

Apache Cassandra 3.x



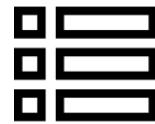


Table of Contents

1. Quick Start: 3
2. Cassandra Architecture: 39
3. Effective CQL: 86
4. Configuring a Cluster: 215
5. Performance Tuning: 273
6. Managing a Cluster: 310
7. Monitoring: 358
8. Application Development:
9. Integration with Apache Spark:



1. Quick Start



Quick Start



In this lesson, we will cover the following topics:

- Introduction to Cassandra
- Installation and configuration
- Starting up and shutting down Cassandra
- Cassandra Cluster Manager (CCM)

Introduction to Cassandra

- Apache Cassandra is a highly available, distributed, partitioned row store.
- It is one of the more popular NoSQL databases used by both small and large companies all over the world to store and efficiently retrieve large amounts of data.



High availability

- Depending on the Replication Factor (RF) and required consistency level, a Cassandra cluster is capable of sustaining operations with one or two nodes in a failure state.
- For example, let's assume that a cluster with a single data center has a keyspace configured for a RF of three.
- This means that the cluster contains three copies of each row of data.
- If an application queries with a consistency level of one, then it can still function properly with one or two nodes in a down state.

Distributed

- Cassandra is known as a distributed database.
- A Cassandra cluster is a collection of nodes (individual instances running Cassandra) all working together to serve the same dataset.
- Nodes can also be grouped together into logical data centers.



Distributed

Assuming that the 500 GB disks of a six node cluster (RF of three) start to reach their maximum capacity, then adding three more nodes (for a total of nine) accomplishes the following:

- Brings the total disk available to the cluster up from 3 TB to 4.5 TB
- The percentage of data that each node is responsible for drops from 50% down to 33%

Partitioned row store

- In Cassandra, rows of data are stored in tables based on the hashed value of the partition key, called a token.
- Each node in the cluster is assigned multiple token ranges, and rows are stored on nodes that are responsible for their tokens.



Installation

The following are the requirements to run Cassandra locally:

- A flavor of Linux or macOS
- A system with between 4 GB and 16 GB of random access memory (RAM)
- A local installation of the Java Development Kit (JDK) version 8, latest patch
- A local installation of Python 2.7 (for cqlsh)
- Your user must have sudo rights to your local system

Installation

- Head to the Apache download site for the Cassandra project (<http://cassandra.apache.org/download/>), choose 3.11.2, and select a mirror to download the latest version of Cassandra.
- When complete, copy the .tar or .gzip file to a location that your user has read and write permissions for.
- This example will assume that this is going to be the ~/local/ directory:

```
mkdir ~/local
```

```
cd ~/local
```

```
cp ~/Downloads/apache-cassandra-3.11.2-bin.tar.gz .
```



Installation

- Untar the file to create your cassandra directory:

```
tar -zxvf apache-cassandra-3.11.2-bin.tar.gz
```

- Some people prefer to rename this directory, like so:

```
mv apache-cassandra-3.11.2/ cassandra/
```



Configuration

cassandra.yaml

- It is usually a good idea to rename your cluster. Inside the conf/cassandra.yaml file, specify a new cluster_name property, overwriting the default Test Cluster:

```
cluster_name: 'PermanentWaves'
```

- The num_tokens property default of 256 has proven to be too high for the newer, 3.x versions of Cassandra.
- Go ahead and set that to 24:

```
num_tokens: 24
```

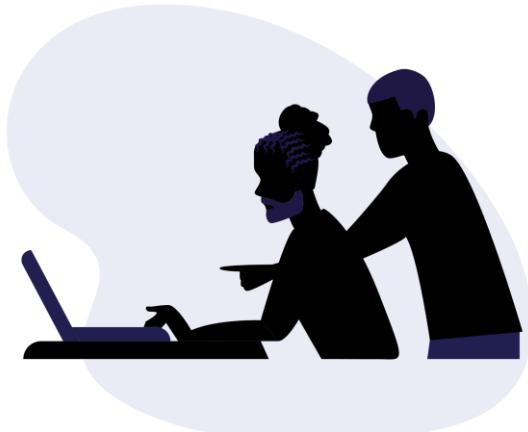


Configuration

- To enable user security, change the authenticator and authorizer properties (from their defaults) to the following values:

authenticator: PasswordAuthenticator

authorizer: CassandraAuthorizer



Configuration

- To configure the node to bind to this IP, adjust the `listen_address` and `rpc_address` properties:

`listen_address: 192.168.0.101`

`rpc_address: 192.168.0.101`

- If you set `listen_address` and `rpc_address`, you'll also need to adjust your seed list (defaults to 127.0.0.1) as well:

`seeds: 192.168.0.101`

- I will also adjust my `endpoint_snitch` property to use `GossipingPropertyFileSnitch`:

`endpoint_snitch: GossipingPropertyFileSnitch`



Configuration

cassandra-rackdc.properties

- To configure this, each node must use GossipingPropertyFileSnitch (as previously mentioned in the preceding `cassandra.yaml` configuration process) and must have its local data center (and rack) settings defined.
- Therefore, I will set the dc and rack properties in the `conf/cassandra-rackdc.properties` file:

`dc=ClockworkAngels`

`rack=R40`



Starting Cassandra

- To start Cassandra locally, execute the Cassandra script.
- If no arguments are passed, it will run in the foreground.
- To have it run in the background, send the -p flag with a destination file for the Process ID (PID):

```
cd cassandra  
bin/cassandra -p cassandra.pid
```



Starting Cassandra

- This will store the PID of the Cassandra process in a file named `cassandra.pid` in the `local/cassandra` directory.
- Several messages will be dumped to the screen.
- The node is successfully running when you see this message:

Starting listening for CQL clients on
localhost/192.168.0.101:9042 (unencrypted).



Starting Cassandra

- This can also be verified with the nodetool status command:

bin/nodetool status

Datacenter: ClockworkAngels

=====

Status=Up/Down

|/ State=Normal/Leaving/Joining/Moving

	-- Address	Load	Tokens	Owns (effective)	Host ID	Rack
UN	192.168.0.101	71.26 KiB	24	100.0%		0edb5efa...
R40						



Cassandra Cluster Manager

- First, let's clone the CCM repository from GitHub, and cd into the directory:

```
git clone https://github.com/riptano/ccm.git  
cd ccm
```



- Next, we'll run the setup program to install CCM:

```
sudo ./setup.py install
```

Cassandra Cluster Manager

- To verify that my local cluster is working, I'll invoke nodetool status via node1:

```
ccm node1 status
```

Datacenter: datacenter1

```
=====
```

Status=Up/Down

|/ State=Normal/Leaving/Joining/Moving

--	Address	Load	Tokens	Owns (effective)	Host ID	Rack	
UN	127.0.0.1	100.56	KiB	1	66.7%	49ecc8dd...	rack1
UN	127.0.0.2	34.81	KiB	1	66.7%	404a8f97...	rack1
UN	127.0.0.3	34.85	KiB	1	66.7%	eed33fc5...	rack1



Cassandra Cluster Manager

- To shut down your cluster, go back and send the stop command to each node:

```
ccm stop node1  
ccm stop node2  
ccm stop node3
```



Using Cassandra with cqlsh

- To start working with Cassandra, let's start the Cassandra Query Language (CQL) shell .
- The shell interface will allow us to execute CQL commands to define, query, and modify our data.
- As this is a new cluster and we have turned on authentication and authorization, we will use the default cassandra and cassandra username and password, as follows:

```
bin/cqlsh 192.168.0.101 -u cassandra -p cassandra  
Connected to PermanentWaves at 192.168.0.101:9042.  
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]  
Use HELP for help.  
cassandra@cqlsh>
```



Using Cassandra with cqlsh

- New users can only be created if authentication and authorization are properly set in the `cassandra.yaml` file:

```
cassandra@cqlsh> CREATE ROLE cassdba WITH  
PASSWORD='flynnLives' AND LOGIN=true and SUPERUSER=true;
```

- Now, set the default `cassandra` user to something long and indecipherable.
- You shouldn't need to use it ever again:

```
cassandra@cqlsh> ALTER ROLE cassandra WITH  
PASSWORD='dsfawesomethingdfhdfshdlongandindecipherabledfhdfh';
```

Using Cassandra with cqlsh

- Then, exit cqlsh using the exit command and log back in as the new cassdba user:

```
cassandra@cqlsh> exit
```

```
bin/cqlsh 192.168.0.101 -u cassdba -p flynnLives
```

```
Connected to PermanentWaves at 192.168.0.101:9042.  
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native  
protocol v4]
```

```
Use HELP for help.
```

```
cassdba@cqlsh>
```



Using Cassandra with cqlsh

- Now, let's create a new keyspace where we can put our tables, as follows:

```
cassdba@cqlsh> CREATE KEYSPACE packt WITH  
replication =  
{'class': 'NetworkTopologyStrategy', 'ClockworkAngels': '1'}  
AND durable_writes = true;
```



Using Cassandra with cqlsh

- With the newly created keyspace, let's go ahead and use it:

```
cassdba@cqlsh> use packt;  
cassdba@cqlsh:packt>
```



Using Cassandra with cqlsh

- Now, let's assume that we have a requirement to build a table for video game scores.
- We will want to keep track of the player by their name, as well as their score and game on which they achieved it.
- A table to store this data would look something like this:

```
CREATE TABLE hi_scores (name TEXT, game TEXT, score  
BIGINT,  
PRIMARY KEY (name,game));
```

Using Cassandra with cqlsh

- Next, we will INSERT data into the table, which will help us understand some of Cassandra's behaviors:

```
INSERT INTO hi_scores (name, game, score) VALUES ('Dad','Pacman',182330);
INSERT INTO hi_scores (name, game, score) VALUES ('Dad','Burgertime',222000);
INSERT INTO hi_scores (name, game, score) VALUES ('Dad','Frogger',15690);
INSERT INTO hi_scores (name, game, score) VALUES ('Dad','Joust',48150);
INSERT INTO hi_scores (name, game, score) VALUES ('Connor','Pacman',182330);
INSERT INTO hi_scores (name, game, score) VALUES ('Connor','Monkey Kong',15800);
INSERT INTO hi_scores (name, game, score) VALUES ('Connor','Frogger',4220);
INSERT INTO hi_scores (name, game, score) VALUES ('Connor','Joust',48850);
INSERT INTO hi_scores (name, game, score) VALUES ('Avery','Galaga',28880);
INSERT INTO hi_scores (name, game, score) VALUES ('Avery','Burgertime',1200);
INSERT INTO hi_scores (name, game, score) VALUES ('Avery','Frogger',1100);
INSERT INTO hi_scores (name, game, score) VALUES ('Avery','Joust',19520);
```

Using Cassandra with cqlsh

- Now, let's execute a CQL query to retrieve the scores of the player named Connor:

```
cassdba@cqlsh:packt> SELECT * FROM hi_scores WHERE  
name='Connor';
```

name	game	score
Connor	Frogger	4220
Connor	Joust	48850
Connor	Monkey Kong	15800
Connor	Pacman	182330
(4 rows)		



Using Cassandra with cqlsh

- That works pretty well.
- But what if we want to see how all of the players did while playing the Joust game, as follows:

```
cassdba@cqlsh:packt> SELECT * FROM hi_scores WHERE  
game='Joust';
```

InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want
to execute this query despite the performance unpredictability, use
ALLOW FILTERING"

Using Cassandra with cqlsh

- Evidently, Cassandra has some problems with that query.
- We'll discuss more about why that is the case later on.
- But, for now, let's build a table that specifically supports querying high scores by game:

```
CREATE TABLE hi_scores_by_game (name TEXT, game TEXT,  
score BIGINT,  
PRIMARY KEY (game,score)) WITH CLUSTERING ORDER BY  
(score DESC);
```

Using Cassandra with cqlsh

- Now, we will duplicate our data into our new query table:

```
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Dad','Pacman',182330);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Dad','Burgertime',222000);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Dad','Frogger',15690);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Dad','Joust',48150);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Connor','Pacman',182330);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Connor','Monkey Kong',15800);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Connor','Frogger',4220);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Connor','Joust',48850);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Avery','Galaga',28880);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Avery','Burgertime',1200);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Avery','Frogger',1100);
INSERT INTO hi_scores_by_game (name, game, score) VALUES ('Avery','Joust',19520);
```

Using Cassandra with cqlsh

- Now, let's try to query while filtering on game with our new table:

```
cassdba@cqlsh:packt> SELECT * FROM  
hi_scores_by_game
```

```
          WHERE game='Joust';
```

game	score	name
Joust	48850	Connor
Joust	48150	Dad
Joust	19520	Avery

(3 rows)



Shutting down Cassandra

- First, we will disable gossip.
- This keeps other nodes from communicating with the node while we are trying to bring it down:

`bin/nodetool disablegossip`

- Next, we will disable the native binary protocol to keep this node from serving client requests:

`bin/nodetool disablebinary`



Shutting down Cassandra

- Then, we will drain the node.
- This will prevent it from accepting writes, and force all in-memory data to be written to disk:

`bin/nodetool drain`

- With the node drained, we can kill the PID:
`kill 'cat cassandra.pid'`



Shutting down Cassandra

- We can verify that the node has stopped by tailing the log:

Refer to file 1_1.txt



Summary

- In this lesson, we introduced Apache Cassandra and some of its design considerations and components.
- These aspects were discussed and a high level description of each was given, as well as how each affects things like cluster layout and data storage.
- Additionally, we built our own local, single-node cluster.



2. Cassandra Architecture



Cassandra Architecture

Specifically, this lesson will cover:

- Problems that Cassandra was designed to solve
- Cassandra's read and write paths
- The role that horizontal scaling plays
- How data is stored on-disk
- How Cassandra handles failure scenarios



Why was Cassandra created?

- What types of problems was Cassandra designed to solve?
- Why does a relational database management system (RDBMS) have difficulty handling those problems?
- If Cassandra works this way, how should I design my data model to achieve the best performance?

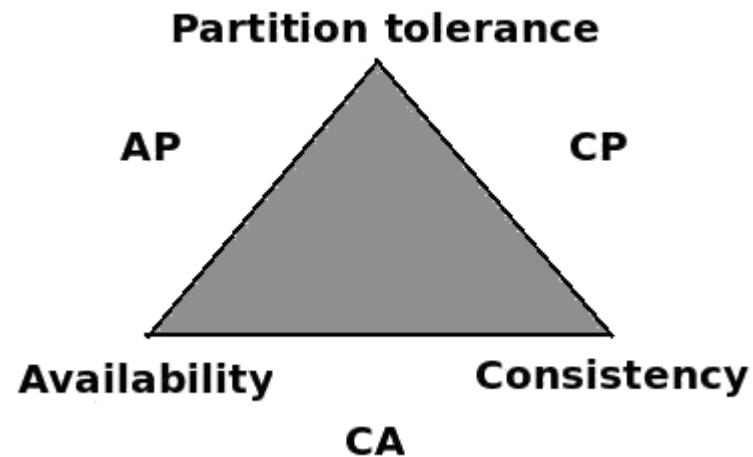
RDBMS and problems at scale

- As the internet grew in popularity around the turn of the century, the systems behind internet architecture began to change.
- When good ideas were built into popular websites, user traffic increased exponentially.
- It was not uncommon in 2001 for too much web traffic being the reason for a popular site being slow or a web server going down.



Cassandra and the CAP theorem

- A graphical representation of Brewer's CAP theorem, using sides of a triangle to represent the combination of the different properties of consistency, availability, and partition tolerance



Cassandra's ring architecture

- An aspect of Cassandra's architecture that demonstrates its AP CAP designation is in how each instance works together.
- A single-instance running in Cassandra is known as a node.
- A group of nodes serving the same dataset is known as a cluster or ring.
- Data written is distributed around the nodes in the cluster.

Cassandra's ring architecture



Partitioners

- The component that helps to determine the nodes responsible for specific partitions of data is known as the partitioner.
- Apache Cassandra installs with three partitioners. You can refer to Apache Cassandra 3.0 for DSE 5.0, Understanding the Architecture.

Cassandra's ring architecture



ByteOrderedPartitioner

- ByteOrderedPartitioner sorts rows in the cluster lexically by partition key.
- Queried results are then returned ordered by that key.
- Tokens are calculated by the hexadecimal value of the beginning of the partition key.



Cassandra's ring architecture

RandomPartitioner

- RandomPartitioner was the default partitioner prior to Apache Cassandra 1.2.
- Tokens are calculated using a MD5 hash of the complete partition key.
- Possible token values range from zero to (2127) - 1.
- For additional information you can refer to Saha S. (2017).
- The Gossip Protocol - Inside Apache Cassandra.

Cassandra's ring architecture

Murmur3Partitioner

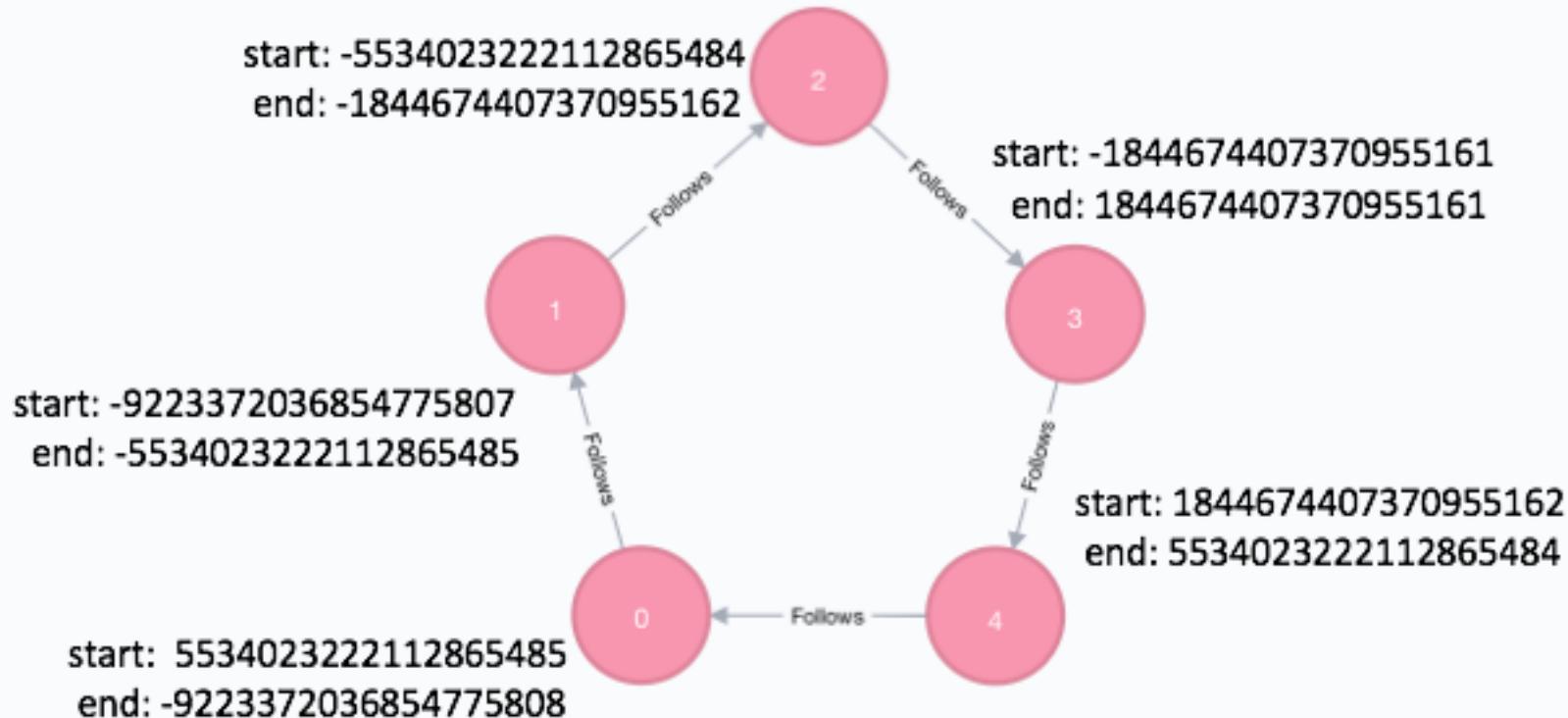
Important points about Apache Cassandra's partitioners:

- A partitioner is configured at the cluster level. You cannot implement a partitioner at the keyspace or table level.
- Partitioners are not compatible. Once you select a partitioner, it cannot be changed without completely reloading the data.
- Use of ByteOrderedPartitioner is considered to be an anti-pattern.
- Murmur3Partitioner is an improvement on the efficiency of RandomPartitioner.

