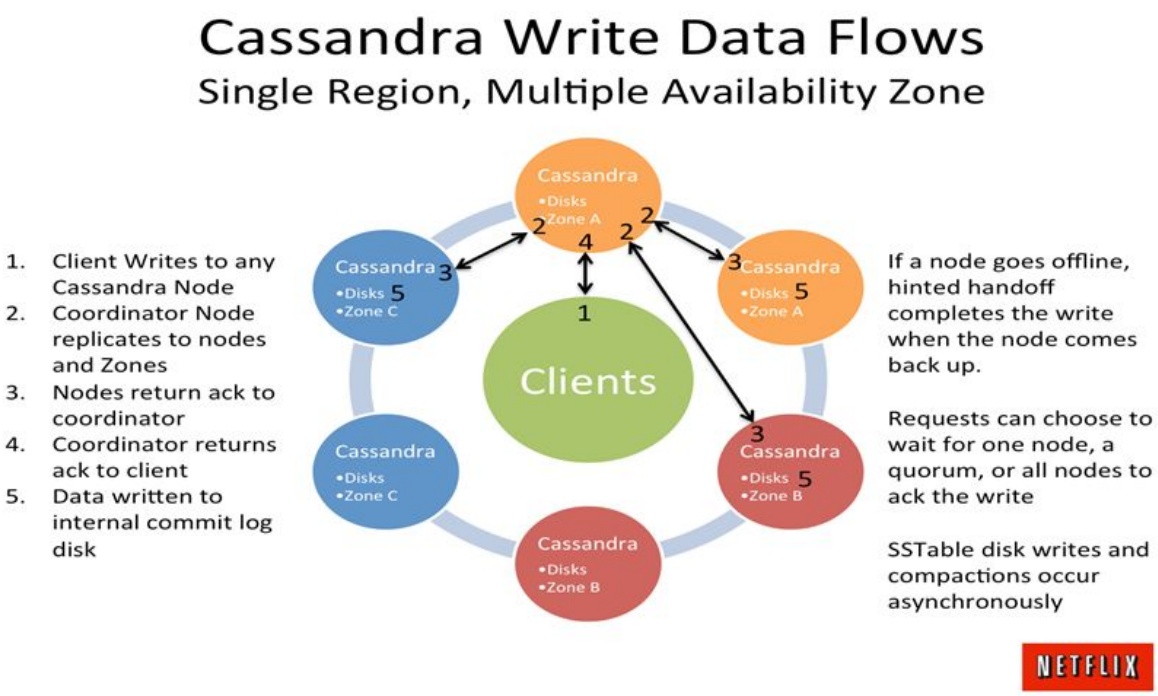
seeds die and get replaced, you should update the configuration for the other nodes, but you do not typically need to rush to restart them for the change to take effect.

40. Difference Between Seeds and Contact Point ?

Description of addContactPoint():

- ❑ Seeds are used during startup to discover the cluster.

41. Cluster



Term	Definition
Keyspace	A logical grouping of tables that all have the same replication factor and replication strategy.
Table	A CQL table is a logical grouping of partitions that share the same schema.
Partition	A partition is a collection of sorted CQL rows. A partition is the unit of data access as most queries access a single partition.
Row	A CQL row is an abstract data structure that is a collection of sorted columns. When a partition has multiple rows, the rows are physically sorted on disk per the clustering column(s).
Column name	A column name is how each column is identified. A column name may be used in the primary key as either the partition key or as a clustering column.
Column value	The value stored in a column.
Primary key	The CQL primary key is a composite key that defines the partition key and, optionally, one or more clustering columns. The partition key itself may be defined as a single key or composite key.
Partition key	The partition key is the first component of the primary key and must be unique within a CQL table. The partition key may be defined as a composite key if it is surrounded by parentheses and supplied with a comma separated list of values.
Clustering column	A clustering column is used to allow a partition to have multiple rows where each row is sorted per the clustering column(s).

Physical

http://exponential.io/blog/2015/01/08/cassandra-terminology/

PPTs

https://www.slideshare.net/davegardnerisme/cassandra-concepts-patterns-and-antipatterns?next_slideshow=1

Physical

The physical layer is how Cassandra actually stores data on disk. Understanding the physical layer is an important part of performance turning and data modeling in Cassandra.

