While Working With Cassandra

A cassandra user guide for administration

By sandip gatkal.

Overview

This guide is helpful if you decided to use cassandra as database for your project. How to configure cassandra cluster? Which is the most suitable cassandra version? Is your project use case really need to use cassandra? Why Cassandra not MongoDb? There will be many problems you will face when you will work on cassandra like, How to run nodetool repair? What will happen if node is decommissioned? What if node is removed without decommissioning it? Will it be helpful to add indexing on particular column? And much more important concepts will be addressed in this document.

What is NoSQL databases?

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL mean "not only SQL". As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

What is cassandra?

Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It is a type of database. Following are the features of Cassandra:

- High scalability: Cassandra allows to add more hardware as needed, which accommodates more customer data. Cassandra is linearly scalable, increase in number of nodes increases performance proportionally.

- No single point of failure: As cassandra has peer to peer architecture. If replication factor is properly set then cassandra is continuously available for business critical applications.
- Flexible data storage: Cassandra stores all possible data formats, including structured, semi-structured and unstructured. It dynamically accommodate changes to data structure according to need.
- Flexibility of data distribution: Cassandra provides flexibility to distribute data according to need by replicating data across multiple data centers.

Why Cassandra

The Apache Cassandra database is the right choice when you need 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.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.

Difference between MongoDB and Cassandra

These both databases are nosql databases.Both databases perform well on reads where the hot data set fits in memory. Both also emphasize join-less data models (and encourage denormalization instead), and both provide indexes on documents or rows, although MongoDB's indexes are currently more flexible.

Following are the key differences that need to be consider if you wanna know which database best fits your use case.

MongoDB	Cassandra
Writes are more problematic in MongoDB, partly because of the b-tree based storage engine, but more because of the per database write lock	Cassandra's storage engine provides constant-time writes, no matter how big your data set grows
If you're looking at a single server, MongoDB	No single point of failure architecture of

is probably a better fit.	cassandra will be easier to set up and more reliable for scaling.Cassandra also gives a lot more control over how your replication works, including support for multiple data centers.
Master slave architecture.If master goes down,system collapses.	cassandra server setup is simple as no special role is needed for nodes.All nodes are peer.
If you're presently using JSON blobs, MongoDB is an insanely good match for your use case,	Not recommended for json document.

How to configure cassandra cluster

Depending upon the size of the data and replication factor, decide number of nodes (server) you are going to use in your cluster. Start those number of nodes and open command line on respective nodes. Make sure to configure conf/cassandra.yaml on each node, it's recommended but not required to be same on every node.
Following configurations are to be made to conf/cassandra.yaml
1. **cluster_name** : this is the name of your cluster, same on each node
2. **seeds** : This is a comma-delimited list of the IP address of each node in the cluster. Significance of configuring seed nodes : In conf/cassandra.yaml configuration file we need to provide seed node configuration when we configure cassandra cluster. The seed node designation has no purpose other than bootstrapping the gossip process for new nodes joining the cluster. Making every node a seed node is not recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).
Note : To prevent problems in gossip communications, use the same list of seed nodes for all nodes in a cluster.
3. **listen_address**: This is IP address that other nodes in the cluster will use to connect to this one. It defaults to localhost and needs to be changed to the private IP address of the node.
4. **rpc_address**: This is the IP address for remote procedure calls. It defaults to localhost. If the server's hostname is properly configured, leave this as is. Otherwise, change to server's IP address or the loopback address (127.0.0.1).
5. **endpoint_snitch**: Name of the snitch, a snitch determines which data centers and racks are written to and read from. Snitches inform Cassandra about the network topology so that requests are routed efficiently and allows Cassandra to distribute replicas by grouping machines into data

centers and racks. All nodes must have exactly the same snitch configuration. This defaults to SimpleSnitch, which is used for networks in one datacenter. Configuring for now we'll change it to GossipingPropertyFileSnitch, which is preferred for production setups.
6. **auto_bootstrap**: (Default: true) This setting has been removed from default configuration. It makes new (non-seed) nodes automatically migrate the right data to themselves. When initializing a fresh cluster without data. So need to add
auto_bootstrap: false
Now restart nodes
Note: If additional node is to be added to the cluster, configure it as mentioned above and start the node.