Single token range per node

Node #	Start token	End token
0	5534023222112865485	-9223372036854775808
1	-9223372036854775807	-5534023222112865485
2	-5534023222112865484	-1844674407370955162
3	-1844674407370955161	1844674407370955161
4	1844674407370955162	5534023222112865484

Cassandra's ring architecture



Cassandra's ring architecture

- If we were to store data for the Pacman game, its hashed token value would look like this:

```
cassdba@cqlsh:packt> SELECT game, token(game)  
FROM hi_scores_by_game WHERE game=Pacman;
```

game	system.token(game)
------	--------------------

Pacman	4538621633640690522
--------	---------------------



Cassandra's ring architecture

- Now let's assume that the packt keyspace has the following definition:

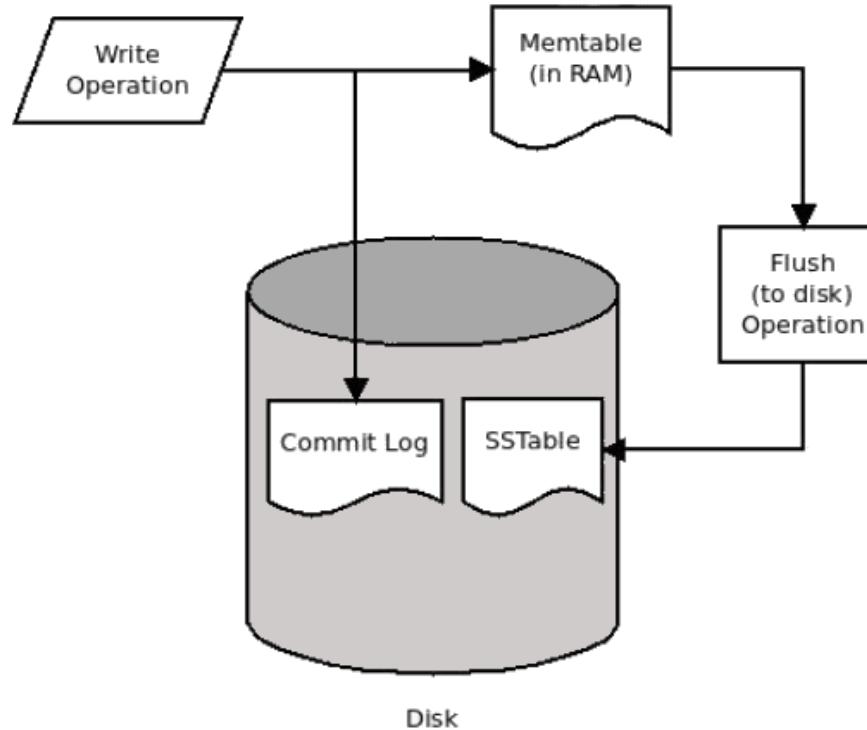
```
CREATE KEYSPACE packt WITH replication =  
  {'class': 'NetworkTopologyStrategy', 'ClockworkAngels':  
  '3'}  
  AND durable_writes = true;
```

Cassandra's ring architecture

Vnodes

Node #	Token ranges
0	79935 to -92233, -30743 to -18446, 55341 to 67638
1	-92232 to -79935, -61488 to 61489, 43043 to 55340
2	-79934 to -67638, -43041 to -30744, 67639 to 79935
3	-67637 to -55340, 61490 to 18446, 18447 to 30744
4	-55339 to -43042, -18445 to -61489, 30745 to 43042

Cassandra's write path

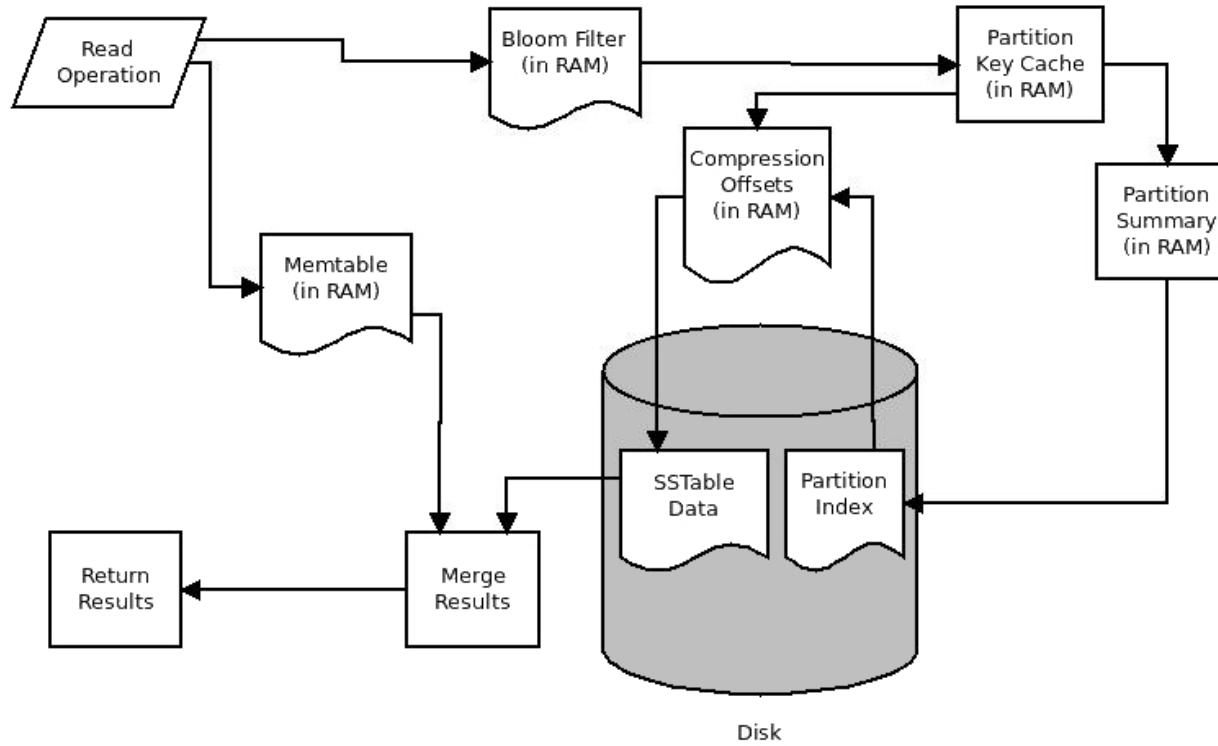


Cassandra's write path

Some quick notes about the write path:

- Data is sorted inside SSTables by token value, and then by clustering keys.
- SSTables are immutable.
- They are written once, and not modified afterward.
- Data written to the same partition key and column values does not update.
- The existing row is made obsolete (not deleted or overwritten) in favor of the timestamp of the new data.

Cassandra's read path



Cassandra's read path

Some quick notes about the read path:

- Data is read sequentially from disk, and will be returned in that order.
- This can be controlled by the table's definition.
- Rows can persist across multiple SSTable files.
- This can slow reads down, so eventually SSTable data is reconciled and the files are condensed during a process called compaction.



On-disk storage

SSTables

- Listing out the files reveals the following:

```
cd data/data/packt/hi_scores-d74bfc40634311e8a387e3d147c7be0f
ls -al
total 72
drwxr-xr-x 11 apoetz apoetz 374 May 29 08:28 .
drwxr-xr-x  6 apoetz apoetz 204 May 29 08:26 ..
-rw-r--r--  1 apoetz apoetz  43 May 29 08:28 mc-1-big-CompressionInfo.db
-rw-r--r--  1 apoetz apoetz 252 May 29 08:28 mc-1-big-Data.db
-rw-r--r--  1 apoetz apoetz  10 May 29 08:28 mc-1-big-Digest.crc32
-rw-r--r--  1 apoetz apoetz  16 May 29 08:28 mc-1-big-Filter.db
-rw-r--r--  1 apoetz apoetz  27 May 29 08:28 mc-1-big-Index.db
-rw-r--r--  1 apoetz apoetz 4675 May 29 08:28 mc-1-big-Statistics.db
-rw-r--r--  1 apoetz apoetz  61 May 29 08:28 mc-1-big-Summary.db
-rw-r--r--  1 apoetz apoetz  92 May 29 08:28 mc-1-big-TOC.txt
```



On-disk storage

How data was structured in prior versions

- As an example, consider the following CQL query from the previous lesson.
- Recall that it retrieves video game high-score data for a player named Connor:

```
cassdba@cqlsh:packt> SELECT * FROM hi_scores WHERE name='Connor';  
name | game      | score  
-----+-----  
Connor | Frogger | 4220  
Connor | Joust    | 48850  
Connor | Monkey Kong | 15800  
Connor | Pacman   | 182330  
(4 rows)
```



On-disk storage

- In Apache Cassandra versions 1.2 to 2.2, CQL is simply a layer that abstracts the SSTable structure.
- If you were using Apache Cassandra 2.1 or earlier, this data could be viewed with the command-line interface tool, also known as `cassandra-cli`:

Refer to the file 2_1.txt

On-disk storage

How data is structured in newer versions

- Consider the CQL query used as the basis for the prior example, where we queried our the hi_scores table for the player named Connor.
- If we were to look at the underlying data using the sstabledump tool, it would look something like this:

Refer to the file 2_2.txt

Additional components of Cassandra

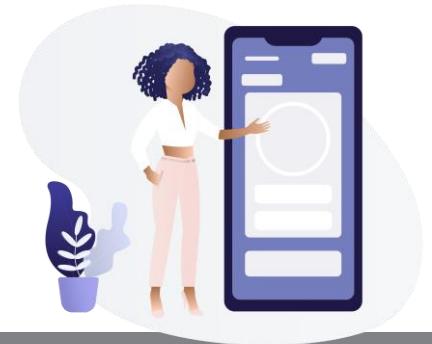
Gossiper

- Gossiper is a peer-to-peer communication protocol that a node uses to communicate with the other nodes in the cluster.
- When the nodes gossip with each other, they share information about themselves and retrieve information on a subset of other nodes in the cluster.

Additional components of Cassandra

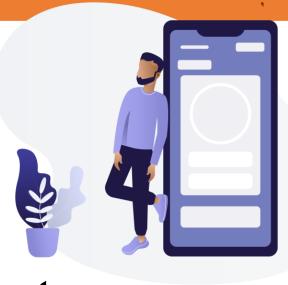
- You can view the currently perceived gossip state of any node using the nodetool utility on any node.
- The following command demonstrates what this looks like with a three-node cluster:

Refer to the file 2_3.txt



Additional components of Cassandra

Snitch



- Cassandra uses a component known as the snitch to direct read and write operations to the appropriate nodes.
- When an operation is sent to the cluster, it is the snitch's job (Williams 2012) to determine which nodes in specific data centers or racks can serve the request.

Additional components of Cassandra

- The replication of data per keyspace is defined in the keyspace definition using the configuration properties of NetworkTopologyStrategy.
- This can be seen in the following example keyspace definition:

```
CREATE KEYSPACE packt WITH replication = {  
    'class': 'NetworkTopologyStrategy',  
    'ClockworkAngels': '3',  
    'PermanentWaves': '2'  
};
```

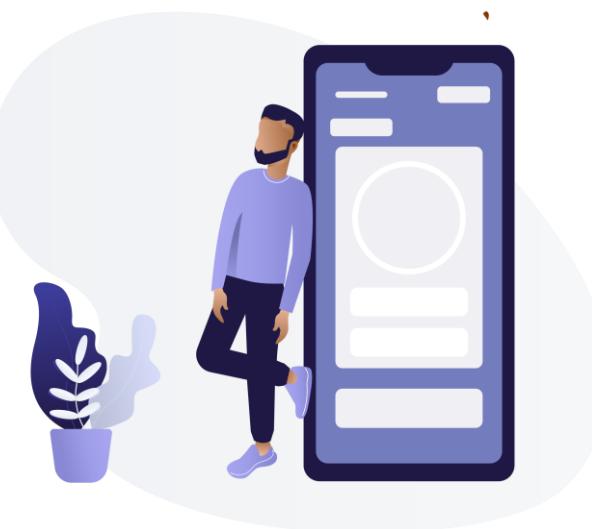


Additional components of Cassandra

- Let's assume that our ClockworkAngels data center has six nodes.
- To maintain the maximum amount of availability possible, we could divide them up like this:

Data center: ClockworkAngels

Node	Rack
192.168.255.1	East_1
192.168.255.2	West_1
192.168.255.3	West_1
192.168.255.4	East_1
192.168.255.5	West_1
192.168.255.6	East_1



Additional components of Cassandra

Phi failure-detector

- Apache Cassandra uses an implementation of the Phi Failure-Accrual algorithm.
- The basic idea is that gossip between nodes is measured on a scale that adjusts to current network conditions.

Additional components of Cassandra

Tombstones

- Deletes in the world of distributed databases are hard to do.
- After all, how do you replicate nothing? Especially a key for nothing which once held a value.
- Tombstones are Cassandra's way of solving that problem.
- As Cassandra uses a log-based storage engine, deletes are essentially writes.

Additional components of Cassandra

Tombstones are eventually removed, but only after two specific conditions are met:

1. The tombstone has existed longer than the table's defined gc_grace_seconds period
2. Compaction runs on that table



Additional components of Cassandra

Hinted handoff

- When a node reports another as down, the node that is still up will store structures, known as hints, for the down node.
- Hints are essentially writes meant for one node that are temporarily stored on another.
- When the down node returns to a healthy status, the hints are then streamed to that node in an attempt to re-sync its data.

Additional components of Cassandra

Compaction

Apache Cassandra's answer to this problem is to periodically execute a process called compaction. When compaction runs, it performs the following functions:

- Multiple data files for a table are merged into a single data file
- Obsoleted data is removed
- Tombstones that have outlived the table's `gc_grace_seconds` period are removed



Additional components of Cassandra

Repair

- Sometimes data replicas on one or more Cassandra nodes can get out of sync.
- There are a variety of reasons for this, including prior network instability, hardware failure, and nodes crashing and staying down past the three-hour hint window.
- When this happens, Apache Cassandra comes with a repair process that can be run to rectify these inconsistencies.

Additional components of Cassandra

Merkle tree calculation



- Merkle trees are created from the bottom up, starting with hashes of the base or leaf data.
- The leaf data is grouped together by partition ranges, and then it is hashed with the hash of its neighboring leaf to create a singular hash for their parent.

Additional components of Cassandra

- We'll query our the hi_scores table for our users whose token value for name (our partition key) falls within that range:

```
cassdba@cqlsh> SELECT DISTINCT name,token(name) FROM packt.hi_scores  
WHERE token(name) >= -9223372036854775807  
AND token(name) <= -7686143364045646507;
```

name system.token(name)

Connor -8880179871968770066
Rob -8366205352999279036
Dad -8339008252759389761
Ryan -8246129210631849419

(4 rows)



Additional components of Cassandra

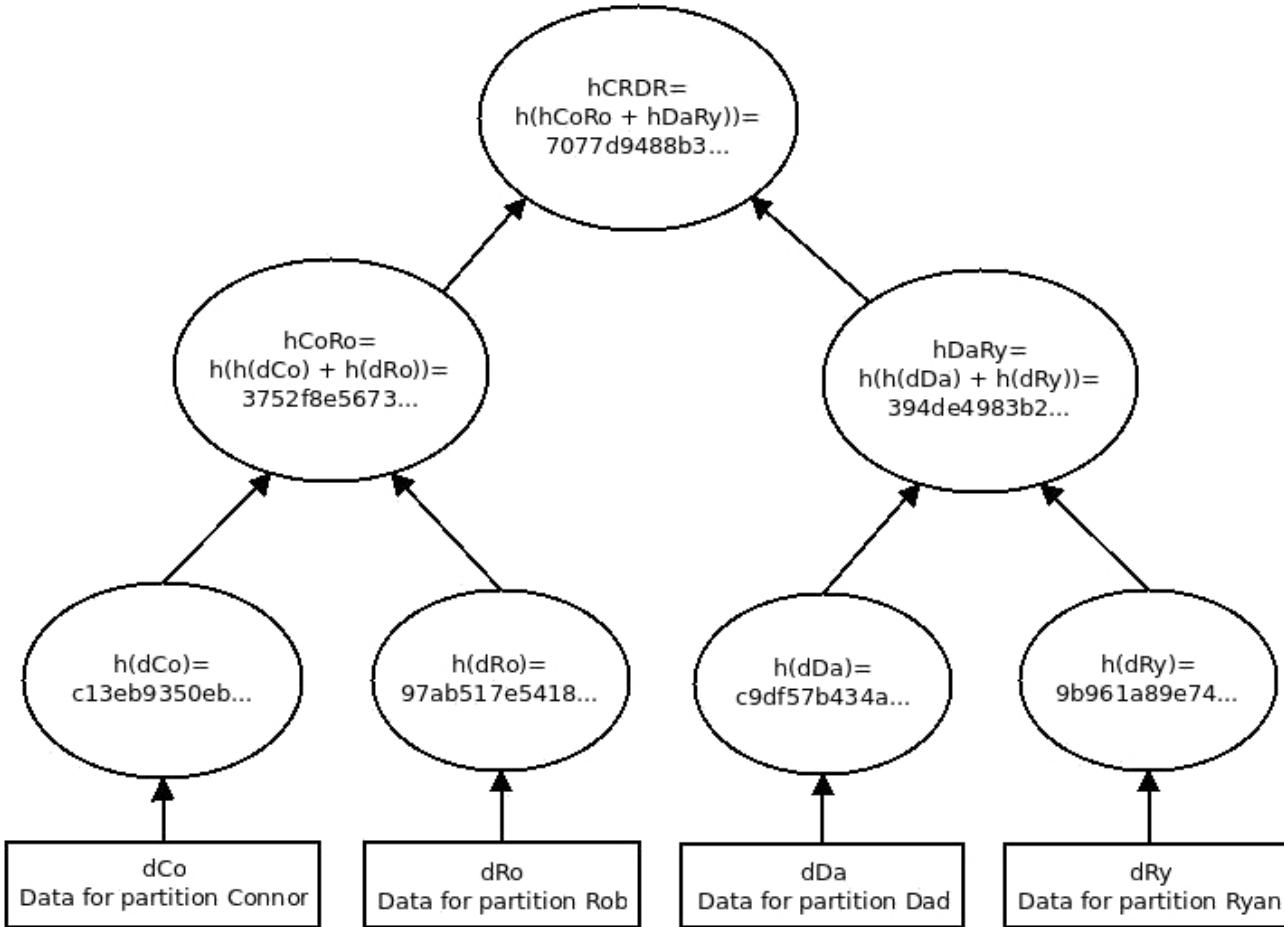
- Now let's compute a hash (MD5 is used for this example) for all data within each partition:

Connor: $h(dCo) = c13eb9350eb2e72eeb172c489faa3d7f$

Rob: $h(dRo) = 97ab517e5418ad2fe700ae45b0ffc5f3$

Dad: $h(dDa) = c9df57b434ad9a674a242e5c70587d8b$

Ryan: $h(dRy) = 9b961a89e744c86f5bffc1864b166514$



Additional components of Cassandra

Once the data for the partitions is hashed, it is then combined with its neighbor nodes to generate their parent node:

- $hCoRo = h(h(dCo) + h(dRo)) =$
3752f8e5673ce8d4f53513aa2cabad40
- $hDaRy = h(h(dDa) + h(dRy)) =$
394de4983b242dcd1603eecc35545a6d
- Likewise, the final hash for the root node can be calculated as
Root node = hCRDR = $h(hCoRo + hDaRy) =$
7077d9488b37f4d39c908eb7560f2323.



Additional components of Cassandra

Read repair

- Read repair is a feature that allows for inconsistent data to be fixed at query time. `read_repair_chance` (and `dclocal_read_repair_chance`) is configured for each table, defaulting to 0.1 (10% chance).
- When it occurs (only with read consistency levels higher than ONE), Apache Cassandra will verify the consistency of the replica with all other nodes not involved in the read request.
- If the requested data is found to be inconsistent, it is fixed immediately.

Additional components of Cassandra

Security

- One criticism of NoSQL data stores is that security is often an afterthought. Cassandra is no exception to this view.
- In fact, Apache Cassandra will install with all security features initially disabled.
- While clusters should be built on secured networks behind enterprise-grade firewalls, by itself this is simply not enough.

Additional components of Cassandra

Authentication

- Client connections are required to provide valid username/password combinations before being allowed access to the cluster.
- To enable this functionality, simply change the authenticator property in the `cassandra.yaml` file from its default (`AllowAllAuthenticator`) to `PasswordAuthenticator`:

`authenticator: PasswordAuthenticator`



Additional components of Cassandra

Authorization

- With authentication set up, additionally enabling user authorization allows each user to be assigned specific permissions to different Cassandra objects. In Apache Cassandra versions 2.2 and up, users are referred to as roles.
- The following are some examples:

```
GRANT SELECT ON KEYSPACE item TO item_app_READONLY;  
GRANT MODIFY ON KEYSPACE store TO store_app_user;  
GRANT ALL PERMISSIONS ON KEYSPACE mobile TO mobile_admin;
```

Additional components of Cassandra

Managing roles

- As role-management is part of the user security backend, it requires PasswordAuthenticator to be enabled before its functionality is activated.
- Here is an example:

```
CREATE ROLE cassdba WITH SUPERUSER=true AND  
LOGIN=true AND PASSWORD='bacon123';  
CREATE ROLE supply_chain_rw WITH LOGIN=true AND  
PASSWORD='avbiuo2t48';
```



Additional components of Cassandra

Client-to-node SSL

- Enabling client-to-node Secure Socket Layer (SSL) security has two main benefits.
- First, each client connecting to the cluster must present a valid certificate that matches a certificate or Certificate Authority (CA) in the node's Java KeyStore.

Additional components of Cassandra

Node-to-node SSL

- Enabling node-to-node SSL security is designed to prevent a specific avenue of attack.
- A node will not be allowed to join the cluster, unless it presents a valid SSL certificate in its Java KeyStore and the Java TrustStore matches with the other nodes in the cluster.

Summary



- In this lesson, we discussed many aspects of Apache Cassandra.
- Some concepts may not have been directly about the Cassandra database, but concepts that influenced its design and use.
- These topics included Brewer's CAP theorem data-distribution and partitioning; Cassandra's read and write paths; how data is stored on-disk; inner workings of components such as the snitch, tombstones, and failure-detection; and an overview of the delivered security features.

