# Cassandra Installation

## Prerequisites:

 java version "1.8.0\_121"

## Create the directory cassandra in /opt

cd /opt

mkdir cassandra

cd cassandra

## Download the Apache-Cassandra-3.9

wget http://www-eu.apache.org/dist/cassandra/3.9/apache-cassandra-3.9-bin.tar.gz

## Untar the apache-cassandra-3.9-bin.tar.gz

tar -zxvf apache-cassandra-3.9-bin.tar.gz

## Give the read and write permission to the installation folder

chmod 777 /opt/cassandra/apache-cassandra-3.9

## Set the Environment variables in bashrc

vim ~/.bashrc

## Add these lines at the end

#Cassandra Environment Variables Starts

export CASSANDRA\_HOME=/opt/cassandra/apache-cassandra-3.9

export PATH=$CASSANDRA\_HOME/bin:$PATH

#Cassandra Environment Variables ends

## Refresh the bashrc

source ~/.bashrc

## Create the specified directories 'data', 'commitlog', 'saved\_caches' in /opt/cassandra/

cd /opt/cassandra

mkdir data

mkdir commitlog

mkdir saved\_caches

## Give the read and write permission to all the folders

chmod 777 /opt/cassandra/data/

chmod 777 /opt/cassandra/saved\_caches/

chmod 777 /opt/cassandra/commitlog/

## Configuring the cassandra.yaml

cd $CASSANDRA\_HOME/conf/

## Open the cassandra.yaml

vim cassandra.yaml

## Start editing the file

### search for 'data\_file\_directories'

/data\_file\_directories

### Edit it, just change the path of the directories, it should look like same as below

data\_file\_directories:

  - /opt/cassandra/data

### Search for 'commitlog\_directory'

/commitlog\_directory

### Edit it, just change the path of the directories, it should look like same as below

commitlog\_directory: /opt/cassandra/commitlog

### Search for 'saved\_caches\_directory'

/saved\_caches\_directory

### Add these lines below '# specified in this configuration file.'

saved\_caches\_directory: /opt/cassandra/saved\_caches

### Search for seeds

/seeds

Change the 127.0.0.1 to your local ip

- seeds: "172.17.0.2"                             //change the ip w.r.t to your ip

### Search for listen\_address

/listen\_address

### Change localhost to the ip address

listen\_address: 172.17.0.2                       //change the ip w.r.t to your ip

### Search the rpc address

/rpc\_address

### Change the local host to ip address

rpc\_address: 172.17.0.2                         //change the ip w.r.t to your ip

## Start the cqlsh

cqlsh 172.17.0.2                               //change the ip w.r.t to your ip

## Output should be

Connected to Test Cluster at 172.17.0.2:9042.

[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]

Use HELP for help.

cqlsh>

## Testing the CQLSH

## Create the keyspace with name test

CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy', 'replication\_factor' : 1};

## open the keyspace

USE test;

## Create the table

CREATE TABLE test1(emp\_id int PRIMARY KEY, emp\_name text, emp\_city text);

## Insert values

INSERT INTO test1 (emp\_id, emp\_city, emp\_name) VALUES ( 1, 'Rohan', 'Mumbai');

## Display the table

SELECT \* FROM test1;

## Output Should be

 emp\_id | dmp\_name | emp\_city

--------+-----------+----------

  1 | Mumbai | Rohan

# Creating the Data Centers /Adding the New Node to Data Center

## Configuration

Follow the installation of Cassandra as above instruction for  the new data center

To make the different data center in the same cluster. Edit the configuration file cassandra-rackdc.properties in both new and old data center.

### vim cassandra-rackdc.properties

Change the values below

dc=dc1 //Data Center Name

rack=rack1 //Data Center rack name

* Create at-least one seed in each data center.
* Then add the seeds of other datacenter into the seed nodes of current datacenter.
* And do the same with the old data center. No need to add the seed of the other data node to all the nodes of new data center.
* Once the configuration is set start the cassandra in new data center. Please start the seed node first and then other nodes.
* Verify the cluster by using **nodetool status**. After that run **nodetool cleanup** on each node of the old datacenter/datacenters wait till one node to finish and then run on the next node.