Term	Definition
Partition	A partition is a physical unit of data that consists of a collection of sorted cells and is identified by a partition key.
Partition key	A hashed value that provides fast access to a partition and that must be unique within a table.
Cell	A cell value is smallest unit of storage. At a minimum, a cell consists of a cell name, a cell value and a timestamp. The cell value may be empty.
Cell name	A cell name is either: ● a single CQL column name when no clustering columns are defined; or ● a composite of multiple CQL column values when all column names are used in the primary key; or ● a composite of multiple CQL column values plus a single CQL column name when clustering columns are defined and when some column names are not used in the primary key.
Cell value	The cell value can contain a value or be left empty.
Timestamp	The timestamp is a cell property that is used internally by Cassandra to keep track of when a cell was inserted or updated. It is also used by Cassandra's merge process during a read request.
TTL	TTL, or time to live, is a cell property that you can set to define the date/time after which a cell will be automatically deleted by Cassandra.
Tombstone	The tombstone is cell property that represents a deletion marker. Cassandra will remove cells that contain a tombstone from the partition that is returned to a client during a read.

Collection
https://www.guru99.com/cassandra-collections-tutorial-set-list-map.html

42. Query Builder Example ?

```
private static void query(Session session) {
    System.out.println("query begin");
    showMemory();
    long beginTime = System.currentTimeMillis();

    Statement stmt = QueryBuilder
            .select()
            .all()
            .from("xxx")
            .where(eq("id", "0"))
            .orderBy(desc("sid"))
            // .limit(10)
            .setFetchSize(100);
    ResultSet rs = session.execute(stmt);
    int i = 0;
    while (!rs.isExhausted()) {
        if (i < 5000) {
            rs.one(); // skip first 5000
            i++;
            continue;
        }
        Row row = rs.one();
        System.out.println(i + " : " + row);

        if (i > 5010) {
            break;
        }
        i++;
    }

    long endTime = System.currentTimeMillis();
    double timeConsume = (endTime - beginTime) / 1000.0;
    showMemory();
    System.out.println("query end : " + timeConsume + "s");
}
```

43. Explain UUID, TIMEUUID, TIMESTAMP,COUNTER ?

44. What is a secondary index?

45. Questions on CRUD ?

46. How is data updated in cassandra ?

- ❏ During a write, Cassandra adds each new row to the database without checking on whether a duplicate record exists. This policy makes it possible that many versions of the same row may exist in the database.
- ❏ Periodically, the rows stored in memory are streamed to disk into structures called SSTables. At certain intervals, Cassandra compacts smaller SSTables into larger SSTables. If Cassandra encounters two or more versions of the same row during this process, Cassandra only writes the most recent version to the new SSTable. After compaction, Cassandra drops the original SSTables, deleting the outdated rows.

47. What is Bloom filter (Filter.db) ?

- ❏ A structure stored in memory that checks if row data exists in the memtable before accessing SSTables on disk.

48. Difference between Sharding and Partitioning ?

- ❏ Partitioning is a general term used to describe the act of breaking up your logical data elements into multiple entities for the purpose of performance, availability, or maintainability.

Sharding is the equivalent of "horizontal partitioning". When you shard a database, you create replica's of the schema, and then divide what data is stored in each shard based on a shard key. For example, I might shard my customer database using CustomerId as a shard key - I'd store ranges 0-10000 in one shard and 10001-20000 in a different shard. When choosing a shard key, the DBA will typically look at data-access patterns and space issues to ensure that they are distributing load and space across shards evenly.

"Vertical partitioning" is the act of splitting up the data stored in one entity into multiple entities - again for space and performance reasons. For example, a customer might only have one billing address, yet I might choose to put the billing address information into a separate table with a CustomerId reference so that I have the flexibility to move that information into a separate database, or different security context, etc.

To summarize - partitioning is a generic term that just means dividing your logical entities into different physical entities for performance, availability, or some other purpose. "Horizontal

partitioning", or sharding, is replicating the schema, and then dividing the data based on a shard key. "Vertical partitioning" involves dividing up the schema (and the data goes along for the ride).