3. Effective CQL





Effective CQL

Specifically, we will cover and discuss the following topics:

- The evolution of CQL and the role it plays in the Apache Cassandra universe
- How data is structured and modeled effectively for Apache Cassandra
- How to build primary keys that facilitate high-performing data models at scale
- How CQL differs from SQL
- CQL syntax and how to solve different types of problems using it

An overview of Cassandra data modeling

- Before we get started, let's create a keyspace for this lesson's work:

```
CREATE KEYSPACE packt_ch3 WITH replication =  
{'class': 'NetworkTopologyStrategy',  
'ClockworkAngels':'1'};
```



An overview of Cassandra data modeling

- The CQL for creating that table could look like this:

```
CREATE TABLE playlist (
    band TEXT,
    album TEXT,
    song TEXT,
    running_time TEXT,
    year INT,
    PRIMARY KEY (band,album,song));
```



An overview of Cassandra data modeling

- Now we'll add some data into that table:

```
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Rush','Moving Pictures','Limelight','4:20',1981);
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Rush','Moving Pictures','Tom Sawyer','4:34',1981);
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Rush','Moving Pictures','Red Barchetta','6:10',1981);
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Rush','2112','2112','20:34',1976);
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Rush','Clockwork Angels','Seven Cities of Gold','6:32',2012);
INSERT INTO playlist (band,album,song,running_time,year)
  VALUES ('Coheed and Cambria','Burning Star IV','Welcome Home','6:15',2006);
```



An overview of Cassandra data modeling

Cassandra storage model for early versions up to 2.2

Row key: Rush

Column Key: 2112 | 2112
running_time: 20:34
year: 1976

Column Key: Clockwork Angels | Seven Cities of Gold
running_time: 6:32
year: 2012

Column Key: MovingPictures | Limelight
running_time: 4:20
year: 1981

Column Key: MovingPictures | Red Barchetta
running_time: 6:10
year: 1981

Column Key: MovingPictures | Tom Sawyer
running_time: 4:34
year: 1981

Row key: Coheed and Cambria

Column Key: Burning Star IV | Welcome Home
running_time: 6:15
year: 2006

Cassandra storage model for versions 3.0 and beyond

Partition: Rush

Row - Clustering: 2112 | 2112

running_time: 20:34

year: 1976

Row - Clustering: Clockwork Angels | Seven Cities of Gold

running_time: 6:32

year: 2012

Row - Clustering: MovingPictures | Limelight

running_time: 4:20

year: 1981

Row - Clustering: MovingPictures | Red Barchetta

running_time: 6:10

year: 1981

Row - Clustering: MovingPictures | Tom Sawyer

running_time: 4:34

year: 1981

Partition: Coheed and Cambria

Row - Clustering: Burning Star IV | Welcome Home

running_time: 6:15

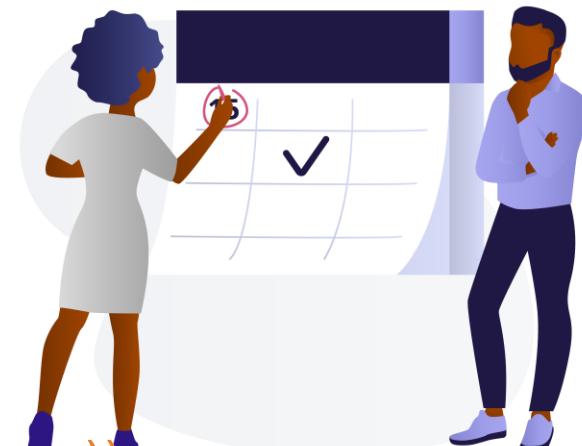
year: 2006

An overview of Cassandra data modeling

Data cells

- Assume the following table definition for keeping track of weather station readings:

```
CREATE TABLE weather_data_by_date (
    month BIGINT,
    day BIGINT,
    station_id UUID,
    time timestamp,
    temperature DOUBLE,
    wind_speed DOUBLE,
    PRIMARY KEY ((month,day),station_id,time));
```



cqlsh

Logging into cqlsh

- You will find cqlsh in the /bin directory of your Cassandra home directory.
- You can start it by invoking it with a host IP address to connect to and any flags required by your cluster or session to connect:

```
cqlsh <host_ip> <port> [flags/options]
```

cqlsh

- In the interest of getting users on the correct path, we'll show you how to log into a cluster secured with both auth and ssl via cqlsh:

```
bin/cqlsh 192.168.0.101 -u cassdba -p flynnLives --ssl
```

```
Connected to PermanentWaves at 192.168.0.101:9042.  
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native  
protocol v4]
```

```
Use HELP for help.
```

```
cassdba@cqlsh>
```

cqlsh

Local cluster without security enabled

- This is the easiest scenario. It's also the least secure, so you shouldn't get into the habit of setting things up this way (which is what too much of running on your own machine can do):

bin/cqlsh



cqlsh

Remote cluster with user security enabled

- When connecting to a remote cluster that uses authentication and authorization, you should have to provide the IP address of the host, along with a valid username and password:

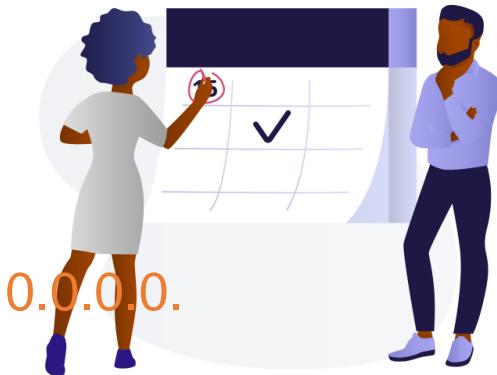
```
bin/cqlsh 192.168.0.101 -u cassdba -p flynnLives
```

cqlsh

- If you have trouble connecting to a remote cluster via cqlsh, try to figure out what it is using as its broadcast address, like this:

```
grep broadcast_ conf/cassandra.yaml
```

```
# broadcast_address: 1.2.3.4
broadcast_address: 192.168.0.101
# set broadcast_rpc_address to a value other than 0.0.0.0.
# rpc_address. If rpc_address is set to 0.0.0.0,
broadcast_rpc_address must
broadcast_rpc_address: 192.168.0.101
# Uses public IPs as broadcast_address to allow cross-region
```

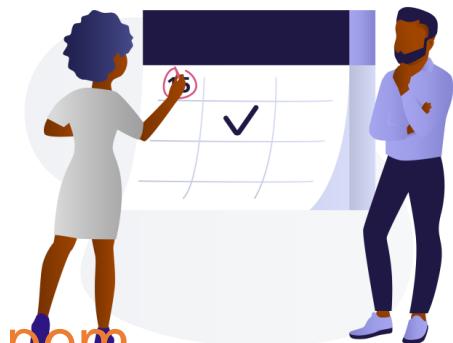


cqlsh

Remote cluster with auth and SSL enabled

- A common cqlshrc file used for connecting to a client-to-node SSL enabled cluster may look like this:

```
cat cqlshrc
[connection]
factory = cqlshlib.ssl.ssl_transport_factory
[ssl]
certfile =
/home/aploetz/.cassandra/permanentWaves.pem
validate = false
```



cqlsh

- With that configuration in place, execute cqlsh providing the IP address of the host, along with a valid username and password and the --ssl flag:

```
bin/cqlsh 192.168.0.101 -u cassdba -p flynnLives --ssl
```

Converting the Java keyStore into a PKCS12 keyStore

For this step, it's a good idea to copy the node's keyStore to a location where the following is applicable:

- You have full access or ownership of it
- You aren't risking corrupting the original



cqlsh

- For this example, I'll copy it to the .cassandra directory underneath my home dir (which you may need sudo access to do):

```
sudo cp conf/keystore ~/.cassandra/keystore
```

- Now, converting the Java keyStore into a PKCS12 keyStore can easily be done:

```
cd ~/.cassandra
```

```
keytool -importkeystore -srckeystore keystore -destkeystore  
p12keystore -deststoretype PKCS12 -srcstorepass  
keystorepassword -deststorepass keystorepassword
```

Exporting the certificate from the PKCS12 keyStore

- Here, you will need the OpenSSL toolkit, and use it to create a Privacy Enhanced Mail (PEM) file:

```
openssl pkcs12 -in p12keystore -nokeys -out  
permanentWaves.pem -passin pass:keystorepassword
```



cqlsh

Modifying your cqlshrc file

- Connecting over SSL via cqlsh requires the use of the cqlshrc file.
- As indicated in the previous section, your cqlshrc file should be adjusted to reference the PEM file created in the prior step:

[connection]

```
factory = cqlshlib.ssl.ssl_transport_factory
```

[ssl]

```
certfile = /home/aploetz/.cassandra/permanentWaves.pem
```

```
validate = false
```



cqlsh

Testing your connection via cqlsh

- With the certificate converted and exported, and your `cqlshrc` file updated, you should now be able to connect to your client-to-node SSL enabled cluster:

```
bin/cqlsh 192.168.0.101 -u cassdba -p flynnLives --ssl
```

Creating a keyspace

- Creating a keyspace is a simple operation and can be done like this:

```
CREATE KEYSPACE [IF NOT EXISTS]  
<keyspace_name>  
WITH replication =  
{'class': '<replication_strategy>',  
 '<data_center_name>':'<replication_factor>' }  
AND durable_writes = <true/false>;
```



Single data center example

- The following example will create the packt_ch3 keyspace.
- When a write occurs, one replica will be written to the cluster. Writes will also be sent to the commit log for extra durability:

```
CREATE KEYSPACE IF NOT EXISTS packt_ch3  
WITH replication =  
{'class': 'NetworkTopologyStrategy',  
'ClockworkAngels':'1'}  
AND durable_writes = true;
```



Multi-data center example

- When a write occurs, two replicas will be written to the ClockworkAngels data center, and three will be written to the PermanentWaves and MovingPictures data centers.
- Writes will also be sent to the commit log for extra durability:

```
CREATE KEYSPACE packt_ch3_mdc  
WITH replication = {'class': 'NetworkTopologyStrategy',  
'ClockworkAngels':'2', 'MovingPictures':'3',  
'PermanentWaves':'3'}  
AND durable_writes = true;
```



Multi-data center example

- Once you have created your keyspace, you can use it to avoid having to keep typing it later.
- Notice that this will also change your Command Prompt:

```
cassdba@cqlsh> use packt_ch3 ;  
cassdba@cqlsh:packt_ch3>
```



Creating a table

- If someone refers to column families, it is safe to assume that they are referring to structures in older versions of Apache Cassandra:

```
CREATE TABLE [IF NOT EXISTS]
[keyspace_name.]<table_name>
<column_name> <column_type>,
[additional <column_name> <column_type>],
PRIMARY KEY ((<partition_key>[, additional
<partition_key>])[, <clustering_keys>]);
[WITH <options>];
```



Simple table example

- For a small table with simple query requirements, something like this might be sufficient:

```
CREATE TABLE users (  
    username TEXT,  
    email TEXT,  
    department TEXT,  
    title TEXT,  
    ad_groups TEXT,  
    PRIMARY KEY (username));
```



Clustering key example

- Let's say that we wanted to be able to offer up the same data, but to support a query for users by department.
- The table definition would change to something like this:

Refer to the file 3_1.txt



Clustering key example

- For a later example, let's write some data to that table:

Refer to the file 3_2.txt



Clustering key example

- Let's assume that we want to query security entrance logs for employees.
- If we were to use a clustering key on a new column named time to one of the preceding examples, we would be continually adding cells to each partition.
- So we'll build this PRIMARY KEY to partition our data by entrance and day, as well as cluster it on checkpoint_time and username:

Refer to the file 3_3.txt

Table options

- Clustering order specifies the column(s) and direction by which the table data (within a partition) should be stored on disk.
- As Cassandra stores data written sequentially, it's also faster when it can read sequentially, so learning how to utilize clustering keys in your models can help with performance:

CLUSTERING ORDER BY (<clustering_key_name <ASC|DESC>)

Table options

- This setting represents the false positive probability for the table's bloom filter.
- The value can range between 0.0 and 1.0, with a recommended setting of 0.1 (10%):

bloom_filter_fp_chance = 0.1



Table options

- The caching property dictates which caching features are used by this table.
- There are two types of caching that can be used by a table: key caching and row caching.
- Key caching is enabled by default.
- Row caching can take up larger amounts of resources, so it defaults to disabled or NONE:

```
caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
```

Table options

- This property allows for the addition of comment or description for the table, assuming that it is not clear from its name:

`comment = "`

- This enables the table for Change Data Capture (CDC) logging and cdc defaults to FALSE:

`cdc = FALSE`

- A map structure that allows for the configuration of the compaction strategy, and the properties that govern it:

`compaction = {'class':`

`'org.apache.cassandra.db.compaction.LeveledCompactionStrategy'}`

Table options

- If you create a table without configuring compaction, it will be set to the following options:

```
compaction = {'class':  
    'org.apache.cassandra.db.compaction.SizeTieredComp  
    actionStrategy',  
    'max_threshold': '32', 'min_threshold': '4'}
```



Table options

- LeveledCompactionStrategy: The leveled compaction strategy builds out SSTable files as levels, where each level is 10 times the size of the previous level.
- In this way, a row can be guaranteed to be contained within a single SSTable file 90% of the time.
- A typical leveled compaction config looks like this:

```
compaction = {'class':  
    'org.apache.cassandra.db.compaction.LeveledCompactionStrategy',  
    'sstable_size_in_mb': '160'}
```

Table options

- TimeWindowCompactionStrategy: Time-window compaction builds SSTable files according to time-based buckets.
- Rows are then stored according to these (configurable) buckets.
- Configuration for time window compaction looks like this:

`compaction = {'class':`

```
'org.apache.cassandra.db.compaction.TimeWindowCompaction
Strategy',
'compaction_window_unit': 'hours', 'compaction_window_size':
'24'}
```

Table options

- This is especially useful for time-series data, as it allows data within the same time period to be stored together.

```
compression = {'chunk_length_in_kb': '64', 'class':  
'org.apache.cassandra.io.compress.LZ4Compressor'}
```



Table options

- Apache Cassandra ships with three compressor classes: LZ4Compressor (default), SnappyCompressor, and DeflateCompressor.
- Once the class has been specified, chunk_length can also be specified.
- By default, a table will use LZ4Compressor, and will be configured as shown here:

```
compression = {'class':  
    'org.apache.cassandra.io.compress.LZ4Compressor',  
    'chunk_length_in_kb': '64'}
```

Table options

- It should be noted that the default chunk_length_in_kb of 64 is intended for write-heavy workloads.
- Access patterns that are more evenly read and write, or read-heavy may see a performance benefit from bringing that value down to as low as four.
- As always, be sure to test significant changes, like this:

`crc_check_chance = 1.0`

Table options

- With compression enabled, this property defines the probability that the CRC compression checksums will be verified on a read operation.
- The default value is 1.0, or 100%:

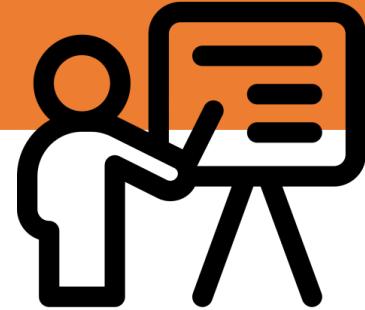
`dclocal_read_repair_chance = 0.1`

- This property specifies the probability that a read repair will be invoked on a read.
- The read repair will only be enforced within the same (local) data center:

`default_time_to_live = 0`



Table options



gc_grace_seconds = 864000

- This property specifies the amount of time (in seconds) that tombstones in the table will persist before they are eligible for collection via compaction.
- By default, this is set to 864000 seconds, or 10 days.

min_index_interval & max_index_interval

Table options

- The actual value used is determined by how often the table is accessed.
- Default values for these properties are set as follows:

`max_index_interval = 2048`

`min_index_interval = 128`

`memtable_flush_period_in_ms = 0`

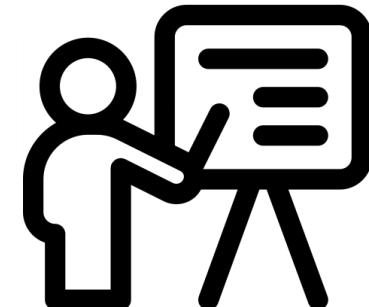


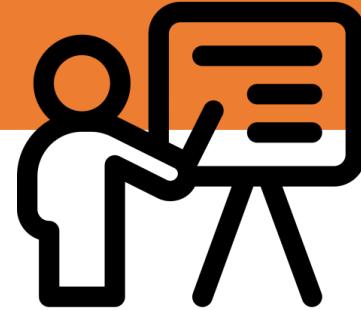
Table options

- The number of milliseconds before the table's memtables are flushed from RAM to disk.
- The default is zero, effectively leaving the triggering of memtable flushes up to the commit log and the defined value of memtable_cleanup_threshold:

`read_repair_chance = 0.0`



Table options



- This defaults to zero (0.0), to ensure that read repairs are limited to a single data center at a time (and therefore less expensive):

```
speculative_retry = '99PERCENTILE';
```

Data types

CQL type	Java class	Description
ascii	String	US-ASCII (United States – American Standard Codes for Information Interchange) character string.
bigint	Long	64-bit signed long.
blob	java.nio.ByteBuffer	Supports storage of BLOBs (binary large objects).
date	java.time.LocalDate	32-bit integer denoting the number of days elapsed since January 1, 1970. Dates can also be specified in CQL queries as strings (such as July 22, 2018).
decimal	java.math.BigDecimal	Arbitrary precision floating-point numeric.
double	Double	64-bit floating-point numeric.
float	Float	32-bit floating-point numeric.

Data types

inet	java.net.InetAddress	String type for working with IP addresses. Supports both IPv4 and IPv4 addresses.
int	Integer	32-bit signed Integer.
list	java.util.List	A collection of ordered items of a specified type.
map	java.util.Map	A key/value collection that stores values by a unique key. Type can be specified for both key and value.
set	java.util.Set	A collection of unique items of a specified type.
smallint	Short	16-bit integer.
text	String	UTF-8 character string.
time	Long	The current time in milliseconds elapsed since midnight of the current day.

Data types

timestamp	java.util.Date	Date/time with milliseconds, can also be specified in CQL queries as strings (such as 2018-07-22 22:51:13.442).
timeuuid	java.util.UUID	Type 1 UUID (128-bit), used for storing times.
tinyint	Byte	8-bit integer.
tuple	com.datastax.driver.core.TupleValue	A tuple type, consisting of a collection of 2 to 3 values.
uuid	java.util.UUID	Type 4 (randomly-generated) UUID.
varchar	String	UTF-8 character string.
varint	java.math.BigInteger	Arbitrary precision integer.

Type conversion

Starting type	Compatible types
ascii	text, varchar
bigint	timestamp, varint
date	timestamp
int	bigint
text	varchar
time	bigint
timestamp	bigint, varint, date
timeuuid	UUID
varchar	text

The primary key

- The most important part of designing your data model is how you define the primary key of your tables.
- As mentioned previously in this lesson, primary keys are built at table-creation time, and have the following options:

PRIMARY KEY ((<partition_key>[,additional
<partition_key>])[,<clustering_keys>])



Designing a primary key

When designing a primary key, Cassandra modelers should have two main considerations:

- Does this primary key allow for even distribution, at scale?
- Does this primary key match the required query pattern(s), at scale?

Selecting a good partition key

- Let's think about a time-series model. If I'm going to keep track of security logs for a company's employees (with multiple locations), I may think about building a model like this:

```
CREATE TABLE security_logs_by_location (
    employee_id TEXT,
    time_in TIMESTAMP,
    location_id TEXT,
    mailstop TEXT,
    PRIMARY KEY (location_id, time_in, employee_id));
```



Type conversion

- We can do that by introducing a day bucket into the model, resulting in a composite partition key:

```
DROP TABLE security_logs_by_location;
```

```
CREATE TABLE security_logs_by_location (
    employee_id TEXT,
    time_in TIMESTAMP,
    location_id TEXT,
    day INT,
    mailstop TEXT,
    PRIMARY KEY ((location_id, day), time_in, employee_id));
```



Type conversion

- Now when I insert into this table and query it, it looks something like this:

Refer to file 3_4.txt



Selecting a good clustering key

- Consider the table used in the preceding examples.
- This section will make more sense with more data, so let's start by adding a few more rows:

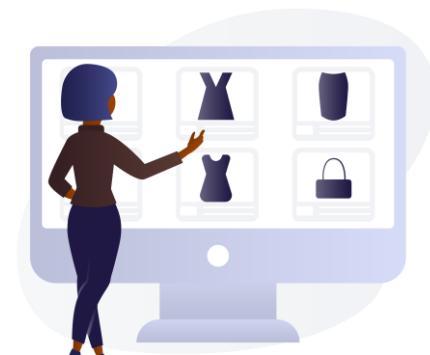
Refer to file 3_5.txt



Selecting a good clustering key

- Now, I'll query the table for all employees entering the MPLS2 building between 6 AM and 10 AM, on July 23, 2018:

Refer to file 3_6.txt



Selecting a good clustering key

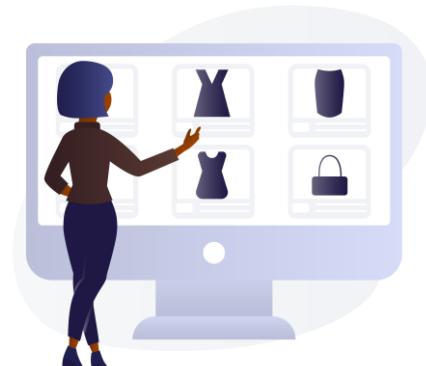
- Now assume that the requirements change slightly, in that the result set needs to be in descending order.
- How can we solve for that? Well, we could specify an ORDER BY clause at query time, but flipping the sort direction of a large result set can be costly for performance.
- What is the best way to solve that? By creating a table designed to serve that query naturally, of course:

Refer to file 3_7.txt

Selecting a good clustering key

- If I duplicate my data into this table as well, I can run that same query and get my result set in descending order:

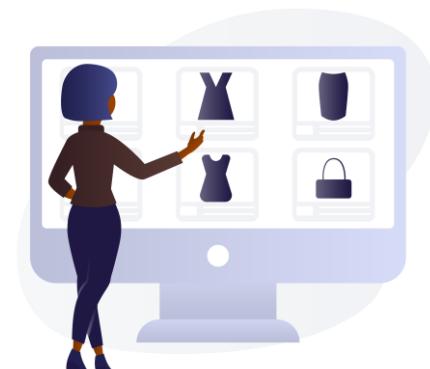
Refer to file 3_8.txt



Querying data

- While Apache Cassandra is known for its restrictive query model (design your tables to suit your queries), the previous content has shown that CQL can still be quite powerful.
- Consider the following table:

Refer to file 3_9.txt



Querying data

- Let's start by querying everything for pk1:

```
SELECT * FROM query_test WHERE pk1='a';
```

InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you
want to execute this query despite the performance
unpredictability, use ALLOW FILTERING"

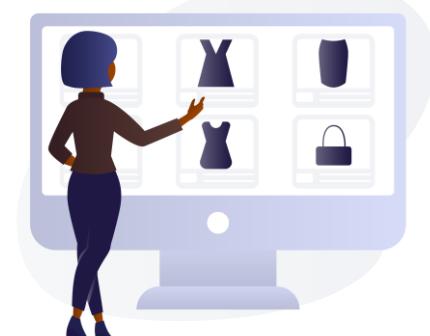
Querying data

- Without both pk1 and pk2 specified, a single node containing the requested data cannot be ascertained. However, it does say the following:

If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING

- So let's give that a try:

Refer to file 3_10.txt



Querying data

- So, let's try that query again, and this time we'll specify the complete partition key, as well as the first clustering key:

```
SELECT * FROM query_test  
WHERE pk1='a' AND pk2='b' AND ck3='c2';
```

pk1 | pk2 | ck3 | ck4 | c5

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

a | b | c2 | d2 | e2

a | b | c2 | d3 | e3

a | b | c2 | d4 | e4

(3 rows)



Querying data

- That works. So what if we just want to query for a specific ck4, but we don't know which ck3 it's under? Let's try skipping ck3:

```
SELECT * FROM query_test  
WHERE pk1='a' AND pk2='b' AND ck4='d2';
```

InvalidRequest: Error from server: code=2200 [Invalid query]
message="PRIMARY KEY column "ck4" cannot be restricted
as preceding column "ck3" is not restricted"

Querying data

- So how do we solve this issue? Let's use ALLOW FILTERING:

```
SELECT * FROM query_test  
WHERE pk1='a' AND pk2='b' AND ck4='d2' ALLOW  
FILTERING;
```

pk1 | pk2 | ck3 | ck4 | c5

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

a | b | c2 | d2 | e2

(1 rows)



Querying data

- For a quick detour, what if I wanted to know what time it was on my Apache Cassandra cluster?
- I could query my table using the now() function:

Refer to file 3_11.txt



Querying data

- Let's solve problems one and two by changing the table we're querying.
- Let's try that on the system.local table:

```
SELECT now() FROM system.local ;
```

system.now()

94a88ad0-8fbb-11e8-91d4-a7c67cc60e89

(1 rows)



Querying data

We can use the dateof() function for this:

```
SELECT dateof(now()) FROM system.local;
```

```
system.dateof(system.now())
```

```
2018-07-25 03:37:08.045000+0000
```

```
(1 rows)
```



The IN operator

- Allowing filters in the WHERE clause to be specified on an OR basis would force Cassandra to perform random reads, which really works against how it was built.
- However, queries can be made to perform similarly to OR, via the IN operator:

```
SELECT * FROM query_test WHERE pk1='a' AND pk2 IN  
('b','c');
```

The IN operator

- We are only giving it one part of the partition key.
- This means that it will have to designate one node as a coordinator node, and then scan each node to build the result set:

```
SELECT * FROM query_test WHERE pk1='a' AND  
pk2='b' AND ck3 IN ('c1','c3');
```

Writing data

- Assume that we need to keep track of statuses for orders from an e-commerce website.
- Consider the following table:

```
CREATE TABLE order_status (
    status TEXT,
    order_id UUID,
    shipping_weight_kg DECIMAL,
    total DECIMAL,
    PRIMARY KEY (status,order_id))
WITH CLUSTERING ORDER BY (order_id DESC);
```



Inserting data

- Additionally, every column that you wish to provide a value for must be specified in a parenthesis list, followed by the VALUES in their own parenthesis list:

Refer to file 3_13.txt



Inserting data

- As you can see, not all columns need to be specified.
- In our business case, assume that we do not know the shipping weight until the order has been PICKED.
- If I query for all orders currently in a PENDING status, it shows shipping_weight_kg as null:

Refer to file 3_13.txt



Updating data

- Updates and inserts are treated the same in Cassandra, so we could actually update our row with the **INSERT** statement:

```
INSERT INTO order_status  
(status,order_id,shipping_weight_kg)  
VALUES ('PENDING',fcb15fc2-feaa-4ba9-a3c6-  
899d1107cce9,1.4);
```



Updating data

- Or, we can also use the UPDATE statement that we know from SQL:

```
UPDATE order_status SET shipping_weight_kg=1.4  
WHERE status='PENDING'  
AND order_id=fcb15fc2-feaa-4ba9-a3c6-  
899d1107cce9;
```



Updating data

- Either way, we can then query our row and see this result:

```
SELECT * FROM order_status  
WHERE status='PENDING'  
AND order_id=fcb15fc2-feaa-4ba9-a3c6-899d1107cce9;
```

status	order_id	shipping_weight_kg	total
PENDING	fcb15fc2-feaa-4ba9-a3c6-899d1107cce9	114.22	

(1 rows)



Updating data

- Ok, so now how do we set the PENDING status to PICKED?
- Let's start by trying to UPDATE it, as we would in SQL:

```
UPDATE order_status SET status='PICKED'  
WHERE order_id='fcb15fc2-feaa-4ba9-a3c6-899d1107cce9';
```

InvalidRequest: Error from server: code=2200 [Invalid query]
message="PRIMARY KEY part status found in SET part"

Updating data

- With the UPDATE statement in CQL, all the PRIMARY KEY components are required to be specified in the WHERE clause.
- So how about we try specifying both PRIMARY KEY components in WHERE, and both columns with values in SET:

```
UPDATE order_status SET  
shipping_weight_kg=1.4,total=114.22  
WHERE status='PICKED'  
AND order_id=fcb15fc2-feaa-4ba9-a3c6-899d1107cce9;
```

Updating data

- So that doesn't error out, but did it work? To figure that out, let's run a (bad) query using ALLOW FILTERING:

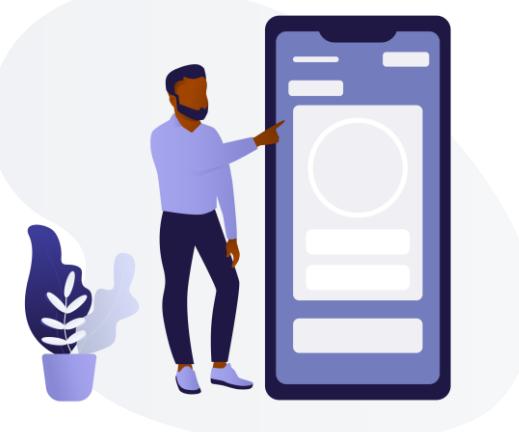
Refer to file 3_14.txt



Deleting data

- In our case, we have an extra row for our order that we need to delete:

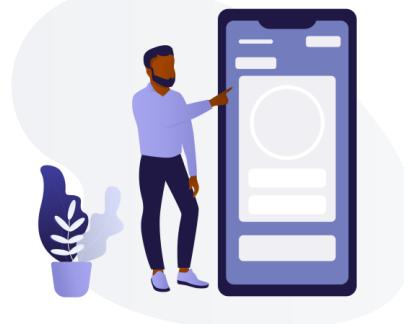
```
DELETE FROM order_status  
WHERE status='PENDING'  
AND order_id=fcb15fc2-feaa-4ba9-a3c6-  
899d1107cce9;
```



Deleting data

- As mentioned previously, DELETE can also enforce the removal of individual column values:

```
DELETE shipping_weight_kg FROM order_status  
WHERE status='PICKED'  
AND order_id=99886f63-f271-459d-b0b1-  
218c09cd05a2;
```



Lightweight transactions

- In any case, an INSERT statement can only check whether a row does not already exist for the specified the PRIMARY KEY components.
- If we consider our attempts to insert a new row to set our order status to PENDING, this could have been used:

```
INSERT INTO order_status  
(status,order_id,shipping_weight_kg  
VALUES ('PENDING',fcb15fc2-feaa-4ba9-a3c6-  
899d1107cce9,1.4)  
IF NOT EXISTS;
```



Lightweight transactions

- In our previous example, we could make our update to shipping_weight_kg and order total based on a threshold for shipping_weight_kg:

```
UPDATE order_status SET  
shipping_weight_kg=1.4,total=114.22  
WHERE status='PICKED' AND order_id=fcb15fc2-feaa-  
4ba9-a3c6-899d1107cce9  
IF shipping_weight_kg > 1.0;
```

Lightweight transactions

- Deletes can also make use of lightweight transactions, much in the same way that updates do:

```
DELETE FROM order_status  
WHERE status='PENDING'  
AND order_id=fcb15fc2-feaa-4ba9-a3c6-899d1107cce9  
IF EXISTS;
```

Executing a BATCH statement

- BATCH statements in CQL can be applied as such:

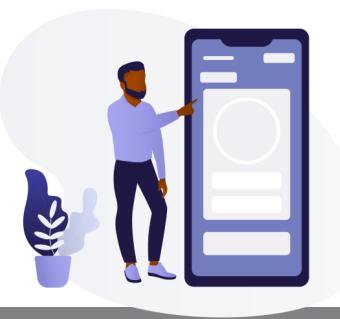
```
BEGIN BATCH
```

```
    INSERT INTO security_logs_by_location  
    (location_id,day,time_in,employee_id,mailstop)  
    VALUES ('MPLS2',20180726,'2018-07-26 11:45:22.004','robp','M266');  
    INSERT INTO security_logs_by_location_desc  
    (location_id,day,time_in,employee_id,mailstop)  
    VALUES ('MPLS2',20180726,'2018-07-26 11:45:22.004','robp','M266');  
    APPLY BATCH;
```

The expiring cell

- Apache Cassandra allows for data in cells to be expired. This is called setting a TTL.
- TTLs can be applied at write-time, or they can be enforced at the table level with a default value.
- To set a TTL on a row at write time, utilize the USING TTL clause:

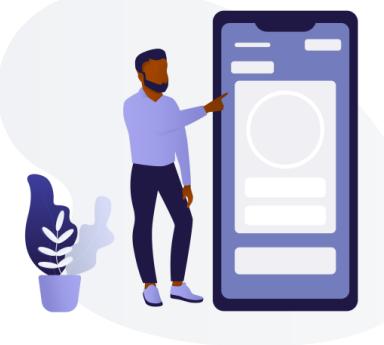
```
INSERT INTO query_test (pk1,pk2,ck3,ck4,c5)  
VALUES ('f','g','c4','d6','e7')  
USING TTL 86400;
```



The expiring cell

- Likewise, TTLs can also be applied with the UPDATE statement:

```
UPDATE query_test  
USING TTL 86400  
SET c5='e7'  
WHERE pk1='f' AND pk2='g' AND ck3='c4' AND ck4='d6';
```



Altering a keyspace

- Changing a keyspace to use a different RF or strategy is a simple matter of using the ALTER KEYSPACE command.
- Let's assume that we have created a keyspace called packt_test:

```
CREATE KEYSPACE packt_test WITH replication = {  
    'class': 'SimpleStrategy', 'replication_factor': '1'}  
AND durable_writes = true;
```

Altering a keyspace

- As it is preferable to use NetworkTopologyStrategy, we can alter that easily:

```
ALTER KEYSPACE packt_test WITH replication = {  
  'class': 'NetworkTopologyStrategy',  
  'datacenter1': '1'};
```

- If, at some point, we want to add our second data center, that command would look like this:

```
ALTER KEYSPACE packt_test WITH replication = { 'class':  
  'NetworkTopologyStrategy',  
  'datacenter1': '1', 'datacenter2': '1'};
```

Altering a keyspace

- If we added more nodes to both data centers, and needed to increase the RF in each, we can simply run this:

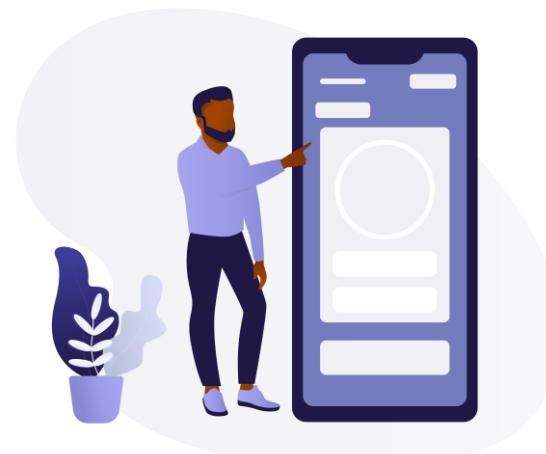
```
ALTER KEYSPACE packt_test WITH replication =  
{'class': 'NetworkTopologyStrategy',  
'datacenter1': '3', 'datacenter2': '3'};
```



Dropping a keyspace

- Removing a keyspace is a simple matter of using the DROP KEYSPACE command:

```
DROP KEYSPACE packt_test;
```



Altering a table

- Tables can be changed with the ALTER statement, using it to add a column:

```
ALTER TABLE query_test ADD c6 TEXT;
```

- Or to remove a column:

```
ALTER TABLE query_test DROP c5;
```



Altering a table

- Table options can also be set or changed with the ALTER statement.
- For example, this statement updates the default TTL on a table to one day (in seconds):

```
ALTER TABLE query_test WITH default_time_to_live =  
86400;
```

Truncating a table

- To remove all data from a table, you can use the TRUNCATE TABLE command:

```
TRUNCATE TABLE query_test;
```



Dropping a table

- Removing a table is a simple matter of using the DROP TABLE command:

```
DROP TABLE query_test;
```



Creating an index

- Next, we'll store the total as a BIGINT representing the number of cents (instead of DECIMAL for dollars), to ensure that we maintain our precision accuracy.
- But the biggest difference is that, in talking with our business resources, we will discover that bucketing by week will give us a partition of manageable size:

Refer to file 3_15.txt

Creating an index

- Next, we will add similar rows into this table:

Refer to file 3_16.txt

- Now I can query for orders placed during the fourth week of July, 2018:

Refer to file 3_17.txt



Creating an index

- This works, but without status as a part of the primary key definition, how can we query for PENDING orders?
- Here is where we will add a secondary index to handle this scenario:

```
CREATE INDEX [index_name] ON  
[keyspace_name.]<table_name>(<column_name>);
```

Creating an index

- In the following code block, we will create an index, and then show how it is used:

Refer to file 3_18.txt



Dropping an index

- Dropping a secondary index on a table is a simple task:

DROP INDEX [index_name]

- If you do not know the name of the index (or created it without a name), you can describe the table to find it.
- Indexes on a table will appear at the bottom of the definition.
- Then you can DROP it:

Refer to file 3_19.txt

Creating a custom data type

- UDTs allow for further denormalization of data within a row.
- A good example of this is a mailing address for customers. Assume a simple table:

```
CREATE TABLE customer (
    last_name TEXT,
    first_name TEXT,
    company TEXT,
    PRIMARY KEY (last_name,first_name));
```



Creating a custom data type

- One way to accomplish this would be to create a collection of a UDT:

```
CREATE TYPE customer_address (
    type TEXT,
    street TEXT,
    city TEXT,
    state TEXT,
    postal_code TEXT,
    country TEXT);
```



Creating a custom data type

- Now, let's add the customer_address UDT to the table as a list.
- This way, a customer can have multiple addresses:

```
ALTER TABLE customer ADD addresses  
LIST<FROZEN <customer_address>>;
```

- With that in place, let's add a few rows to the table:
Refer to file 3_20.txt

Creating a custom data type

- Querying it for Zoey Washburne shows that her company has two addresses:

Refer to file 3_21.txt



Altering a custom type

- UDTs can have columns added to them.
- For instance, some addresses have two lines, so we can add an address2 column:

```
ALTER TYPE customer_address ADD address2 TEXT;
```

- UDT columns can also be renamed with the ALTER command:

```
ALTER TYPE customer_address RENAME address2  
TO street2;
```

Dropping a custom type

- UDTs can be dropped very easily, just by issuing the DROP command. If we create a simple UDT:

```
CREATE TYPE test_type (value TEXT);
```

- It can be dropped like this:

```
DROP TYPE test_type;
```



Creating a user and role

- This creates a new role called cassdba (as seen in lesson 1, Quick Start) and gives it a password and the ability to log in and makes it a superuser:

```
CREATE ROLE cassdba WITH PASSWORD='flynnLives' AND  
LOGIN=true and SUPERUSER=true;
```

- We can also create simple roles:

```
CREATE ROLE data_reader;  
CREATE ROLE data_test;
```



Creating a user and role

- Creating non-superuser roles looks something like this:

```
CREATE ROLE kyle WITH PASSWORD='bacon' AND  
LOGIN=true;
```

```
CREATE ROLE nate WITH PASSWORD='canada' AND  
LOGIN=true;
```

Altering a user and role

- By far, the most common reason for modifying a role is to change the password.
- This was also demonstrated in lesson 1, Quick Start, in showing how to change the default cassandra password:

```
ALTER ROLE cassandra WITH  
PASSWORD='dsfawesomethingdfhdfshdlongandindeci-  
pherabledfdfh';
```

Dropping a user and role

- Removing a user or role is done like this:

```
DROP ROLE data_test;
```



Granting permissions

- Once created, permissions can be granted to roles:

```
GRANT SELECT ON KEYSPACE packt_test TO
```

```
data_reader;
```

```
GRANT MODIFY ON KEYSPACE packt_test TO
```

```
data_reader;
```

Granting permissions

- Roles can also be granted to other roles:

```
GRANT data_reader TO kyle;
```

- More liberal permissions can also be granted:

```
GRANT ALL PERMISSIONS ON KEYSPACE  
packt_test TO kyle;
```

Revoking permissions

- Sometimes a permission granted to a role will need to be removed.
- This can be done with the REVOKE command:

```
REVOKE MODIFY ON KEYSPACE packt_test FROM  
data_reader;
```

COUNT

- CQL allows you to return a count of the number of rows in the result set. Its syntax is quite similar to that of the COUNT aggregate function in SQL.
- This query will return the number of rows in the customer table with the last name Washburne:

```
SELECT COUNT(*) FROM packt_ch3.customer WHERE  
last_name='Washburne';
```

count

2

(1 rows)



COUNT

- The most common usage of this function in SQL was to count the number of rows in a table with an unbound query.
- Apache Cassandra allows you to attempt this, but it does warn you:

```
SELECT COUNT(*) FROM packt_ch3.customer;
```

```
count
```

```
-----
```

```
3
```

```
(1 rows)
```

```
Warnings :
```

```
Aggregation query used without partition key
```



DISTINCT

- CQL has a construct that intrinsically removes duplicate partition key entries from a result set, using the DISTINCT keyword.
- It works in much the same way as its SQL counterpart:

SELECT DISTINCT last_name FROM customer;

last_name

Tam
Washburne
(2 rows)



LIMIT

- CQL allows the use of the LIMIT construct, which enforces a maximum number of rows for the query to return.
- This is done by adding the LIMIT keyword on the end of a query, followed by an integer representing the number of rows to be returned:

```
SELECT * FROM security_logs_by_location LIMIT 1;
```

location_id	day	time_in	employee_id	mailstop
MPLS2	20180723	2018-07-23 11:49:11	samb	M266

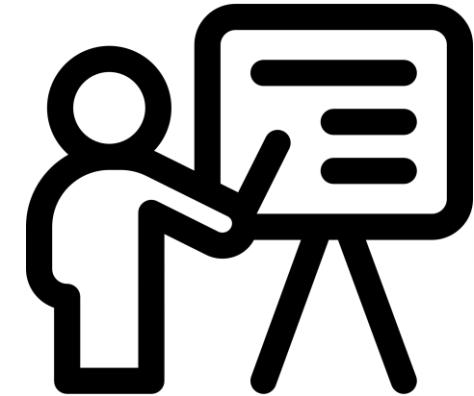
(1 rows)



STATIC

- A new table can be created with a STATIC column like this:

```
CREATE TABLE packt_ch3.fighter_jets (
    type TEXT PRIMARY KEY,
    nickname TEXT STATIC,
    serial_number BIGINT);
```



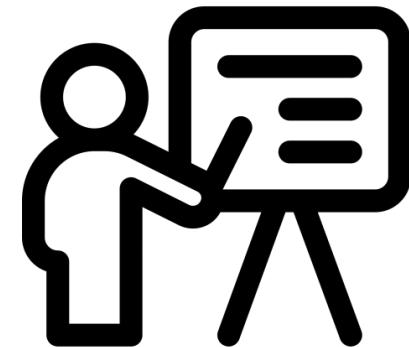
STATIC

- Likewise, an existing table can be altered to contain a STATIC column:

```
ALTER TABLE packt_ch3.users_by_dept ADD  
department_head TEXT STATIC;
```

- Now, we can update data in that column:

Refer to file 3_22.txt



User-defined functions

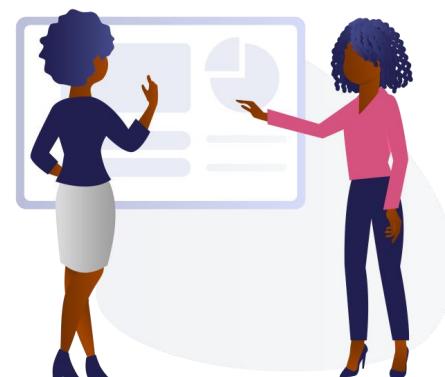


- The date column will return the complete year, month, and day:

```
SELECT todate(now()) FROM system.local;
```

```
system.todate(system.now())
```

```
-----  
2018-08-03  
(1 rows)
```



User-defined functions

- To just get the year back, we could handle that in the application code, or, after enabling user-defined functions in the `cassandra.yaml` file, we could write a small UDF using the Java language:

```
CREATE OR REPLACE FUNCTION year (input DATE)
RETURNS NULL ON NULL INPUT RETURNS TEXT
LANGUAGE java AS 'return input.toString().substring(0,4);';
```

User-defined functions

- Now, re-running the preceding query with `todate(now())` nested inside my new `year()` UDF returns this result:

```
SELECT packt_ch3.year(todate(now())) FROM system.local;
```

```
packt_ch3.year(system.todate(system.now()))
```

```
-----
```

2018

(1 rows)



CONSISTENCY

- By default, cqlsh is set to a consistency level of ONE.
- But it also allows you to specify a custom consistency level, depending on what you are trying to do.
- These different levels can be set with the CONSISTENCY command:

Refer to file 3_23.txt



CONSISTENCY

CONSISTENCY ALL;
Consistency level set to ALL.