Grace Period

When we perform delete in cassandra,Cassandra treats a delete as an insert.The data being added to the partition in the DELETE command is a deletion marker called a tombstone. The tombstones go through Cassandra's write path, and are written to SSTables on one or more nodes. The key difference feature of a tombstone: it has a built-in expiration date/time. At the end of its expiration period called as grace period the tombstone is deleted as part of Cassandra's normal compaction process.
In a multi-node cluster, Cassandra can store replicas of the same data on two or more nodes. This helps prevent data loss, but it complicates the delete process. If a node receives a delete for data it stores locally, the node tombstones the specified record and tries to pass the tombstone to other nodes containing replicas of that record. But if one replica node is unresponsive at that time, it does not receive the tombstone immediately, so it still contains the pre-delete version of the record. If the tombstoned record has already been deleted from the rest of the cluster before that node recovers, Cassandra treats the record on the recovered node as new data, and propagates it to the rest of the cluster. This kind of deleted but persistent record is called a zombie.To prevent the reappearance of zombies, Cassandra gives each tombstone a grace period. The purpose of the grace period is to give unresponsive nodes time to recover and process tombstones normally. If a client writes a new update to the tombstoned record during the grace period, Cassandra overwrites the tombstone. If a client sends a read for that record during the grace period, Cassandra disregards the tombstone and retrieves the record from other replicas if possible.
Default Grace Period is 10 days
It can be changed per column by following command in cqlsh :
Example : consider that we are changing grace period to 1 day.

```
cqlsh:dms> USE  keyspace_name;
cqlsh:dms> ALTER TABLE table_name WITH gc_grace_seconds = 86400;
```

Procedure of restarting Cassandra node without outage

Prerequisites : replication_factor should be of atleast 2.
Cassandra node can be restarted by executing following commands :
1. nodetool -h localhost disablegossip
- Disables the gossip protocol.
- This command effectively marks the node as being down

2. nodetool -h localhost disablethrift
- Disables the Thrift server.

3. nodetool -h localhost drain
- Cassandra stops listening for connections from the client and other nodes.

Now cassandra service running on the node need to be stop.
After that again start the cassandra process on the node.

Nodetool repair

Node repair performs following tasks:
- Ensures that all data on a replica is consistent.
- Repairs inconsistencies on a node that has been down.

Procedure :
Run command**:** ***nodetool repair***
- Use the -hosts option before repair to list the good nodes to use for repairing the bad nodes.
- Use -h to name the bad nodes**.**
- Use the -inc option for an incremental repair.
- Use the -par option for a parallel repair

Replication Factor

Replication factor describes how many copies of your each row exist on different nodes in entire cluster.Replication factor is set while creating keyspace.
cqlsh> CREATE KEYSPACE "KeySpace Name"
WITH replication = {'class': 'Strategy name', 'replication_factor' : 'No.Of   replicas'};
Replication factor can be change anytime if needed by following following process

Procedure of changing replication factor of cassandra node

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance.Replication factor of 1 means that there is only one copy of each row on one node. Replication factor of 2 means two copies of each row, where each copy is on a different node.
Now we are going to change replication factor of keyspace my_keyspace from 1 to 2.
Precaution :
If security features are used, Increase replication_factor in system_auth. As if the node with keyspace system_auth goes down.We will not be able to login from other nodes.

Procedure :
Step 1: Alter the replication strategy parameters :
cqlsh> ALTER KEYSPACE keyspace_name WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2 };

Step 2 : On each affected node, run the nodetool repair command.

Step 3: Wait until repair completes on a node, then move to the next node.
- after nodetool repair gets complete on all affected nodes. Replication factor takes effect.

Consider following example :
Consider cluster of 3 nodes A,B,C with replication factor of each keyspace on each node is 1.

Step 1
Change replication factor of one of the keyspace my_keyspace on a node to 2.
cqlsh> ALTER KEYSPACE my_keyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2 };
Observations after executing alter keyspace command :
Cluster behaviour is same as that of before running alter command. Means replication factor has no effect on number of replicas yet.