Final note: **you can combine both horizontal and vertical partitioning techniques - sometimes required in big data, high traffic environments**.

Reference links

- Main Site
  - http://cassandra.apache.org/
  - www.planetcassandra.org
- Community Forums:
  To register, drop a mail to - user-subscribe@cassandra.apache.org
  Datastax also provied a forum
  http://www.datastax.com/support-forums/
  http://stackoverflow.com/questions/tagged/cassandra?id=cassandra
- Download
  - Cassandra: http://cassandra.apache.org/download/
  - Third Party Distributions
    http://planetcassandra.org/cassandra/ - DataStax Community Edition free
    DataStax Enterprise Edition paid
    Compare - http://www.datastax.com/download/dse-vs-dsc
    Cloud Deployment Guides - http://planetcassandra.org/cassandra/
- FaQ
  - http://www.datastax.com/resources/faq
- Documentation for Cassandra
  - http://wiki.apache.org/cassandra/GettingStarted
- Wiki
  - http://wiki.apache.org/cassandra/
- Documentation for Datastax
  - http://docs.datastax.com/en/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html

CQLSH

https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlSelect.html

Expand on;
Select token(A) from exp;

1. Login using cqlsh:
    - *cqlsh 127.0.0.1 9042 -u cassandra -p cassandra*
    - *cqlsh 127.0.0.1 9042*
2. List all users
    a. *list users;*
3. Show;
    a. *List all the Keyspaces: Describe keyspaces;*
    b. *Use <Keyspace_name>;*
    c. *Describe <Table_name>;*
4. *SELECT cluster_name, listen_address FROM system.local;*
5. *select release_version from system.local;*
6. *CREATE KEYSPACE [name] WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};*

JSON
*select json * from example;*

Filtering

1. *select * from example where field2 = 2 allow filtering; (When field2 is not a PK)*
2. *SELECT * FROM ruling_stewards WHERE king = 'Brego' AND reign_start >= 2450 AND reign_start < 2500 ALLOW FILTERING;*
3. *CREATE INDEX ON playlists(artist); - To create secondary index, as Filed1 which is a PK is already a primary index*
4. *select json * from example;*

SSH - Secure Shell Commands

Documented Shell Commands

Given below are the Cqlsh documented shell commands. These are the commands used to perform tasks such as displaying help topics, exit from cqlsh, describe,etc.

- **HELP** - Displays help topics for all cqlsh commands.

- **CAPTURE** - Captures the output of a command and adds it to a file.
- **CONSISTENCY** - Shows the current consistency level, or sets a new consistency level.
- **COPY** - Copies data to and from Cassandra.
- **DESCRIBE** - Describes the current cluster of Cassandra and its objects.
- **EXPAND** - Expands the output of a query vertically.
- **EXIT** - Using this command, you can terminate cqlsh.
- **PAGING** - Enables or disables query paging.
- **SHOW** - Displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.
- **SOURCE** - Executes a file that contains CQL statements.
- **TRACING** - Enables or disables request tracing.

CQL Data Definition Commands

- **CREATE KEYSPACE** - Creates a KeySpace in Cassandra.
- **USE** - Connects to a created KeySpace.
- **ALTER KEYSPACE** - Changes the properties of a KeySpace.
- **DROP KEYSPACE** - Removes a KeySpace
- **CREATE TABLE** - Creates a table in a KeySpace.
- **ALTER TABLE** - Modifies the column properties of a table.
- **DROP TABLE** - Removes a table.
- **TRUNCATE** - Removes all the data from a table.
- **CREATE INDEX** - Defines a new index on a single column of a table.
- **DROP INDEX** - Deletes a named index.

CQL Data Manipulation Commands

- **INSERT** - Adds columns for a row in a table.
- **UPDATE** - Updates a column of a row.
- **DELETE** - Deletes data from a table.
- **BATCH** - Executes multiple DML statements at once.

CQL Clauses

- **SELECT** - This clause reads data from a table
- **WHERE** - The where clause is used along with select to read a specific data.
- **ORDERBY** - The orderby clause is used along with select to read a specific data in a specific order.