SELECT COUNT(*) FROM system_auth.roles;

count

4

(1 rows)

Warnings :

Aggregation query used without partition key



COPY

- The COPY command delivered with cqlsh is a powerful tool that allows you to quickly export and import data.
- Let's assume that I wanted to duplicate my customer table data into another query table. I'll start by creating the new table:

```
CREATE TABLE customer_by_company ( last_name text,  
first_name text,  
addresses list<frozen<customer_address>>,  
company text,  
PRIMARY KEY (company,last_name,first_name));
```



COPY

- Next, I will export the contents of my customer table using the COPY TO command:

Refer to file 3_24.txt

- And finally, I will import that file into a new table using the COPY FROM command:

Refer to file 3_25.txt



DESCRIBE

- DESCRIBE is a command that can be used to show the definition(s) for a particular object.
- Its command structure looks like this:

DESC[RIBE] (KEYSPACE|TABLE|TYPE|INDEX) <object_name>;

- In putting it to use, you can quickly see that it can be used to view things such as full table options, keyspace replication, and index definitions.

DESCRIBE

- Here, we will demonstrate using the DESC command on a table:
DESC TYPE customer_address;

```
CREATE TYPE packt_ch3.customer_address (
```

```
    type text,
```

```
    street text,
```

```
    city text,
```

```
    state text,
```

```
    postal_code text,
```

```
    country text,
```

```
    street2 text
```

```
);
```



DESCRIBE

- Likewise, the DESC command can be used to describe an INDEX:

```
DESC INDEX order_status_idx;
```

```
CREATE INDEX order_status_idx ON  
packt_ch3.order_status_by_week (status);
```



TRACING

- The TRACING command is a toggle that allows the tracing functionality to be turned on:

TRACING ON

Now Tracing is enabled

TRACING

Tracing is currently enabled. Use TRACING OFF to disable





Summary

- Our goals for this lesson included discussing the details behind CQL.
- This includes its syntax and usage and evolution as a language and comparing some of its capabilities to the well-known SQL of the relational database world.
- In addition, I have included tips and notes regarding the application of the functionalities covered, as well as how they can be leveraged in certain situations.

4. Configuring a Cluster



Configuring a Cluster



Specifically, this lesson will cover the following topics:

- Sizing hardware and computer resources for Cassandra deployments
- Operating system optimizations
- Tips and suggestions on orchestration
- Configuring the JVM
- Configuring Cassandra

Evaluating instance requirements

RAM

- Cassandra runs on a JVM, and therefore needs to have a sufficient amount of RAM available.
- The JVM heap size will be the primary consideration, as most of the memory that Cassandra will use will be on-heap.
- The off-heap activity will be largely driven by how much is configured for off-heap memtables (see the following section on configuring Cassandra).
- And of course, the operating system needs to have its share of RAM as well.

Evaluating instance requirements

CPU



- The CPU is still the heart of the instance, and today most have multiple CPU cores.
- While CPU is not typically a bottleneck with Cassandra, having more available processing power can give a boost to heavy operational workloads.



Evaluating instance requirements

Disk

This is the one resource that probably has the most room for variance. Several factors can help to determine the optimal disk size:

- Anticipated size of a single copy of the dataset
- Replication Factor (RF)
- Operational throughput requirements
- Cost of cloud volumes (usually per hour)
- Compaction strategy used on the larger tables
- Whether the size of the dataset will be static, or grow over time
- Whether the application team has an archival strategy



Evaluating instance requirements



Solid state drives



- Disk I/O tends to be the main bottleneck with Cassandra.
- Therefore, if money isn't a barrier, you should always use solid state drives (SSDs).
- Cassandra has configuration properties that can help to take advantage of the increased read times that SSDs can offer.

Evaluating instance requirements

Firewall considerations

Once the network is properly configured, it is possible that you may have to open firewall rules to allow traffic between Cassandra nodes and clients. Remember that Cassandra requires the following ports to be opened for TCP traffic:

- 9042 native binary client protocol
- 7199 JMX
- 7000 internode communication
- 7001 internode communication via SSL



Evaluating instance requirements

Strategy for many small instances versus few large instances

- One aspect to consider when planning Cassandra node instances is to take the approach of many, smaller instances versus fewer, larger instances.
- Each approach has its share of advantages and shortcomings.

Operating system optimizations

- Apache Cassandra has a long-standing presence on Linux-based operating systems (OS), and will run just fine on many flavors of Linux (both RHEL and Debian-based), UNIX, and BSD.
- As of Apache Cassandra 2.2, Windows is now supported as a host operating system.



Operating system optimizations

Disable swap

- Disabling swap is fairly simple, and can be done with this command:

```
sudo swapoff –all
```



- Persisting this change after a reboot is done by editing /etc/fstab and removing the swap entries.

Operating system optimizations

limits.conf

- It is not recommended to have Cassandra running as the root user.
- Part of the deployment process should be to create a cassandra user, which Cassandra should be made to run as.
- These entries should be made to the file:

cassandra - memlock unlimited

cassandra - nofile 100000

cassandra - nproc 32768

cassandra - as unlimited



Operating system optimizations

sysctl.conf

- The following adjustments to /etc/sysctl.conf are recommended.
- For starters, the default kernel value on max map counts will be too low, so adjust that to the following value:

`vm.max_map_count = 1048575`



Operating system optimizations

- These keep connections between nodes, data centers, and client apps from being forcibly disconnected:

```
net.ipv4.tcp_keepalive_time=60  
net.ipv4.tcp_keepalive_probes=3  
net.ipv4.tcp_keepalive_intvl=10
```



Operating system optimizations

- Additionally, adjusting the following settings allows the instance to be better equipped to handle the multitude of concurrent connections required by Cassandra (DataStax, 2018):

net.core.rmem_max=16777216

net.core.wmem_max=16777216

net.core.rmem_default= 6777216

net.core.wmem_default=16777216

net.core.optmem_max=40960

net.ipv4.tcp_rmem=4096 87380 16777216

net.ipv4.tcp_wmem= 096 65536 16777216



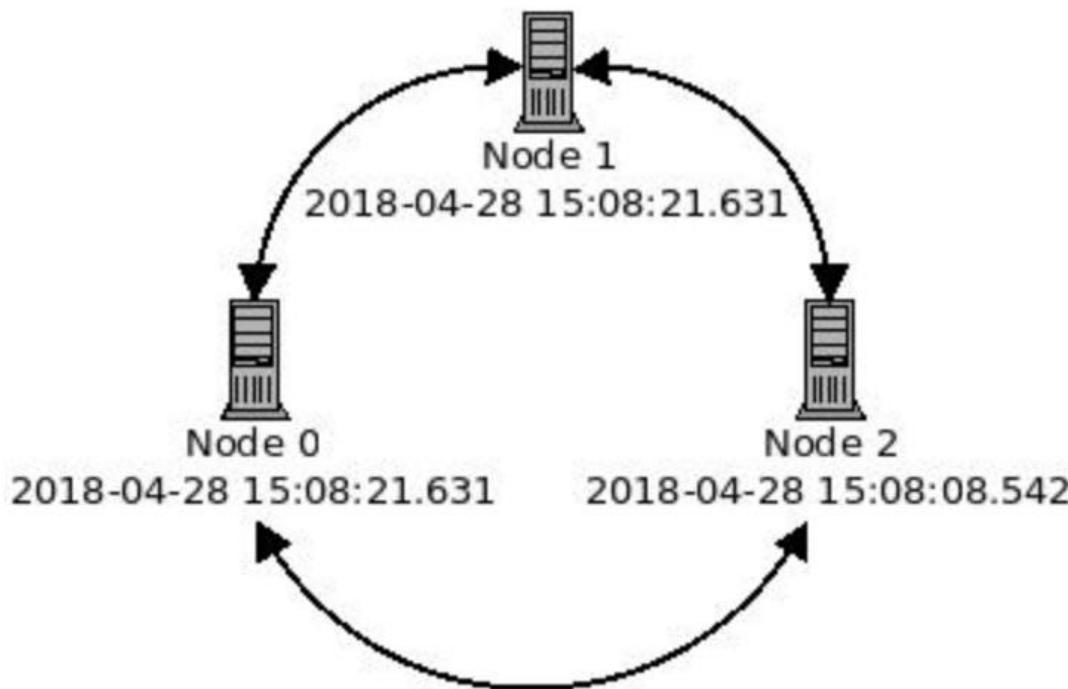
Operating system optimizations

- Restart the instance to make sure these changes are active, or execute the following command:

```
sudo sysctl -p
```



Operating system optimizations



Operating system optimizations

- Let's say I run the following CQL to create a simple table:

```
cassdba@cqlsh:packt> CREATE TABLE keyvalue (key  
TEXT PRIMARY KEY, value TEXT);  
cassdba@cqlsh:packt> INSERT INTO  
keyvalue(key,value) VALUES('timetest','blue');
```

Operating system optimizations

- These requests are handled by Node 1, and, with an RF of three, replicate out to Node 0 and Node 2. Selecting the key, value, and writetime of that value returns this result:

```
cassdba@cqlsh:packt> SELECT key,value,writetime(value)  
FROM keyvalue WHERE key='timetest';
```

key	value	writetime(value)
timetest	blue	1524929184866290



Operating system optimizations

- So far, so good. Now, let's say about 5 seconds later, I run another **INSERT** request:

```
cassdba@cqlsh:packt> INSERT INTO  
keyvalue(key,value) VALUES('timetest','green');
```



Operating system optimizations

- Let's see what happens when we re-run our SELECT query:

```
cassdba@cqlsh:packt> SELECT key,value,writetime(value)  
FROM keyvalue WHERE key='timetest';
```

key	value	writetime(value)
timetest	blue	1524929184866290



Configuring the JVM

- Apache Cassandra was written in Java, and therefore requires a JVM to run.
- Make sure to download a Java Runtime Environment (JRE) or Java Development Kit (JDK) to include as a part of your Cassandra installation.
- Unless otherwise noted, the latest patch of version 8 of the Oracle JDK or OpenJDK should be used.

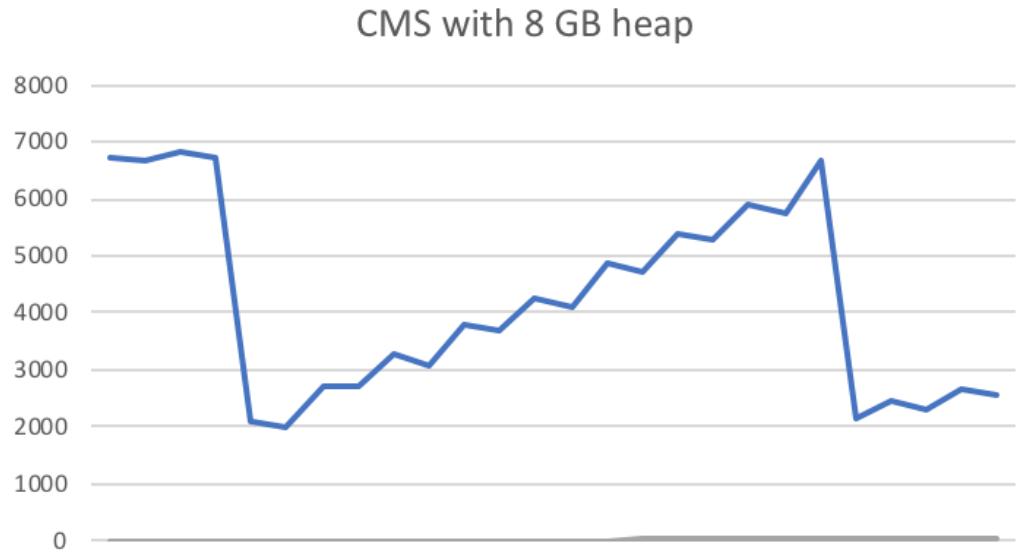
Configuring the JVM

- When certain conditions arise, the garbage collector runs and reclaims heap space by collecting unused objects, and promoting the remaining objects (survivors) to a different generation:

	Survivor Space			
Eden	S0	S1	Tenured Space	Permanent Space
Young Gen		Old Gen		Permanent Gen

Configuring the JVM

- Let's examine a graph which exhibits a common, healthy heap usage pattern with the CMS garbage collector:



Configuring the JVM

- When configuring a CMS heap for use with Apache Cassandra, the `cassandra-env.sh` file will compute the default size of your heap.
- First, the maximum heap size is calculated, based on the following formula:

$$\text{Max Heap Size} = \max\left(\left(\frac{1}{2} \text{ RAM}, \text{max of } 1024 \text{ MB}\right), \left(\frac{1}{4} \text{ RAM}, \text{max of } 8192 \text{ MB}\right)\right)$$

Configuring the JVM

- Similarly, the size of the Young Gen is computed with this formula:

$$\text{Heap New Size} = \min\left(\left(\frac{1}{4} \text{ Max Heap Size}\right), (\text{cpu cores} * 100 \text{ MB})\right)$$


Configuring the JVM

G1GC

- A newer garbage collector innovated in the last few years is the G1GC.
- The G1GC organizes its heap memory differently to the contiguous memory diagram previously seen, used for parallel collectors, such as CMS.



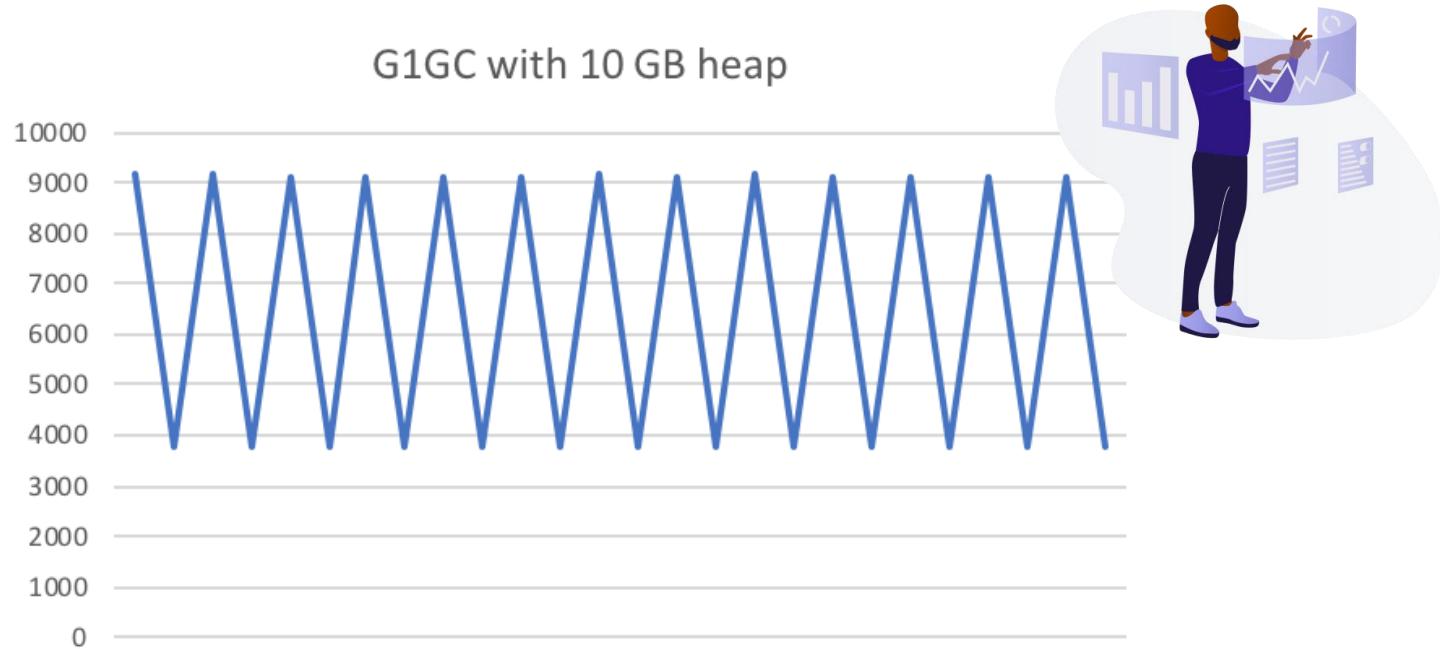
Configuring the JVM

This is demonstrated in
the following diagram:

	E	O		O
O		S	S	
O			O	E
S		E	O	O
	O		O	

Configuring the JVM

- The collection of other regions are then delayed until later:



Configuring the JVM

Garbage collection with Cassandra

- High throughput use cases will almost always generate garbage collection.
- Even clusters with access patterns that tend to be mostly read-serving can see heavy collection activity when their batch loaders run.

Configuring the JVM

Installation of JVM

- With the JDK downloaded, untar it, and move it to a location other than your home directory.
- For the purposes of this example, we'll put it in /usr/local/:

```
sudo tar -zxvf jdk-8u171-linux-x64.tar.gz
```

```
sudo mv jdk1.8.0_171 /usr/local
```

```
sudo alternatives --install /usr/bin/java java  
/usr/local/jdk1.8.0_171/bin/java 1
```

```
sudo alternatives --set java /usr/local/jdk1.8.0_171/bin/java
```



Configuring the JVM

- Alternatively, you can install the JDK via your Linux package management system.
- For Debian Linux, try:

```
sudo apt-get install openjdk-8-jdk
```

- Or for Red Hat Linux:

```
sudo yum install java-1.8.0-openjdk
```



Configuring the JVM

Now test your JDK by executing `java -version`:

`java -version`

openjdk version "1.8.0_171"

OpenJDK Runtime Environment (build 1.8.0_171-b11)

OpenJDK 64-Bit Server VM (build 25.171-b11, mixed mode)



Configuring the JVM

JCE

- Downloading and installing the JCE enables certain additional cipher suites to be used with Apache Cassandra.
- With Ubuntu or another Debian Linux flavor, installing the JCE is as simple as this:

```
sudo apt-get install oracle-java8-unlimited-jce-policy
```

Configuring the JVM

unzip jce_policy-8.zip

Archive: jce_policy-8.zip

creating: UnlimitedJCEPolicyJDK8/

inflating: UnlimitedJCEPolicyJDK8/local_policy.jar

inflating: UnlimitedJCEPolicyJDK8/README.txt

inflating: UnlimitedJCEPolicyJDK8/US_export_policy.jar



Configuring the JVM

- Once the file is unzipped, move the policy files to the jre/lib/security directory of your JDK (\$JAVA_HOME):

```
cd /usr/local/jdk1.8.0_171/jre/lib/security/  
sudo cp ~/UnlimitedJCEPolicyJDK8/US_export_policy.jar .  
sudo cp ~/UnlimitedJCEPolicyJDK8/local_export_policy.jar .
```

Configuring Cassandra

- Configuring a single node for Apache Cassandra is done in a few files located in the conf directory of the instance's Cassandra installation.
- Modification of many of these files can be optional for local instance, development deployments (the defaults should suffice).

Configuring Cassandra

cassandra.yaml

- The `cassandra.yaml` file is the main configuration file for each node in a Cassandra cluster.
- Many of the behaviors of a node can be controlled or influenced from this file.
- While the settings in `cassandra.yaml` are specific to the node on which the file resides, some settings do need to be the same throughout the cluster (these will be noted).

Configuring Cassandra

hints_compression: This indicates the compression class and options to be used for on-disk hint storage.

- The default is none, which means hints will be written uncompressed.
- The available compression classes are LZ4Compressor, SnappyCompressor, and DeflateCompressor.
- Additional properties can also be added, as follows:

hints_compression:

- **class_name: LZ4Compressor**

parameters:

- **chunk_length_kb: 64**
- **crc_check_chance: 1.0**



Configuring Cassandra

data_file_directories: This specifies where on-disk the data for the node's keyspaces and tables should be stored.

- The default location is \$CASSANDRA_HOME/data/data.
- If multiple directories are specified, the data will be spread across them evenly; for example:

data_file_directories:

- /cass_ssd_data0
- /cass_ssd_data1
- /cass_ssd_data2



Configuring Cassandra

commitlog_compression: The commitlog can be compressed by specifying the compression class and options to be used.

- The default is none, which means the commitlog will be uncompressed.
- Available compression classes are LZ4Compressor, SnappyCompressor, and DeflateCompressor, as follows:

commitlog_compression:

- class_name: LZ4Compressor

parameters:

- chunk_length_kb: 64

- crc_check_chance: 1.0



Configuring Cassandra

seeds: A parameter of SimpleSeedProvider, this is where you specify your list of seed nodes (in double quotes, comma-delimited).

- The default value is 127.0.0.1, which is always wrong in a plural node configuration, for example:

seeds: "192.168.0.101,192.168.0.102,192.168.1.21,192.168.1.22"

Configuring Cassandra

cassandra-env.sh

- The `cassandra-env.sh` file is where much of the JVM configuration used to happen, before it was moved to `thejvm.optionsfile` (to be explained in a following section).
- Currently, it is invoked when the node is started to ensure that Cassandra has the environment variables it needs.

Configuring Cassandra

- `-Dcom.sun.management.jmxremote.password.file`: This specifies the location of the password file to use when JMX authentication is activated.
- By default, it is set to `/etc/cassandra/jmxremote.password`.
- Here is an example of a JMX password file:

cassdba	flynnLives
apoletz	c0rr3ctH0rs3B@tt3rySt@pl3



Configuring Cassandra

cassandra-rackdc.properties

- Ec2Snitch
- Ec2MultiRegionSnitch
- GoogleCloudSnitch
- GossipingPropertyFileSnitch

Four properties can be set inside this file: dc, rack, dc_suffix, and prefer_local.



Configuring Cassandra

dc

- This property sets the logical data center for the current node.
- Data center names need to match up with the data center names used to define RFs at keyspace creation time.
- Good dc names can represent physical locations or a cloud region.

For example:

dc=US_Central

Configuring Cassandra

rack

- This property sets the logical rack for the current node.
- These are used by the snitch to help distribute data in ways that support high availability.
- As with dc, good rack names may be physical racks on the data center floor, or they may be representative of availability zones in a public cloud.

For example:

`rack=Central_2A`

Configuring Cassandra

dc_suffix

- This optional property sets a suffix to be added to the dc name for the current node.

For example:

`dc_suffix=_externalFacing`



Configuring Cassandra

prefer_local

- In the event that nodes in a cluster are configured with both internal and external IP addresses, this optional property instructs the snitch to prefer local, internal IP addresses when possible.