## Adding the new Node to the Datacenter/Cluster

* Follow the installation steps as mentioned above.
* Once the installation is done don’t start the node, if started stop the node first.
* Edit the configuration file cassandra.yaml add the seed.
* cassandra-rackdc.properties   add the data center and rack to which data center you are adding.
* Once the configuration is completed start the node.
* Verify using the **nodetool status**.
* After that verification run **nodetool cleanup** on each old nodes wait till it finish one node, this will clean up the keys which no longer belongs to it.

## Adding the Data Center of different cluster

Change the Cluster Name  
Start the cqlsh on the node, then run the below query

UPDATE system.local SET cluster\_name = '<new\_cluster\_name>' where key='local';  
new\_cluster\_name = this should be the cluster name to which this node/datacenter you are going to add  
Exit the cqlsh and run nodetool flush system. Stop the Cassandra.

### Configuration

cassandra.yaml  
*cluster\_name* =add the cluster name in which cluster you are adding.   
*seeds* = add the seed node of other datacenters. And the seeds of the same datacenter and the seed of new datacenter to the existing datacenter of the cluster, where this node is added

cassandra-rackdc.properties   
dc=dc1 //Data Center Name  
rack=rack //Data Center rack name  
Save both the files.

Start the Cassandra  
./cassandra -Dcassandra.ignore\_dc=true

* Verify using the **nodetool status**.
* After that verification run **nodetool cleanup** on each old nodes wait till it finish one node, this will clean up the keys which no longer belongs to it.

# Removing the node from the cluster

## Case 1: nodetool decommission

If node is up and running use the above command, before that check the status of the node using the **nodetool status**Monitor **nodetool netstats** to monitor the progress

## Case 2: nodetool removenode

If node is not running

## Case 3: nodetool assassinate

If node doesn’t stop streaming of data to other nodes

# Taking the Backup

# Status of the nodes

***nodetool status***

## Finding the size of the keyspace and column family in cassandra

### Find the Size of the KeySpace(Database)

***nodetool cfstats -H <keyspace\_name>***

### Find the size of the ColumnFamily(Table)

***nodetool cfstats - H <keyspace\_name.columnfamily>***

## Taking the cassandra schema backup

***cqlsh -p <password> -u <username> <host\_ip> -e 'DESCRIBE KEYSPACE <keyspace\_name>' > ~/filename.cql***

## Taking the data backup

***nodetool -h <host\_ip> snapshot keyspace\_name***

**Reuslt wil be stored in**

**data\_file\_directories/keyspace\_name/table\_name-UUID/snapshots/snapshot\_name**

## Removing the snapshot

When snapshot/backup is not necessary you remove using

***nodetool -h <host\_ip> -p <port\_no> clearsnapshot***

## Restoring the data from the snapshot

?

# NodeTool

## Cassandra Version

***nodetool version***

Gives the version of Cassandra

## NodeTool Monitoring and Status Check Commands

***nodetool status***

Give the status of the node in the datacenter U --> Up N --> Normal D --> Down

***nodetool describecluster***

prints the basic information of the cluster including the name, snitch and partitioner

***nodetool ring keyspace***

Displays node status and information about the ring as determined by the node being queried. This can give you an idea of the load balance and if any nodes are down. If your cluster is not properly configured, different nodes may show a different ring; this is a good way to check that every node views the ring the same way.

***nodetool info***

Outputs node information including the token, load info (on disk storage), generation number (times started), uptime in seconds, and heap memory usage. of that particular node

***nodetool gossipinfo***

Shows the gossip information for the cluster. Most useful to check the datasync

***nodetool describering keyspace***

Shows the token ranges for a given keyspace.