Step 2
On each affected node, run the nodetool repair command.
user@node1>nodetool repair
Observation: I was getting less number of records than the total records.
Step 3
As nodetool repair on node A is completed, so applying nodetool repair on other remaining nodes
Run nodetool repair on node B
user@node2>nodetool repair

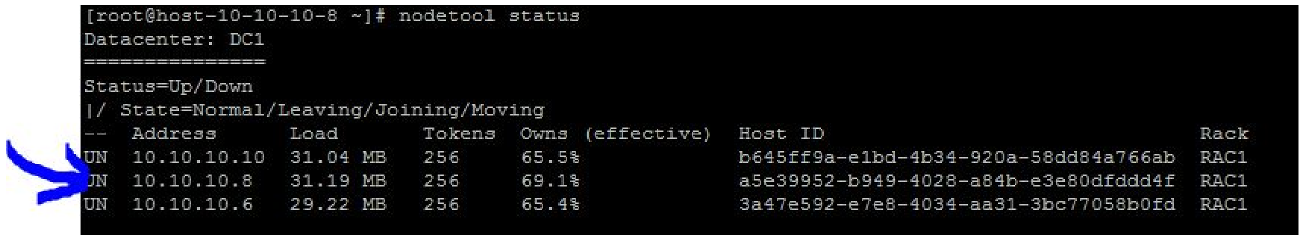
Observation : I was getting more number of records than the first one but less than the total records.
Now need to run nodetool repair on node C
user@node3>nodetool repair
Observation : I was getting number of records equal to the total number of records.Now for keyspace ams there are 2 replicas in the cluster.

What is Decommissioning ?

Decommissioning is a general term for a formal process to remove something from an active status. In Cassandra process of removing node safely from cluster by replicating data on the node to be decommissioned on the other node in the cluster.

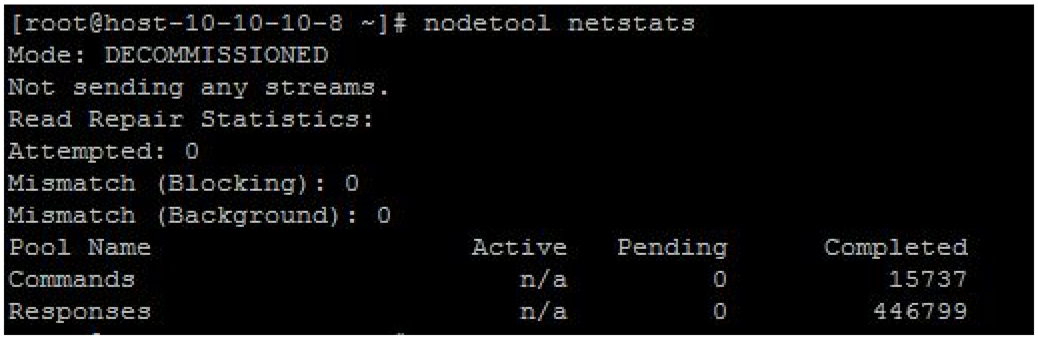
Process of decommissioning a node

First check whether the node is up or down by running ***nodetool status***.



```
[root@host-10-10-10-8 ~]# nodetool status
Datacenter: DC1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens  Owns (effective)  Host ID                               Rack
UN  10.10.10.10  31.04 MB   256     65.5%             b645ff9a-e1bd-4b34-920a-58dd84a766ab  RAC1
UN  10.10.10.8   31.19 MB   256     69.1%             a5e39952-b949-4028-a84b-e3e80dfddd4f  RAC1
UN  10.10.10.6   29.22 MB   256     65.4%             3a47e592-e7e8-4034-aa31-3bc77058b0fd  RAC1
```