For example:

```
prefer_local=true
```



Configuring Cassandra

cassandra-topology.properties



- The `cassandra-topology.properties` file is used with the `PropertyFileSnitch`.
- It is also used as a backup to `GossipingPropertyFileSnitch`.
- It contains a hard-coded list of every node in the cluster, including its IP address, data center, and rack definition.

Configuring Cassandra

- For example:

```
# datacenter US Central  
10.9.9.12=US-Central:central_13  
10.9.19.22=US-Central:central_16  
10.9.29.32=US-Central:central_19  
# datacenter US West  
10.4.4.16=US-West:west_14  
10.4.14.26=US-West:west_2  
10.4.24.36=US-West:west_27
```

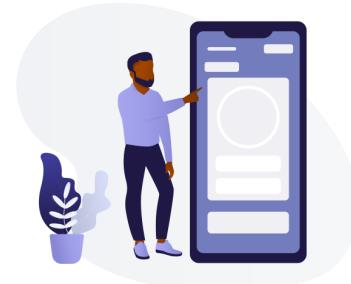


Configuring Cassandra

logback.xml

- The logback.xml file allows for configuration of (some of) Cassandra's logging outputs.
- One of the main sections of this file is the system log appender, as follows:

Refer to file 4_1.txt



Configuring Cassandra

- The location for the logs is controlled by the \${cassandra.logdir} variable.
- By default, this is set to the \$CASSANDRA_HOME/log directory.
- It can be altered by adding the following line to your cassandra-env.sh file:



```
JVM_OPTS="$JVM_OPTS -Dcassandra.logdir=/var/cassandra/log"
```

```
sed -i "s/\${cassandra\.logdir}\//var\log\cassandra/g" logback.xml
```

Managing a deployment pipeline

- When you start working with large, production-level clusters, having a good orchestration tool can save you a lot of work.
- After all, building and configuring a three-node cluster is one thing, but building and maintaining a 300 node cluster requires a different approach.

Configuring Cassandra

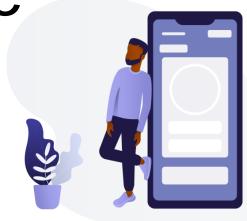
Orchestration tools

- Orchestration tools (Birkman, 2016) are designed to deploy instances.
- Popular choices for orchestration tools are Terraform, Spinnaker, and CloudFormation.
- These types of tools are useful with large-scale Cassandra deployments, as they can allow for quick remediation of availability issues, through the approach of immutable deployments (Birkman 2016).

Configuring Cassandra

Configuration management tools

- Configuration management tools (Birkman, 2016) allow for the installation and management of software on existing instances.
- These configuration changes happen in a mutable fashion.



Configuring Cassandra

Recommended approach

- While some configuration management tools offer some degree of deployment management, some orchestration tools also allow a degree of configuration management.
- But the approach is vastly different.
- While tools like Chef and Ansible alter existing instances, orchestration tools make similar changes by destroying and recreating instances.

Configuring Cassandra



Local repository for downloadable files

- One thing to consider when deploying to a larger scale is where to put the components of your build process.
- You could be relying on one or more websites for the JDK, or even Cassandra itself.



Summary

- There certainly is a lot to consider when planning and building a new Apache Cassandra cluster, and this lesson has put forth a great deal of information.
- We have considered details regarding compute resources, networking, and sizing strategies.
- Linux operating system adjustments to help optimize Apache Cassandra have also been discussed.

5. Performance Tuning





Performance Tuning

These aspects will be the focus of this lesson:

- Using the Cassandra-Stress tool to discover opportunities for improvement
- Looking into situations to apply different table-compaction strategies
- Examining Apache Cassandra's cache and compression options
- Improving upon the efficiency of the JVM
- Optimizing network settings and configuration to avoid performance bottlenecks

Cassandra-Stress

- When examining the performance of your data model, firing up the Cassandra-Stress tool should be your first option.
- Cassandra-Stress comes built-in with Apache Cassandra, and allows you to benchmark your data model with predefined queries for reads, writes, and mixed operational loads.



Cassandra-Stress

The Cassandra-Stress YAML file

- To start using the Cassandra-Stress tool, you must first construct a stress.yaml file. For this example, we will use the security_logs_by_location_id_desc table, which we created in lesson 3, Effective CQL. First, let's create a new keyspace:

```
CREATE KEYSPACE IF NOT EXISTS packt_ch5  
    WITH replication = {'class': 'NetworkTopologyStrategy',  
'ClockworkAngels': '1'}  
    AND durable_writes = true;
```

```
use packt_ch5;
```



Cassandra-Stress

- We'll append a string of _stress to the end of the table's name.
- This way, the _stress tables should only contain randomly-generated stress data, and we ensure that we're keeping our real tables separate:

```
CREATE TABLE packt_ch5.security_logs_by_location_desc_stress (  
    location_id text,  
    day int,  
    time_in timestamp,  
    employee_id text,  
    mailstop text,  
    PRIMARY KEY ((location_id, day), time_in, employee_id)  
) WITH CLUSTERING ORDER BY (time_in DESC, employee_id ASC);
```



Cassandra-Stress

- Let's start by specifying our keyspace and table names:

```
#stress yaml for security_logs_by_location_desc_stress  
#keyspace name  
keyspace: packt_ch5  
#table name  
table: security_logs_by_location_desc_stress
```



Cassandra-Stress

- For reference, let's take a look at a similar table that we built in lesson 3, Effective CQL:

Refer to file 5_1.txt

- Based on this, we'll define a columnspec section as follows:

Refer to file 5_2.txt



Cassandra-Stress

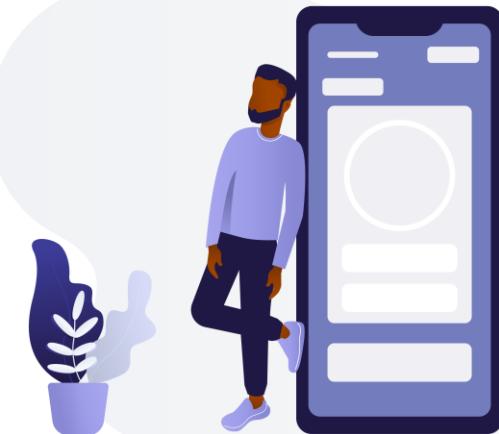
- We'll start by defining our insert behavior to write a single, fixed partition per batch, as well as only one row per partition per batch.
- Our batches will also be unlogged:

insert:

partitions: fixed(1)

batchtype: UNLOGGED

select: fixed(1)/10



Cassandra-Stress

- We will designate that we want our key values to come from the same row (instead of getting location_id and day from multiple rows; combinations that may not exist):

queries:

query1:

cql: SELECT * FROM security_logs_by_location_desc
WHERE location_id=? AND day=?

fields: samerow

Cassandra-Stress



- The following command will invoke our insert definition:

```
tools/bin/cassandra-stress user  
profile=security_logs_by_location_desc_stress.yaml  
ops\insert=1 -node 192.168.0.101 -mode native cql3  
user=cassdba password=flynnLives
```

Cassandra-Stress

- Once that completes, we can re-run that command, but this time referencing our query1 definition in the ops parameter:

```
tools/bin/cassandra-stress user  
profile=security_logs_by_location_desc_stress.yaml  
ops\query1=1 -node 192.168.0.101 -mode native cql3  
user=cassdba password=flynnLives
```

Cassandra-Stress

Cassandra-Stress results

Running with 24 threadCount																			
Running [query1] with 24 threads until stderr of mean < 0.02																			
type	total	ops	op/s	pk/s	row/s	mean	med	.95	.99	.999	max	time	stderr	errors	gc: #	max ms	sum ms		
total,	2342,	2342,	0,	0,	3.0,	0.5,	7.6,	94.2,	252.6,	253.2,	1.0,	NaN,	0,	0,	0,	0,	0,	0,	
total,	4618,	2276,	0,	0,	3.9,	0.8,	16.3,	50.0,	149.6,	152.4,	2.0,	NaN,	0,	1,	7,	7,	7,	7,	
total,	6868,	2250,	2,	60211,	5.5,	0.6,	13.9,	86.0,	337.1,	1289.7,	3.0,	-0.81650,	0,	1,	5,	5,	5,	5,	
total,	8191,	1323,	1,	64980,	5.1,	0.6,	15.4,	70.1,	340.5,	1651.5,	4.0,	-0.55833,	0,	2,	6,	12,	12,	12,	
total,	8602,	411,	0,	0,	4.2,	0.4,	15.5,	92.9,	130.5,	130.5,	5.0,	-0.60145,	0,	1,	6,	6,	6,	6,	
total,	9332,	730,	1,	91065,	16.2,	0.6,	17.7,	144.0,	949.5,	2753.6,	6.0,	-0.51513,	0,	1,	6,	6,	6,	6,	
total,	10210,	878,	1,	43470,	5.4,	0.7,	21.1,	52.9,	341.0,	524.8,	7.0,	-0.86659,	0,	3,	7,	19,	19,	19,	
total,	10840,	630,	3,	307665,	22.3,	0.4,	13.9,	62.7,	4446.0,	7163.9,	8.0,	-0.73133,	0,	2,	7,	13,	13,	13,	
total,	11354,	514,	3,	523258,	33.7,	0.4,	16.4,	250.5,	7335.8,	7524.6,	9.0,	-0.59181,	0,	2,	7,	13,	13,	13,	
total,	11630,	276,	2,	263013,	35.6,	0.4,	11.6,	21.6,	7826.6,	7826.6,	10.0,	-0.53973,	0,	1,	7,	7,	7,	7,	
total,	11637,	7,	2,	60264,	747.0,	32.2,	2787.1,	2787.1,	2787.1,	2787.1,	11.0,	-0.50144,	0,	0,	0,	0,	0,	0,	
total,	12338,	701,	4,	747857,	47.7,	0.4,	13.7,	88.8,	10687.1,	11232.3,	12.0,	-0.43728,	0,	2,	7,	13,	13,	13,	
total,	13158,	820,	2,	259744,	14.8,	0.4,	13.6,	79.5,	655.4,	8824.8,	13.0,	-0.41260,	0,	3,	6,	18,	18,	18,	
total,	13748,	590,	2,	473238,	42.0,	0.6,	19.6,	68.0,	10729.0,	11240.7,	14.0,	-0.38200,	0,	2,	7,	14,	14,	14,	
total,	14458,	710,	3,	629654,	41.9,	0.7,	20.1,	125.1,	7608.5,	11366.6,	15.0,	-0.34881,	0,	1,	13,	13,	13,	13,	
total,	15653,	1195,	4,	534393,	23.4,	0.9,	12.9,	50.9,	9126.8,	13337.9,	16.0,	-0.32804,	0,	1,	10,	10,	10,	10,	
total,	15830,	177,	1,	85956,	8.8,	0.4,	11.2,	19.1,	1057.5,	1057.5,	17.0,	-0.64198,	0,	1,	6,	6,	6,	6,	
total,	16068,	238,	2,	292786,	101.4,	0.5,	51.2,	1750.1,	17314.1,	17314.1,	18.0,	-0.63871,	0,	0,	0,	0,	0,	0,	
total,	17594,	1526,	5,	642187,	24.8,	0.4,	12.1,	64.4,	6828.3,	14931.7,	19.0,	-0.62672,	0,	2,	8,	15,	15,	15,	
total,	19524,	1930,	5,	547430,	16.9,	0.4,	8.2,	106.3,	673.7,	15208.5,	20.0,	-0.61729,	0,	2,	6,	12,	12,	12,	
total,	21171,	1647,	5,	786958,	27.3,	0.5,	12.7,	99.7,	8707.4,	16575.9,	21.0,	-0.60476,	0,	1,	6,	6,	6,	6,	
total,	22522,	1351,	2,	38512,	5.2,	0.6,	15.7,	97.1,	394.0,	766.5,	22.0,	-0.66141,	0,	2,	8,	15,	15,	15,	
total,	23755,	1233,	5,	626066,	24.0,	0.4,	10.6,	75.2,	9814.7,	11232.3,	23.0,	-0.64382,	0,	1,	6,	6,	6,	6,	
total,	25856,	2101,	6,	991906,	21.3,	0.6,	12.8,	68.3,	7851.7,	12239.0,	24.0,	-0.61975,	0,	2,	6,	11,	11,	11,	

Cassandra-Stress

- Next, each iteration will also show a summary of the attempted operations.
- This is a report from a query load run with a thread count of 24:

Refer to file 5_3.txt



Write performance

- Given its log-based storage model and memtable/commitlog design, Apache Cassandra usually does very well when it comes to high-performance data-ingestion.
- That being said, it is not uncommon to hear of application teams complaining about Cassandra nodes being saturated with load due to writes.

Write performance

Commitlog mount point

- One piece of advice, going back to the early days of Apache Cassandra, was to ensure that the commitlog was on a different physical mount point than the data drives.
- This is because disk I/O could become bottlenecked at the device level during periods of heavy writes.
- Putting the commitlog and data directories on separate mount points can alleviate this contention.

Write performance

Scaling out a data center

- It is common for enterprises to have a cluster spread across multiple data centers and infrastructure providers.
- A common pattern is for the application team to load data internally, and replicate it out to the cloud to be served by a customer-facing application.

Write performance

- One solution, would be to create three logical data centers, as shown in the following table:

BigBoxCo Ecommerce cluster:

Data center name	RF	Nodes	Writes/second	Reads/second	CPUs	RAM (GB)
BBDC1	3	6	18,000	0	24	256
Cloud_East	3	18	0	36,000	16	64
Cloud_West	3	18	0	36,000	16	64

Read performance

- If a cluster is exhibiting poor read performance, there are some obvious things to check on the application and data model (table definition) before adjusting any configuration settings.
- As previously mentioned, tables in Cassandra must be designed based on the queries that they are required to serve.

Read performance

Compaction strategy selection

- Apache Cassandra 3.0 ships with three compaction strategies, which offer some control over the frequency and methods for managing the underlying files.
- These compaction strategy classes are `SizeTieredCompactionStrategy` (default), `LeveledCompactionStrategy`, and `TimeWindowCompactionStrategy`.



Read performance

Optimizing read throughput for time-series models

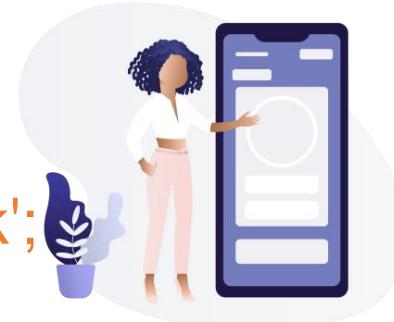
- Since version 2.1, Apache Cassandra has delivered a compaction strategy that helps with optimizing file storage for time-series use cases.
- Originally, DateTieredCompactionStrategy was delivered as a way to optimize data storage by time. With this strategy in place, SSTable files were sorted by relative time.

Read performance

Optimizing tables for read-heavy models

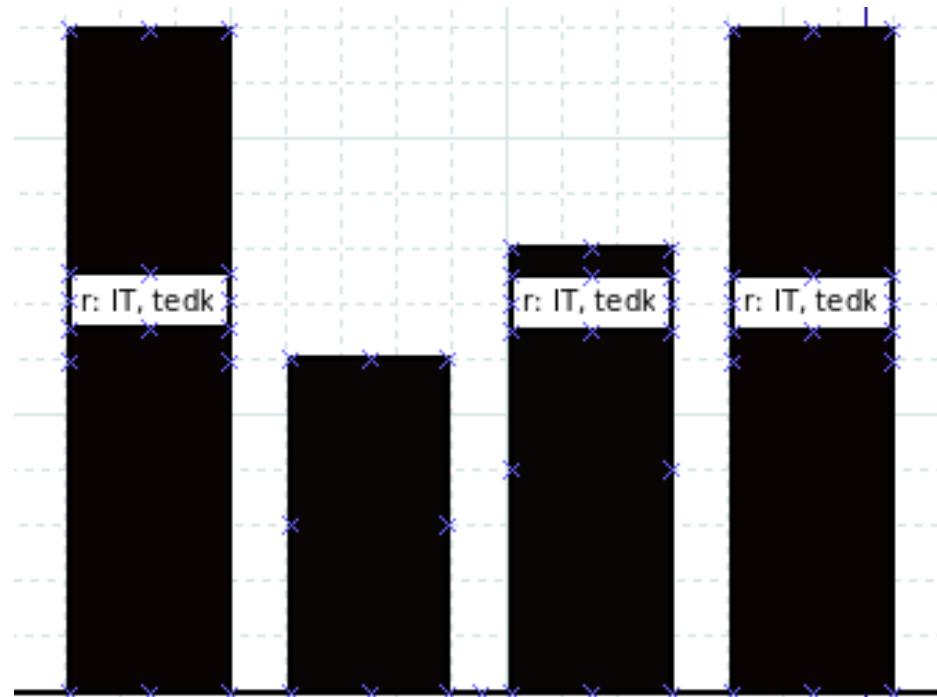
- Let's assume that I have a long-lived table that is updated on a semi-frequent basis, and that I want to query for a specific row:

```
SELECT * FROM users_by_dept  
WHERE department='IT' AND username='tedk';
```



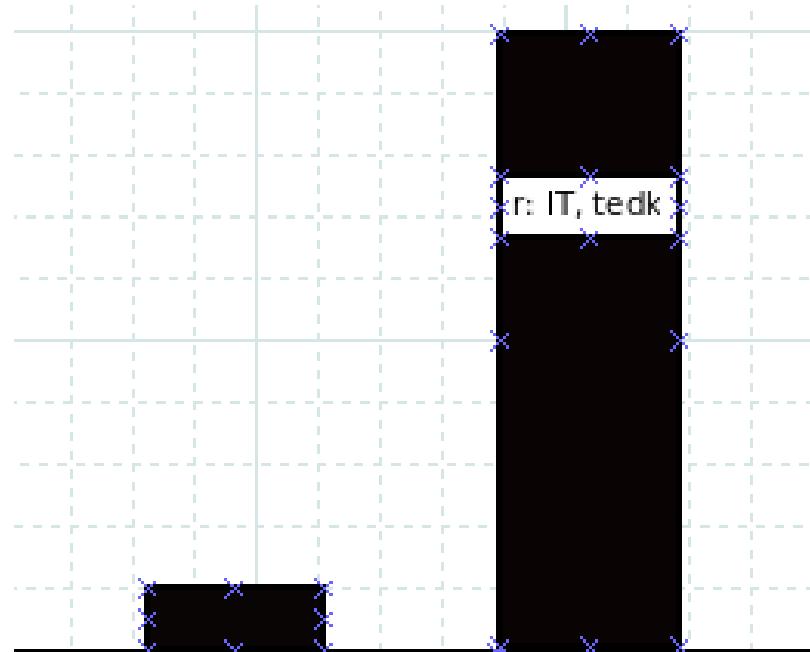
Read performance

- A possible file layout for SSTable files behind the users_by_dept table is depicted here:



Read performance

- But with leveled compaction, each additional level is 10 times the size of the previous.
- Therefore, the mean chance (Hobbs 2012) that a row will be entirely contained within a single SSTable file is 90%:



Read performance

Cache settings

- Apache Cassandra has two mechanisms that can be used to cache table data: the key cache, and the row cache.
- As the name suggests, the key cache simply caches the location of the keys in RAM, providing a boost in efficiency to disk seeks for rows as well (Gilmore 2011).

Read performance

Appropriate uses for row-caching

- As entire rows are stored in RAM with row-caching, it is important to note that some data models are intrinsically better for caching than others.
- Tables modeled with slim or narrow rows (Gilmore 2011) per partition are very efficiently cached.



Read performance

Compression

- As discussed in prior lessons, table-level compression can also be specified.
- Apache Cassandra ships with both the LZ4 and snappy compressor classes, and the LZ4 compressor is used by default.
- Compression can also be disabled on a table, but this is not recommended.



Read performance

Chunk size



- By default, Apache Cassandra implements LZ4 compression with a chunk_length_in_kb size of 64.
- It's important to note that in read-heavy or mixed workloads, using a smaller chunk size can help to improve (reduce) our overall disk I/O usage.

Read performance

The bloom filter configuration

- Another table setting that can help to tune performance is `bloom_filter_fp_chance`.
- Remember from earlier lessons that the bloom filter is a probabilistic data structure.
- Its job in the Cassandra read path is to provide guidance during query time.



Read performance

Read performance issues

As previously indicated, reads can be slow for other reasons:

- Too many tombstones
- Result set is too large
- Partitions are too large



Other performance considerations

- In addition to the performance-tuning options mentioned in the preceding section, there are additional configuration settings that can be adjusted, depending on workload type.
- Also, it is important to understand that there are use cases for which Apache Cassandra is just not a good fit, and is not likely to perform well regardless of performance tuning.

Other performance considerations

Cassandra anti-patterns

- With many distributed databases, there are known anti-patterns, or use cases, that they are just not good at.
- Apache Cassandra definitely has its share of these. Identifying them is important, as you will want to avoid heading down these paths at all costs.



Other performance considerations

Building a queue



Well, what does a queue do? It does the following:

- Data gets written to a queue.
- Data gets updated while it's in the queue (example: status). Sometimes several times.
- When the data is no longer required, it gets deleted.

Other performance considerations

Query flexibility

- As with many distributed databases, the flexibility to work with dynamic query patterns is a key price to pay for not having all of your data in one place.
- While Apache Cassandra does have the ability to apply secondary indexing, they exist for convenience and not for performance.

Other performance considerations

Querying an entire table

- Almost as bad as building a table with multiple secondary indexes are use cases that require multi-key or unbound queries.
- Many developers just need to know exactly how many rows their 200 GB table contains, so they run an unbound query while selecting a count:

```
SELECT COUNT(*) FROM  
some_really_huge_table_that_will_timeout;
```



Other performance considerations

Incorrect use of BATCH

- As a Cassandra DBA, this is a common problem for which you will have to monitor your logs.
- All developers have been taught that batching-up tens of thousands of writes together helps performance; in the RDBMS world, it does.
- Inevitably, a developer will try this on one of your clusters, and could possibly cause the node to crash.

Other performance considerations

Network



- The network is one of those aspects of platform architecture over which Cassandra administrators have little control.
- Unfortunately, all theories of how distributed systems are supposed to work are premised on the fact that the network is an optimally-functioning constant.