***nodetool statusgossip***

Gives the current status of the internode gossip

***nodetool statusthrift***

Gives the current status of the thrift server

## Node Repair, rebuild, flush Commands

***nodetool rebuild***

Rebuilds data by streaming from other nodes (similar to bootstrap). Use this command to bring up a new data center in an existing cluster.

***nodetool repair***

Begins an anti-entropy node repair operation. If the -pr option is specified, only the first range returned by the partitioner for a node is repaired. This allows you to repair each node in the cluster in succession without duplicating work. Without -pr, all replica ranges that the node is responsible for are repaired.

Optionally takes a list of column family names.

##### Best Practice for Nodetool Repair

###### Repair Frequeny

If you are using the read/write consistency level that dont guarantee immediate consistency you will want to do more frequent repair

###### Repair Scheduling

Minimise the impact of repair by scheduling them off peak hours.

###### Operation Requring the Repair

When snitch and replication of the keyspace is altered

***nodetool removetoken status | force | token***

Shows status of a current token removal, forces the the completion of a pending removal, or removes a specified token. This token’s range is assumed by another node and the data is streamed there from the remaining live replicas.

**nodetool repair keyspace column family**

Begins an anti-entropy node repair operation. If the -pr option is specified, only the first range returned by the partitioner for a node is repaired. This allows you to repair each node in the cluster in succession without duplicating work.

***nodetool flush***

Flushes all memtables for a keyspace to disk, allowing the commit log to be cleared. Optionally takes a list of column family names.

***nodetool drain***

Flushes all memtables for a node and causes the node to stop accepting write operations. Read operations will continue to work. You typically use this command before upgrading a node to a new version of Cassandra or routinely before stopping a node to speed up the restart process. Because this operation writes the current memtables to disk, Cassandra does not need to read through the commit log when you restart the node. If you have durable writes set to false, which is unlikely, there is no commit log and you must drain the node before stopping it to prevent losing data.

***nodetool cleanup keyspace columnfamily***

Triggers the immediate cleanup of keys no longer belonging to this node. This has roughly the same effect on a node that a major compaction does in terms of a temporary increase in disk space usage and an increase in disk I/O. Optionally takes a list of column family names

***nodetool rebuild\_index***

Cassandra repair mechanism aren’t helpful for keeping the secondary indexes up to date, Because secondary indexes cannot be repaired and there is simple way to check their validity. Rebuild\_index is used to rebuild the indexes from the scratch.

## NodeTool Stats Commands

***nodetool cfhistograms all\_trade.local\_service\_requests***

Give the read/write latency of the Column Family, include row size, column count useful for monitoring the column family

***nodetool cfstats -H all\_trade.testsnapshot***

Gives the stats of the Keyspace and Column Family like size

***nodetool netstats***

Displays network information such as the status of data streaming operations (bootstrap, repair, move, and decommission) as well as the number of active, pending, and completed commands and responses.

***nodetool tpstats***

To find statics on the thread pools.Top portions indicates how many operations are in what stage. Bottom portion indicates number of dropped messages, when more requests come the node defend itself.

***nodetool tablestats***

Gives the information of the read and write latency and info at Keyspace and table level

## Enabling and Disabling the Services

***nodetool disablegossip***

Disable Gossip. Effectively marks the node dead.

***nodetool disablethrift***

Disables the Thrift server.

***nodetool enablegossip***

Re-enables Gossip.

***nodetool enablethrift***

Re-enables the Thrift server.

## Log Commands

***nodetool getlogginglevels***

Gives the log levels

***nodetool setlogginglevel***

Set the log level threshold for a given class. If both class and level are empty/null, it will reset to the initial configuration

## Databackup and Deleting

***nodetool -h <host\_ip> snapshot keyspace\_name***

Reuslt wil be stored in

data\_file\_directories/keyspace\_name/table\_name-UUID/snapshots/snapshot\_name

***nodetool clearsnapshot keyspace snapshot-name-uuid***

This will clear the snapshot (databackup)