The nodetool command shows the status of the node (UN=up, DN=down):
If the node is up, run ***nodetool decommission.***
Result of running nodetool decommission :
This assigns the ranges that the node was responsible for to other nodes and replicates the data appropriately. So there will be no data loss and all the data on the decommissioned node would be replicated on other active nodes
Run nodetool netstats to verify the status of the nodetool decommission.



```
[root@host-10-10-10-8 ~]# nodetool netstats
Mode: DECOMMISSIONED
Not sending any streams.
Read Repair Statistics:
Attempted: 0
Mismatch (Blocking): 0
Mismatch (Background): 0
Pool Name                    Active   Pending      Completed
Commands                        n/a         0          15737
Responses                       n/a         0         446799
```

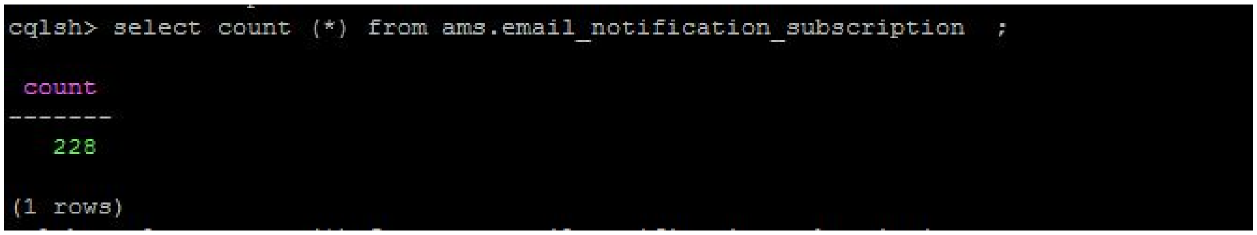
Observation : In Mode you will see DECOMMISSIONED.
Conclusion : Nodetool decommission is safe way to remove node from the cluster if we no longer needed an extra node in the cluster without data loss,with no downtime.
Note : You can bring back the decommissioned node in cluster just by restarting the node.

Effects of Removing cassandra node from cluster without decommissioning it

Following reasons because of which node goes down in the cluster :

- Node is not reachable because of some network issues.
- Node not responding to requests (High CPU, memory issues etc)
- Cassandra process stopped due to some reason.

Following example explains removing cassandra node from cluster without decommissioning it
Consider 2 cases :
Case 1 :
Replication Factor = 1,
(there will be only one copy of each row across the cluster, among any of the three nodes)
Number of Nodes = 3 let's say A ,B , C.
First take count of records on one of the column.

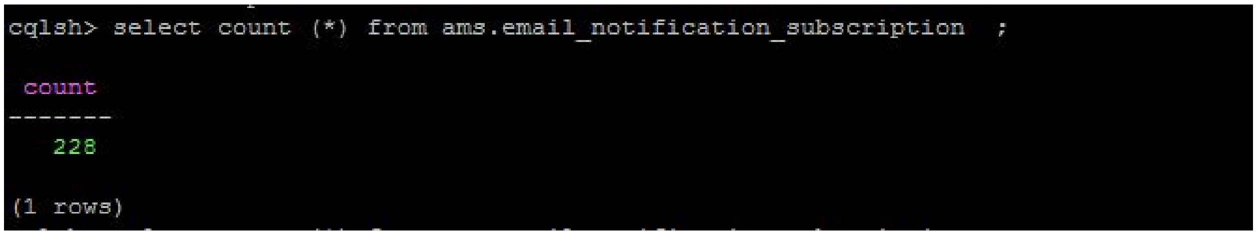


```
cqlsh> select count (*) from ams.email_notification_subscription  ;

 count
-------
   228

(1 rows)
```

After removing node A from cluster .You will not be able to get data that was stored on node A. That will result into reduced count of rows from table .