PPT

- New Problems which can't be handled by traditional RDBMS
- Tradeoff between Consistency, Availability, Partition Tolerance (CAP theorem)
- What are the different solutions available?
- What is Cassandra?
- Use-Cases for Cassandra
- Cassandra Features – Tunable Consistency, P2P Architecture, Elastic Scalability ,Col Orientation
- Demo Application using Cassandra
- Questions??

- Understand what database model is
- Understand the analogy between the RDBMS and Cassandra Data Model
- Understand the following Cassandra database elements:
  - ✓Cluster
  - ✓Keyspaces
  - ✓Column Families
  - ✓Columns
  - ✓Super Columns
  - ✓Rows
  - ✓Indexes in Cassandra
  - ✓Primary and Composite Keys and their limitations
  - ✓Design Differences between RDBMS and Cassandra
  - ✓Materialized Views
  - ✓Valueless Columns
  - ✓Aggregate Keys

- Learn about the System Keyspaces
- Learn about internode communication such as Peer to Peer structure as well as Gossip Protocols
- Learn how Cassandra detects the failures in the nodes and repairs it
- Learn about Anti Entropy and Read Repair
- Learn about the Memtables, Sstables, and Commit logs
- Hinted Handoffs
- Compaction
- Bloom Filters
- Tombstones
- SEDA
- Manager and Services

- Identify challenges faced by RDBMS
- Identify various possible available solutions
- Indentify the rational behind choosing Cassandra
- Understand how data modelling differs in Cassandra from traditional relational databases
- Understand how queries are used to design Cassandra data model
- Apply Cassandra data modelling to various use cases
- Create the application which would involve creating various data elements you learned about in Module 2
- Perform batch updates and search column families
- Overview of the whole project specifying how Cassandra solved the problem which was laid out in the beginning

- Learn about various options of configuring Keyspaces and Column Families
- Learn about various Cassandra Replacement Strategies
- Learn about Replication
- Learn about Partitioners
- Learn about Snitches
- Learn about configuring Cluster
- Learn about Security
- Learn about Monitoring Cassandra Cluster
- Learn about Cassandra Maintenance
  - ✓Getting Ring information
  - ✓Basic Maintenance
  - ✓Snapshots
  - ✓Load Balancing
  - ✓Decommissioning and Updating nodes
- Learn about Performance Tuning
  - ✓Data storage, Reply timeouts
  - ✓Commit Logs, MemTables, Caching and Buffer sizes

Integrating Cassandra with Hadoop

- Learn what Hadoop is
- Learn Hadoop Disribution File System
- Learn how to work with Map Reduce
- Learn Tools like PIG and HIVE
- Learn PIG and HIVE interaction with Cassandra

CRUD Operations in Cassandra

- Learn about Reading and writing data in Cassandra
- Learn about Cassandra API (Thrift)
- Learn about Slice Predicates
- Learn Data Definition Language (DDL) in Cassandra
- Learn Data Manipulation Language (DML) statements within Cassandra
- Learn to execute CQL scripts from with in CQL and from Command prompt
- Learn to Create and Modify Users
- Learn about Batch Mutates and Batch Deletes
- Learn various Security configurations in Cassandra
- Learn to Capture CQL outputs to a file
- Learn to Import and Export data with CQL

Cassandra Use Case – Summary

✓eCommerce (Travel Portal)

- ✓ Both B2B & B2C Consumers
- ✓ High volume of shopping transactions (> 500 Million Visits / Day)
- ✓ High volume supply changes (Manual & System) generated.
- ✓ Huge Inventory Database (Millions of hotels)
- ✓ High Read/Write (Thousands Reads & Writes/Second)
- ✓ Application has to 99.99% Available
- ✓ Fault Tolerant & Reliable.
- ✓ Fast & Quick Shopping Experience.
- ✓ Elastic Scale
- ✓ Innovative Recommendations & Algorithms.
- ✓ Should be fast for new changes
- ✓ Should be cost effective for maintenance.

✓Development Approaches

- ✓ Legacy Way (Pure RDBMS)
- ✓ Augmented (RDBMS + Caching, Heavy Database Hardware)
- ✓ Using Cassandra