Summary



Here are some important takeaways from this lesson:

- There is no amount of tuning that can mitigate a poorly-designed data model.
- Cassandra-Stress is one of the most underrated, yet helpful, tools available for Apache Cassandra.

6. Managing a Cluster



Managing a Cluster

Topics discussed will include the following:

- Adding and removing nodes
- Working with logical data centers
- Backups
- Techniques for ensuring data consistency



Revisiting nodetool

Before we get too far into discussing it, there are some important things to note about nodetool:

- It uses port 7199 for JMX.
- Remote JMX access must be explicitly enabled within each node's configuration.
- Without remote JMX, you will need to SSH into the node to utilize it in nodetool.
- Nodetool is separate from Apache Cassandra. Different versions of nodetool can communicate with different versions of Cassandra, so use it with caution.

Revisiting nodetool

A warning about using nodetool

- As far as Apache Cassandra is concerned, nodetool is an admin-level management tool.
- Make sure that you know what you are doing prior to invoking nodetool commands.
- If care is not taken, serious damage can be done to a cluster with nodetool.

Scaling up

- One common task for Apache Cassandra DBAs is to scale horizontally or scale up (that is, add more nodes to) a cluster.
- Usually, this is because the cluster needs to store additional data, or provide additional operational throughput.



Scaling up

Adding nodes to a cluster

- The basic task behind scaling up a cluster is adding a new node.
- To accomplish this, start with your newly provisioned instance.
- If you follow good DevOps practices, and can install Apache Cassandra simply by executing your deployment pipeline, then this should be very easy for you.

Scaling up

- This can be verified and altered by describing your keyspace using cqlsh to connect to the live node:

DESC KEYSPACE packt;

CREATE KEYSPACE packt WITH replication

```
= {'class': 'NetworkTopologyStrategy', 'ClockworkAngels': '1'};
```

...

ALTER KEYSPACE packt WITH replication

```
= {'class': 'NetworkTopologyStrategy', 'ClockworkAngels': '2'};
```



Scaling up

- Once configuration is complete, start the new node.
- To accomplish this with service-based installations, run the following command:

`sudo service cassandra start`

- Otherwise, you should have a script that runs the following command from the location where you installed Apache Cassandra:

`bin/cassandra -p cassandra.pid`



Scaling up

- Depending on the size of your data, your new node may take a while to fully bootstrap.
- You can monitor the bootstrap process with the following command:

```
nodetool netstats | grep Already  
Already received 35 files, 5085120999 bytes total  
Already received 7 files. 8325238 bytes total  
Already received 18 files. 3105392851 bytes total
```

Scaling up

- During this time, nodetool status queries will show that the new node is UJ, for up and joining.
- In the example shown in the following snippet, 192.168.0.102 is shown joining the cluster:

```
nodetool status
```

```
Datacenter: ClockworkAngels
```

```
=====
```

```
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
```

--	Address	Load	Tokens	Owns	Host ID	Rack
UN	192.168.0.101	114.26 GB	24	100.0%	0edb5efa...	R40
UJ	192.168.0.102	901.62 MB	24	?	38782ca0...	R40



Scaling up

Cleaning up the original nodes

- The original nodes will still have the data that they have streamed but are no longer responsible for.
- In larger clusters, this data can be sizable.
- To ensure that a node only contains data for which it is responsible, you can invoke the nodetool cleanup command:

`nodetool cleanup`

Scaling up

Adding a new data center

- Sometimes, there are reasons to add a new data center to a cluster.
- New data centers provide an easy means by which to migrate or expand your cluster to a new provider.
- Typically, development teams ask for their Cassandra cluster to be built on the same provider, region, or data center as their application runs in.

Scaling up

Adjusting the `cassandra-rackdc.properties` file

- To add a new data center, all of the new nodes will have to share the same dc property.
- An example from the `cassandra-rackdc.properties` file is shown as follows:

```
dc=ClockworkAngels  
rack=R40
```



Scaling up

- For the new data center, I will specify the following for all nodes:

dc=VaporTrails
rack=R30



Scaling up

A warning about SimpleStrategy

- To get data to the nodes in the new data center, we'll need to first modify the keyspace(s) to allow replication to them, as follows:

```
ALTER KEYSPACE packt WITH replication = {'class':  
    'NetworkTopologyStrategy', 'ClockworkAngels': '3',  
    'VaporTrails': '3'};
```

Scaling up

Streaming data

- Streaming data to new nodes in a data center should be done with the nodetool rebuild command.
- This command takes an existing data center as a parameter, which it uses as a source.
- On each new node, run the following command:

`nodetool rebuild -- ClockworkAngels`



Scaling down

- Needing to scale down a cluster isn't a terribly common task.
- But sometimes a cluster uses more resources than it needs, due to application retirement or migration.
- This is especially true in the case of deploying into a public cloud, where extra compute comes with a price tag.

Scaling down

Removing nodes from a cluster

- The most obvious way to trim resources is to remove nodes from the cluster.
- Sometimes, nodes may need to be removed after crashing on their own.
- In a cloud environment built with efficient DevOps processes, it is often much faster to re-provision a troublesome node than it is to spend time resurrecting it.

Scaling down

Removing a live node

- Scaling down by removing a live node (that is, a node which is still functioning) can be accomplished using the nodetool decommission command.
- Decommissioning a live node begins by putting the node in a status of UL, for up and leaving.
- This prevents the node from being included in serving queries by a coordinator.

Scaling down

- The command begins the decommissioning process on the node specified by the host (-h) parameter, or runs on the current node if none is specified:

```
nodetool decommission -h 192.168.0.101
```

- Or we can use as follows:

```
nodetool decommission
```



Scaling down

- If I wanted to remove 192.168.0.102, I could do so with this command:

```
nodetool decommission -h 192.168.0.102
```

```
^C
```

```
nodetool status
```

```
Datacenter: ClockworkAngels
```

```
=====
```

```
Status=Up/Down
```

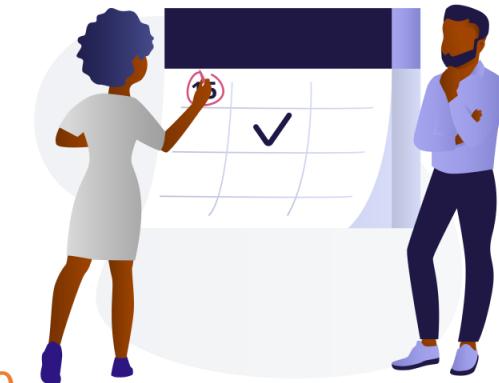
```
|/ State=Normal/Leaving/Joining/Moving
```

```
-- Address Load Tokens Owns Host ID Rack
```

```
UN 192.168.0.101 114.26 GB 24 100.0% 0edb5efa... R40
```

```
UL 192.168.0.102 111.62 GB 24 100.0% 38782ca0... R40
```

```
UN 192.168.0.103 101.81 GB 24 100.0% e45b2ee0... R40
```



Scaling down

Removing a dead node

```
nodetool status
```

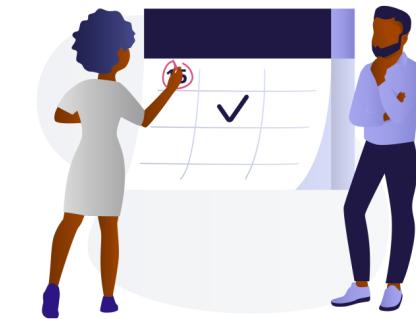
```
Datacenter: ClockworkAngels
```

```
=====
```

```
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
```

	-- Address	Load	Tokens	Owns	Host ID	Rack
UN	192.168.0.101	114.26 GB	24	100.0%	0edb5efa...	R40
DN	192.168.0.102	111.62 GB	24	100.0%	38782ca0...	R40
UN	192.168.0.103	101.81 GB	24	100.0%	e45b2ee0...	R40



Scaling down

- As data cannot be streamed from a down node, nodetool decommission will not work.
- At this point, the best option for removing the node from the cluster will be to remove it.
- The nodetool removenode command exists for this scenario, and works by identifying the node by host ID:

```
nodetool removenode 38782ca0-5dee-4576-b0f9-  
5c54ab6fef6b
```

Scaling down

Other removenode options

- Once you do that, you can query the current node for the status of the removal:

nodetool removenode status

RemovalStatus: Removing token (-7858886458974273909). Waiting for replication confirmation from [192.168.0.103].

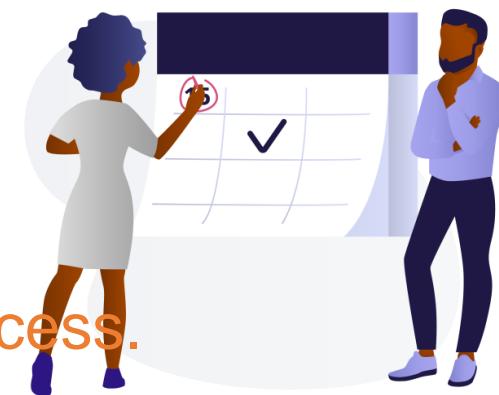
Scaling down

- If this should happen, you can invoke force from the node responsible for the removal as follows:

`nodetool removenode force`

`nodetool removenode status`

RemovalStatus: No token removals in process.



Scaling down

When removenode doesn't work (`nodetool assassinate`)

- Whatever the cause, sometimes a node may appear to be removed to some nodes, but not to others.
- For edge cases like these, the `nodetool assassinate` command exists, and is used as follows:

```
nodetool assassinate 192.168.0.102
```

Scaling down

Assassinating a node on an older version

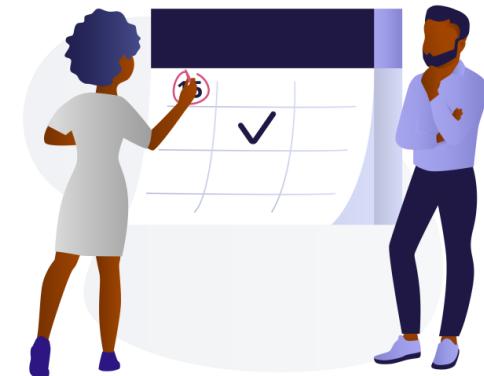
- The nodetool assassinate command is new to Apache Cassandra as of version 2.2.
- Prior to Cassandra 2.2, this operation was still possible, but by invoking a command via JMX.
- For this to work, you will first need to download the JMXTerm JAR file from
<http://wiki.cyclopsgroup.org/jmxterm/>.
- Then, run the JAR file from the command line, connect to your node, and assassinate the endpoint:
Refer to file 6_1.txt

Scaling down

Removing a data center

- First of all, you should identify the name of the target data center and nodes to be removed.
- Consider the following cluster:

Refer to file 6_2.txt



Scaling down

- In this scenario, we will remove the FlyByNight data center from our cluster.
- Let's cqlsh into one of our nodes, and take a look at our keyspace:

```
desc keyspace packt_chapter6 ;
```

```
CREATE KEYSPACE packt_chapter6 WITH replication =  
{'class': 'NetworkTopologyStrategy', 'ClockworkAngels': '3',  
'FlyByNight': '3'} AND durable_writes = true;
```

Scaling down

- Let's start by changing our keyspace definition to no longer replicate to the FlyByNight data center.
- We can accomplish this with the ALTER KEYSPACE command:

```
ALTER KEYSPACE packt_chapter6 WITH replication =  
{'class': 'NetworkTopologyStrategy', 'ClockworkAngels': '3'};
```

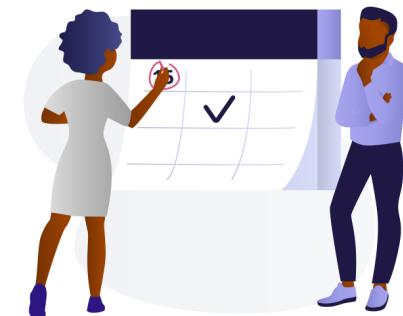
Scaling down

- With the keyspace replication of those nodes disabled for the target data center, you can now SSH to them and run decommission on each of them:

```
aploetz@skywalker:~/local/apache-cassandra-3.11.2$ ssh  
aaron@192.168.17.91
```

...

```
aaron@192.168.17.91: ~$ nodetool decommission  
aaron@192.168.17.91: ~$ exit  
Connection to 192.168.17.91 closed.
```



Backing up and restoring data

- Apache Cassandra provides the ability to take a snapshot of all SSTable files on a node.
- The resulting snapshot files are essentially hard links or pointers to these files.
- Combining snapshots with the incremental backup feature helps to ensure that lost data can be restored in a quick and timely manner.

Backing up and restoring data

Taking snapshots

- To take a snapshot for all tables in a specific keyspace, simply type the name of that keyspace on the end of the command:

```
nodetool snapshot packt_chapter3
```

```
Requested creating snapshot(s) for [packt_chapter3] with snapshot  
name [1538941504326] and options {skipFlush=false}  
Snapshot directory: 1538941504326
```

Backing up and restoring data

- To take a snapshot for a specific table, use the --table option.
- Follow that with the table and keyspace names:

```
nodetool snapshot --table security_logs_by_location  
packt_chapter3
```

Requested creating snapshot(s) for [packt_chapter3] with snapshot name [1538941525101] and options {skipFlush=false}
Snapshot directory: 1538941525101

Backing up and restoring data

Enabling incremental backups

- Incremental backups instruct Apache Cassandra to create links to newly flushed memtables since the last snapshot was taken.
- This has the effect of keeping point-in-time snapshots up-to-date with the latest writes.
- Incremental backups are disabled by default, but can be enabled by editing the following line in the `cassandra.yaml` file of each node:

`incremental_backups: true`

Backing up and restoring data

Recovering from snapshots

- **Run nodetool refresh.** This will load the recently copied snapshot or backup files into the cluster.
- A restart is not required.
- An example of this process is as follows:

Refer to file 6_3.txt



Maintenance

- Let's say that 192.168.0.103 goes down, and I need to replace it.
- I can re-instance a new node, say, 192.168.0.105.
- To ensure that 105 gets the same token ranges as 103 had, I can add this one line to the end of the cassandra-env.sh file at 105:

```
JVM_OPTS="$JVM_OPTS -  
Dcassandra.replace_address=192.168.0.103"
```



Maintenance

Repair

- Simply running this statement without any parameters will invoke a full repair on all keyspaces and tables in the cluster:

`nodetool repair -full`

- As the system keyspaces shouldn't need to be repaired regularly, this command can be focused on repairing all tables in a particular keyspace:

`nodetool repair -full packt_chapter3`

Maintenance

- Likewise, you can also specify a particular table to be repaired as follows:

```
nodetool repair -full packt_chapter3  
security_logs_by_location
```

- If you're in a multi-region cluster, you may want to restrict repairs to certain nodes.
- This helps to avoid heavy cross-region streaming:

```
nodetool -h 192.168.0.101 repair -full packt_chapter3 -hosts  
192.168.0.101 -hosts 192.168.0.102 -hosts 192.168.0.103
```

Maintenance

A warning about incremental repairs

- The idea behind incremental repairs is pretty simple—mark certain data as repaired, and don't bother repairing it until it changes.
- However, incremental repairs are still inherently broken, and there have been consistency issues reported with its use.

Maintenance

Cassandra Reaper

- Within the last year, an open source tool for managing Apache Cassandra repairs has emerged.
- Its name is Cassandra Reaper, and it can be downloaded from <http://cassandra-reaper.io/>.



Maintenance

Forcing read repairs at consistency – ALL

- One quick way to fix consistency on the system_auth tables is to set consistency to ALL, and run an unbound SELECT on every table, tickling each record:

```
use system_auth ;  
consistency ALL;  
consistency level set to ALL.
```

```
SELECT COUNT(*) FROM resource_role_permissions_index ;  
SELECT COUNT(*) FROM role_permissions ;  
SELECT COUNT(*) FROM role_members ;  
SELECT COUNT(*) FROM roles;
```

Maintenance

Snapshots

- Clearing snapshots is a simple matter, and can be done via nodetool:

`nodetool clearsnapshot`

- Running `clearsnapshot` without any parameters instructs Apache Cassandra to remove all snapshot files on a node.
- It can also be focused for a particular keyspace, like the following:

`nodetool clearsnapshot -- packt_chapter3`

Maintenance

- Additionally, `clearsnapshot` can be focused even further, by specifying a snapshot by name:

```
nodetool clearsnapshot -t 1538941504326
```



Maintenance

Adjusting compaction throughput due to available resources

- A common way to remedy this problem is to lower the compaction throughput, as follows:

```
nodetool getcompactionthroughput  
Current compaction throughput: 256 MB/s
```



Maintenance

- Woah! Allowing compaction to consume 256 Mbps (megabits per second) is a sure way to saturate your instance's resources.
- Let's lower that, and monitor our node's progress:

```
nodetool setcompactionthroughput 8  
nodetool getcompactionthroughput  
Current compaction throughput: 8 MB/s
```





Summary

- This lesson covered a broad range of subjects related to the Apache Cassandra node tool utility.
- The commands covered here will help you to scale your cluster horizontally, as well as to remove and replace failed nodes.
- If you support a cluster that has nodes deployed in public or private cloud instances, you will find that it is often much faster to replace a failed node, than to try and resurrect it.

7. Monitoring



Monitoring

Specifically, this lesson will cover the following topics:

- The Java Management Extension (JMX) interface:
JConsole, JMXTerm
- Nodetool
- Metrics: JMXTrans, Telegraf, InfluxDB, Grafana,
Alerting
- Logging: Filebeat, Kibana
- Troubleshooting



JMX interface

- JMX is a Java technology that provides tools for managing and monitoring applications, services, components, devices, and settings.
- These settings are represented as Java objects, known as Managed Beans (MBeans).
- As Apache Cassandra is written in Java, all the required classes have exposed MBeans, through which we can manage/monitor them accordingly.

JMX interface

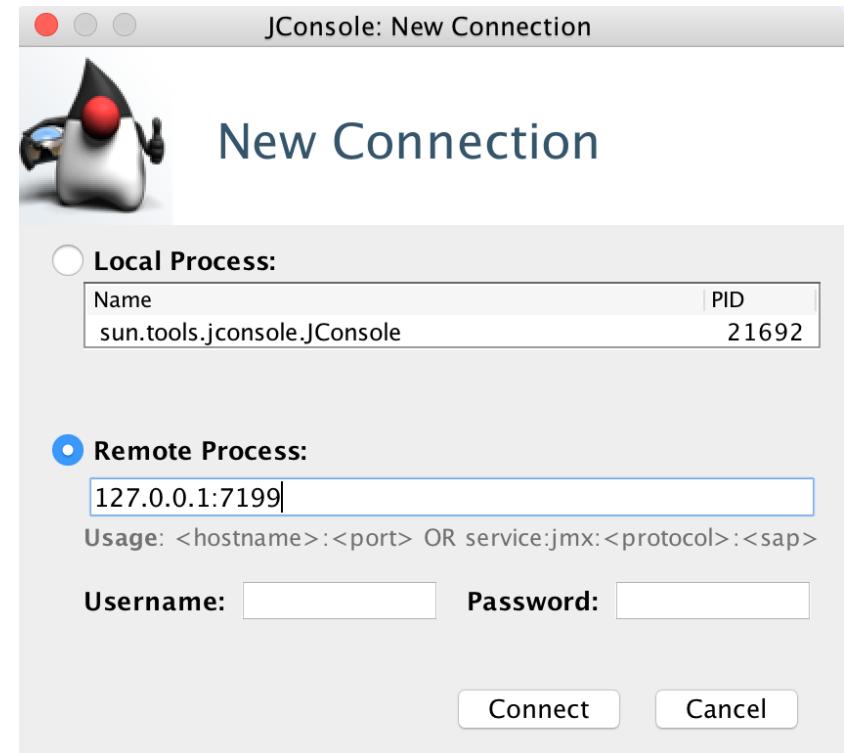
JConsole (GUI)

- JConsole is a built-in utility for the JDK, which gives a graphical view of resource utilization for a Java process.
- It also provides an interface to view metrics and perform operations to manage applications.
- Refer to the JConsole docs for further informationat