Case 2 :

Replication Factor = 2,
(there will be 2 copies of each row across the cluster but not on the same node)
Number of Nodes = 3 let's say A ,B , C.
First take count of records on one of the column.



```
cqlsh> select count (*) from ams.email_notification_subscription  ;

 count
-------
   228

(1 rows)
```

After removing node A from cluster. You will not be able to get data that was stored on node A. Count of rows from table will be same. As same data you will be getting same data from the other node, on which it was copied.
Example :
Let's consider the Data distribution of 100 rows, numbered 1 to 100.
First Copy of data:
On node A rows ranging from 1 to 33
On node B rows ranging from 34 to 67
On node C rows ranging from 68 to 100

Second copy of data:
On node A rows ranging from 34 to 67
On node B rows ranging from 68 to 100
On node C rows ranging from 1 to 33

After removing node A scenario will be …
Copy 1:
On node B rows ranging from 34 to 67
On node C rows ranging from 68 to 100

Copy 2:
On node B rows ranging from 68 to 100
On node C rows ranging from 1 to 33

Observation :
All rows range from 1 to 100 are present so there will be data availability.

Conclusion :
Even if one node goes down and provided that replication factor is properly set there is no data unavailability.

Significance and drawback of Secondary Index

Significance of Secondary Indexing

Database provides fast access to items in a table by specifying primary key values.However, many applications might benefit from having one or more secondary (or alternate) keys available, to allow efficient access to data with attributes other than the primary key.To address this, you can create one or more secondary indexes on a table, and issue Query or Scan requests against these indexes. Secondary indexes are used to query a table using a column that is not normally queryable.

How to create index on column

Using cql, to create index on any column in table you have to perform.

CREATE INDEX ON keyspace.column_family (column_name);

Drawbacks of Using secondary Indexing

- Secondary indexes are tricky to use and can impact performance greatly.
- Cassandra stores tombstones in the index until the tombstone limit reaches 100K cells. After exceeding the tombstone limit, the query that uses the indexed value will fail.
- Performing indexing on high cardinality column is not recommended, as it will perform multiple seeks to yield few results.
- Conversely, performing indexing on column with extremely less cardinality such as boolean is not useful.

Cassandra Incremental Backup And Restoring From Snapshots

When incremental backups are enabled (step 2) (disabled by default), Cassandra hard-links each flushed SSTable to a backups directory under the keyspace data directory. This allows storing backups offsite without transferring entire snapshots. Also, incremental backups combine with snapshots to provide a dependable, up-to-date backup mechanism.

Step 1 :

Taking snapshot

The snapshot is created in *data_directory_location/keyspace_name/table_name/snapshots /snapshot_name* directory. Each snapshot directory contains numerous .db files that contain the data at the time of the snapshot.

Procedure:

Perform following command on each node to take snapshot on each node

nodetool -h localhost -p 7199 snapshot

Step 2:

Enable incremental backup

As with snapshots, Cassandra does not automatically clear incremental backup files. DataStax recommends setting up a process to clear incremental backup hard-links each time a new snapshot is created.

Procedure:

Enable incremental Cassandra backup
conf/cassandra.yaml
incremental_backups: true
When incremental backup is enabled (default is off), Cassandra persists flushed SSTable to a backup directory under /var/lib/cassandra/data/my_keyspace/backups/

Step 3:

Restoring From Snapshot

Restoring keyspaces from a snapshot requires all snapshot files for the table, and if using incremental backups, any incremental backup files created after the snapshot was taken.

Generally, before restoring a snapshot, you should truncate the table. If the backup occurs before the delete and you restore the backup after the delete without first truncating, you do not get back the original data (row). Until compaction, the tombstone is in a different SSTable than the original row, so restoring the SSTable containing the original row does not remove the tombstone and the data still appears to be deleted.

Cassandra can only restore data from a snapshot when the table schema exists. If you have not backed up the schema,

Procedure :

Stop the Cassandra process on a node to be restored. For every keyspace, remove the .db files

rm /var/lib/cassandra/data/mykeyspace/*.db

Note : Do not remove the snapshots directory in it

Locate the latest snapshot directory:

/var/lib/cassandra/data/mykeyspace/snapshots/timestamp-thissnapshotname

Copy the snapshot to the data directory:

cp /var/lib/cassandra/data/mykeyspace/snapshots/1304617358646-mylatest snapshot/* /var/lib/cassandra/data/mykeyspace

Copy the incremental backups to the data directory:

cp /var/lib/cassandra/data/mykeyspace/backups/*   /var/lib/cassandra/data/mykeyspace

Repeat the above steps for other keyspaces

Restart the Cassandra process on node