JMX interface

Connection and overview

Figure: JConsole connection



JMX interface

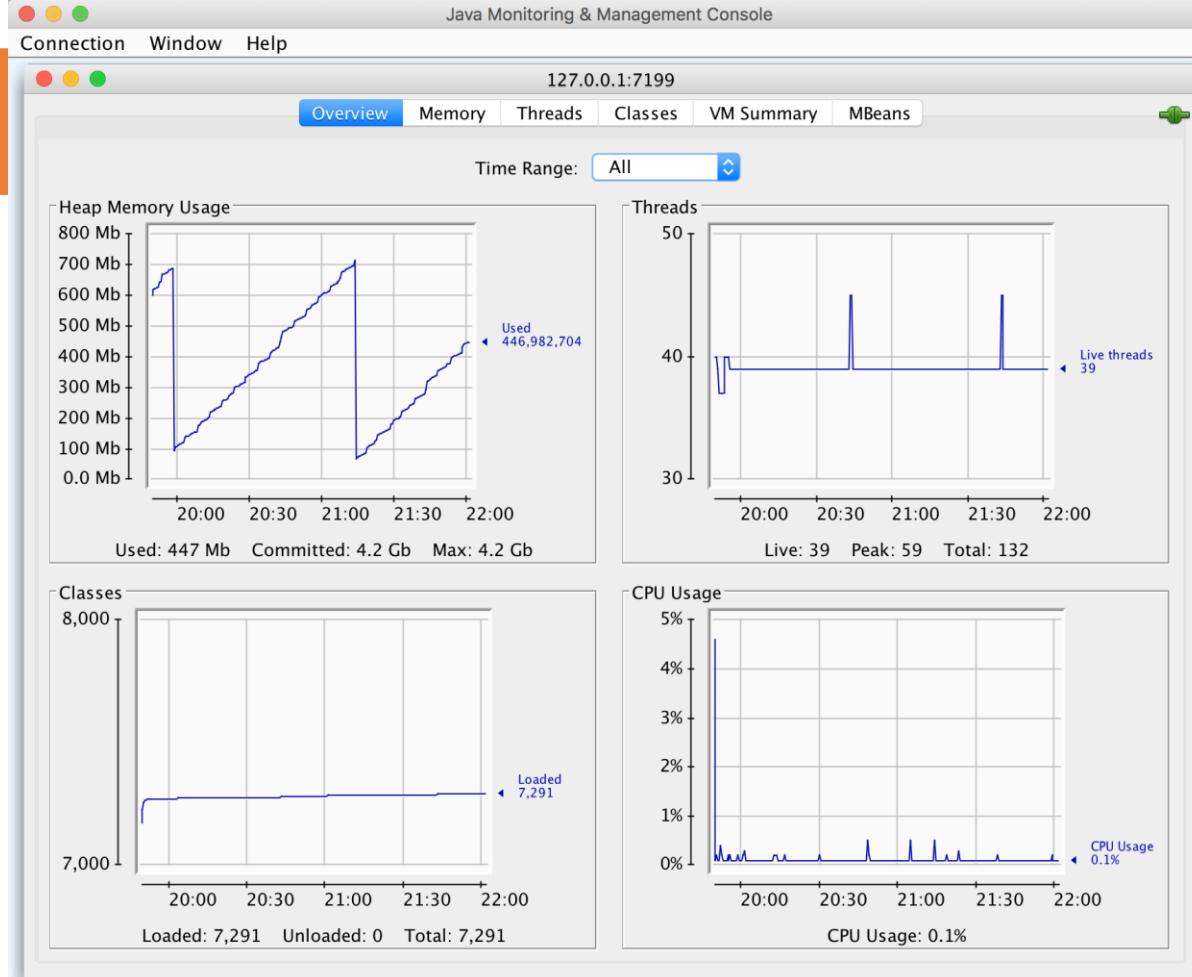
- In certain implementations, a node would have both internal and external IPs.
- These would be set in the Cassandra.yaml config, with the listen_address and rpc_address parameters set to internal IPs, and the broadcast_address and broadcast_rpc_address parameters set to external IPs.
- Additionally, some additional parameters need to be updated in Cassandra-env.sh as follows:

Refer to file 7_1.txt



JMX interface

The following shows an of JConsole:



JMX interface

Viewing metrics

Java Monitoring & Management Console
Connection Window Help
127.0.0.1:7199
Overview Memory Threads Classes VM Summary MBeans

Attribute value

Name	Value
DownEndpointCount	0

Refresh

MBeanAttributeInfo

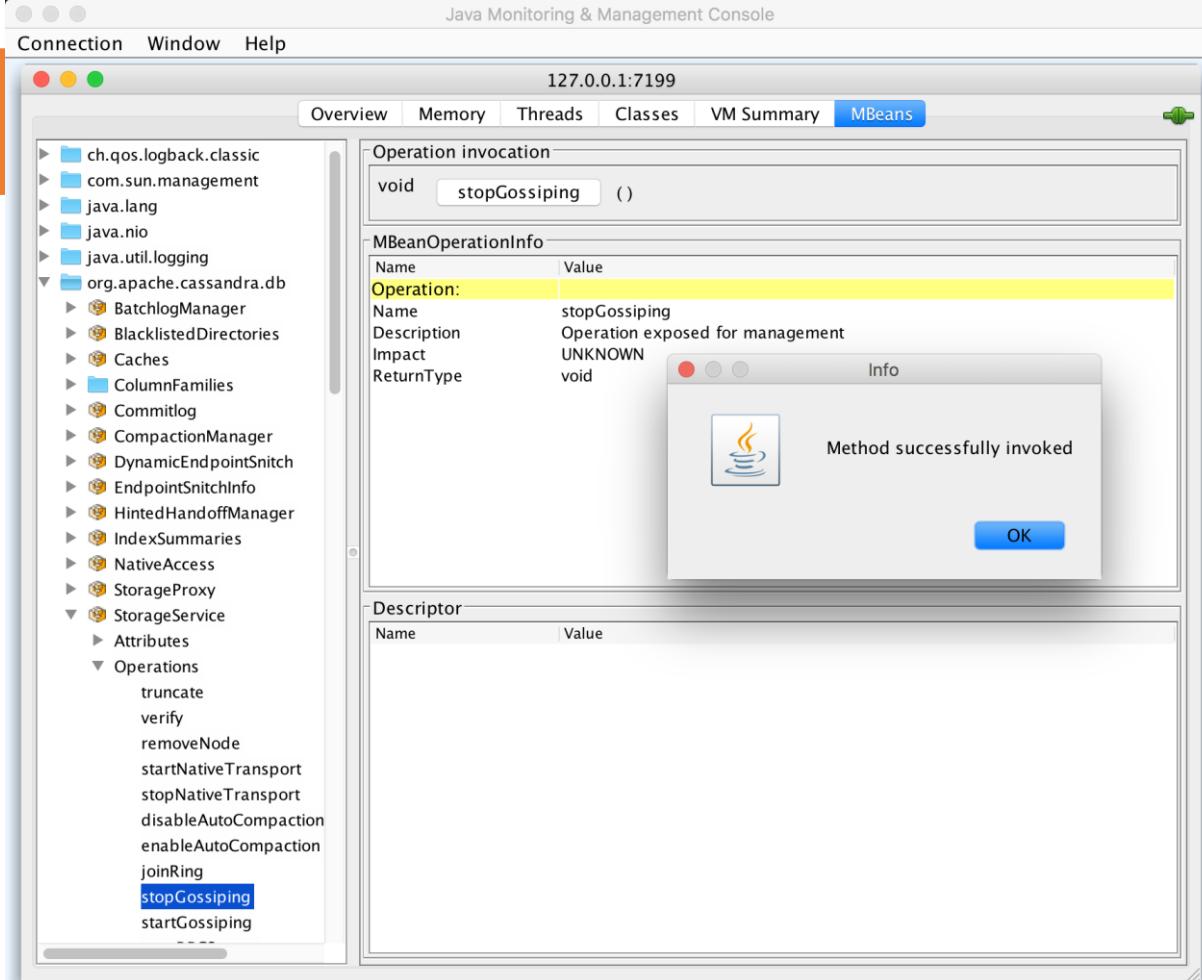
Name	Value
Attribute:	DownEndpointCount
Name	DownEndpointCount
Description	Attribute exposed for management
Readable	true
Writable	false
Is	false
Type	int

Descriptor

Name	Value

JMX interface

Which is your most-used operation for Cassandra?



JMX interface

Connection and domains

- Download the JMXTerm jar from <http://wiki.cyclopsgroup.org/jmxterm/download.html>, then invoke JMXTerm from the CLI.
- Once you run the following command, Java just provides an interface to interact through JMX.
- This allows you to create a connection to the desired IP and port

JMX interface

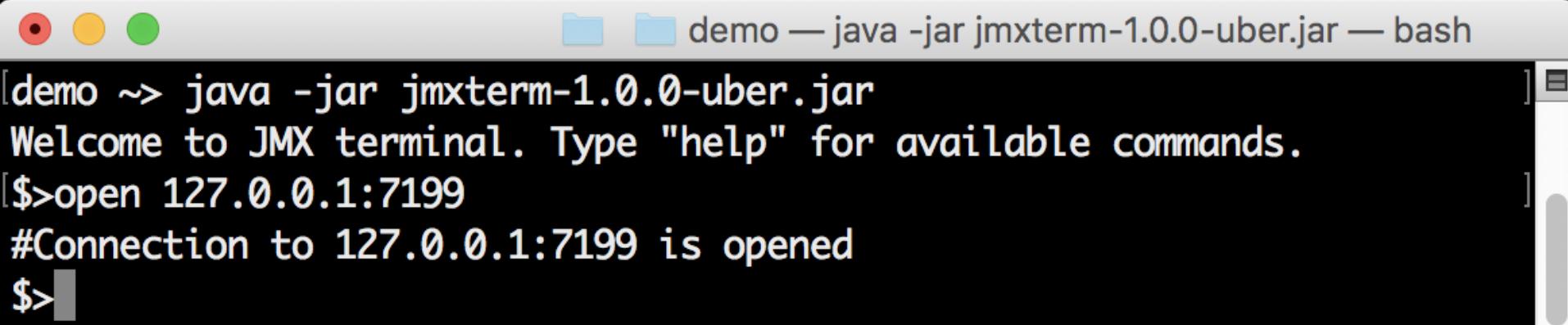
- If Cassandra is running on my local, 127.0.0.1:7199 can be used where JMX authentication and SSL are disabled on Cassandra:

```
java -jar <absolutePath>/jmxterm-1.0.0-uber.jar  
open <IP>:7199
```



JMX interface

- The preceding command will give the following output:



A screenshot of a macOS terminal window. The window title bar shows the path: demo — java -jar jmxterm-1.0.0-uber.jar — bash. The terminal itself displays the following text:

```
[demo ~] java -jar jmxterm-1.0.0-uber.jar
Welcome to JMX terminal. Type "help" for available commands.
[$>open 127.0.0.1:7199
#Connection to 127.0.0.1:7199 is opened
$>
```

JMX interface

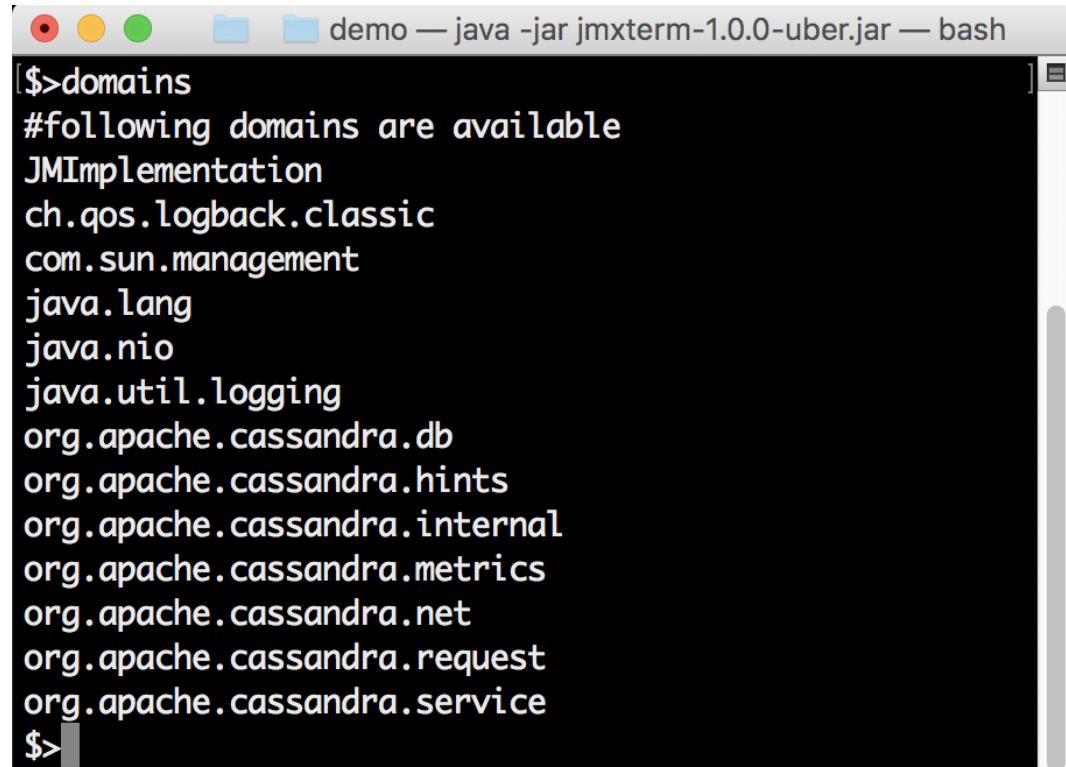
- The domains command gives all the MBean packages exposed by Cassandra, along with any Java-related packages exposed by default for any Java process:

domains



JMX interface

The output of the domains command is as follows:



A screenshot of a terminal window titled "demo — java -jar jmxterm-1.0.0-uber.jar — bash". The window shows the output of the "\$>domains" command. The output lists various JMX domain names, including "JMImplementation", "ch.qos.logback.classic", "com.sun.management", "java.lang", "java.nio", "java.util.logging", "org.apache.cassandra.db", "org.apache.cassandra.hints", "org.apache.cassandra.internal", "org.apache.cassandra.metrics", "org.apache.cassandra.net", "org.apache.cassandra.request", and "org.apache.cassandra.service". The terminal prompt "\$>" is visible at the bottom.

```
$>domains
#following domains are available
JMImplementation
ch.qos.logback.classic
com.sun.management
java.lang
java.nio
java.util.logging
org.apache.cassandra.db
org.apache.cassandra.hints
org.apache.cassandra.internal
org.apache.cassandra.metrics
org.apache.cassandra.net
org.apache.cassandra.request
org.apache.cassandra.service
$>
```

JMX interface

Getting a metric

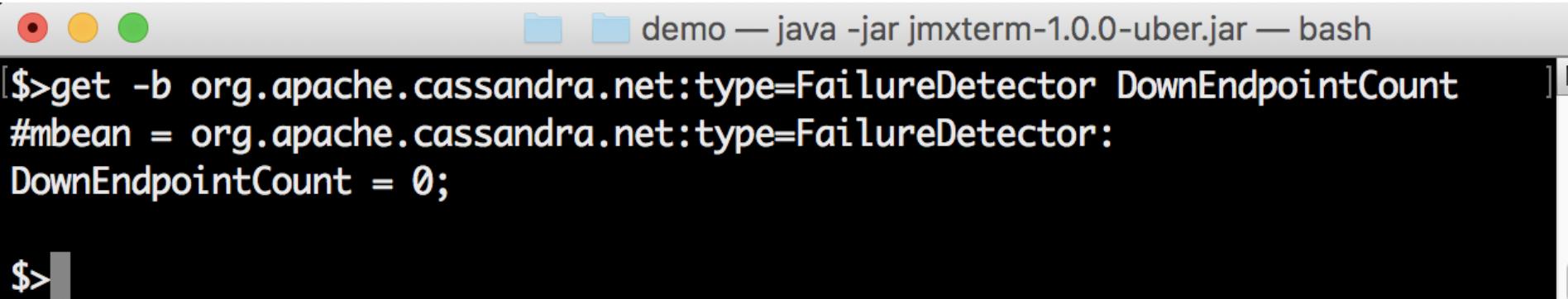
- Which is your most-used metric in Cassandra?

```
get -b org.apache.cassandra.net:type=FailureDetector  
DownEndpointCount
```



JMX interface

- The output of the preceding command is as follows:



A screenshot of a terminal window on a Mac OS X desktop. The window title bar shows the path: demo — java -jar jmxterm-1.0.0-uber.jar — bash. The terminal itself displays the following text:

```
$>get -b org.apache.cassandra.net:type=FailureDetector DownEndpointCount  
#mbean = org.apache.cassandra.net:type=FailureDetector:  
DownEndpointCount = 0;  
$>
```

JMX interface

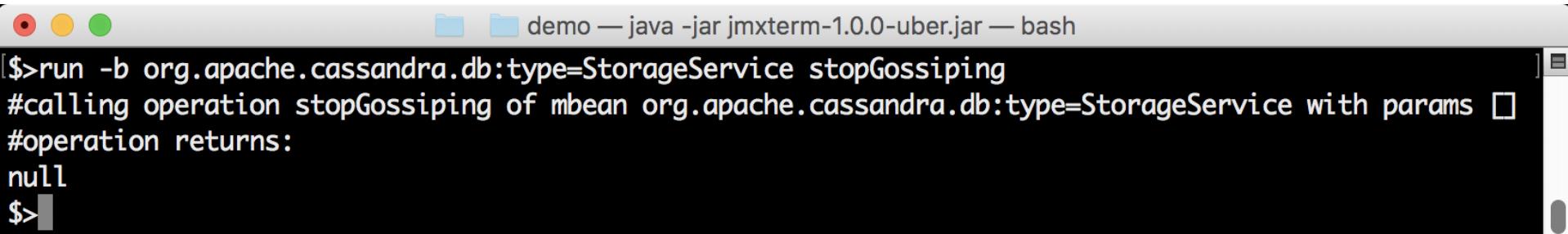
Performing an operation

- Performing an operation is also possible through JMXTerm.
- For example, you can invoke thestopGossiping operation under the org.apache.Cassandra.db package using the following run command.
- This stops gossiping to other nodes. Which is your favorite operation on Cassandra?

```
run -b org.apache.cassandra.db:type=StorageService  
stopGossiping
```

JMX interface

- The output of the preceding command is as follows:



A screenshot of a terminal window on a Mac OS X desktop. The window title bar shows the path: demo — java -jar jmxterm-1.0.0-uber.jar — bash. The terminal itself contains the following text:

```
[${>}run -b org.apache.cassandra.db:type=StorageService stopGossiping
#calling operation stopGossiping of mbean org.apache.cassandra.db:type=StorageService with params []
#operation returns:
null
$>]
```

JMX interface

- Instead of the interactive CLI, you can run a list of commands using a file, as follows:

```
echo-e"open <IP>:7199\nrun -b  
org.apache.cassandra.db:type=StorageService  
forceTerminateAllRepairSessions\nclose">jmxcommands  
java -jar<absolutePath>/jmxtterm-1.0.0.-uber.jar -v silent -n  
< jmxcommands
```

The nodetool utility

- Running a nodetool command remotely is very helpful as it wouldn't require SSH.
- But at the same time, it would be more prone to a security risk if endpoints are wide open to the public.
- The remote command can be triggered with the -h parameter through nodetool:

```
$CASSANDRA_HOME/bin/nodetool -h <ipAddress of host> ring
```



The nodetool utility

Monitoring using nodetool

- Monitoring Apache Cassandra clusters using nodetool has multiple parameters that can be passed based on the use case through the CLI.
- This is very good for checking on a single node, but when there are more nodes, it's hard to keep track of this setup across all nodes, as it is a tedious process.

The nodetool utility

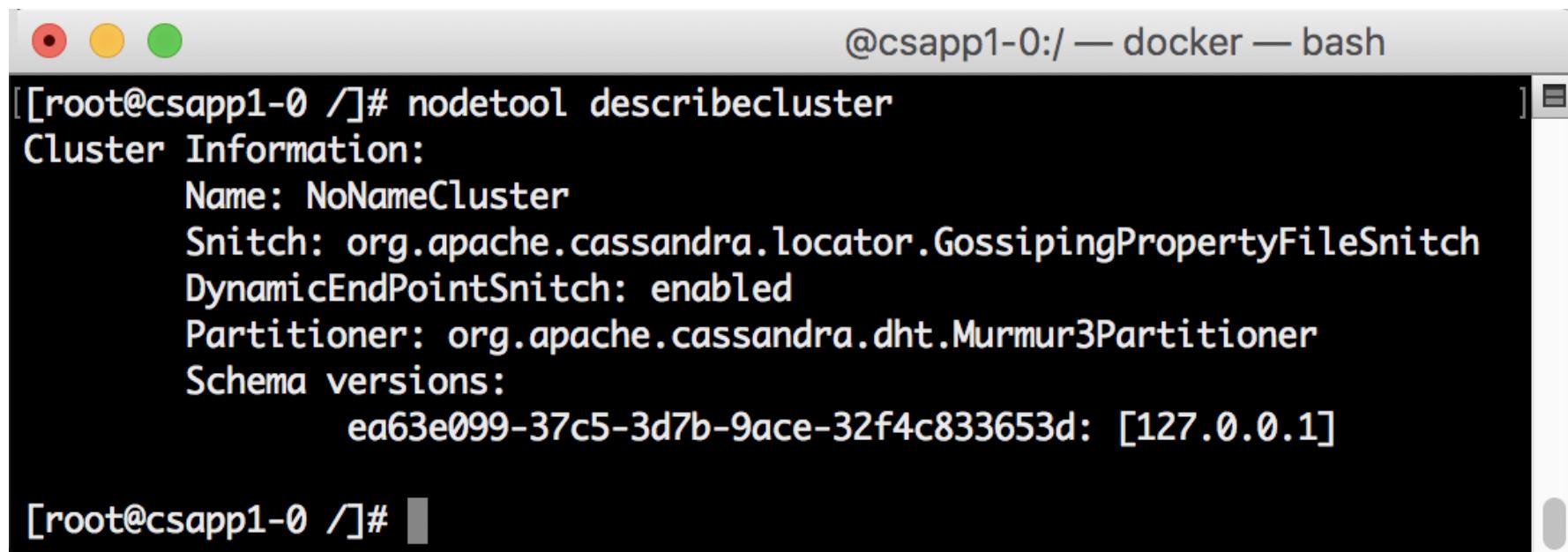
describecluster

- As name suggests, it gives details about name of the cluster, the snitching type, the partitioner type, schema version, and the dynamic endpoint snitch flag.
- The schema version will display all the nodes along with nodes that have any schema version mismatches between nodes, resulting in an easier way of identifying faulty nodes:

```
$CASSANDRA_HOME/bin/nodetool describecluster
```

The nodetool utility

- The output of the preceding command is as follows:



```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool describecluster
Cluster Information:
  Name: NoNameCluster
  Snitch: org.apache.cassandra.locator.GossipingPropertyFileSnitch
  DynamicEndPointSnitch: enabled
  Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
  Schema versions:
    ea63e099-37c5-3d7b-9ace-32f4c833653d: [127.0.0.1]
[root@csapp1-0 /]#
```

The nodetool utility

Gcstats

- This displays key garbage-collection statistics for the Apache Cassandra JVM and those stats are related to garbage collection, which have run since the last time the nodetool gcstats command was executed:

```
$CASSANDRA_HOME/bin/nodetool gcstats
```



The nodetool utility

- The output of the preceding command is as follows:

```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool gcstats
      Interval (ms) Max GC Elapsed (ms)Total GC Elapsed (ms)Stdev GC Elapsed (ms)  GC Reclaimed (MB)
Collections          Direct Memory Bytes
      2547667                  14           14                 0           28835928
      1                      -1
[root@csapp1-0 /]#
```

The nodetool utility

getcompactionthreshold

- This provides minimum and maximum thresholds for compaction on a table in a keyspace.
- This can be configured, but would require a restart of Cassandra for that config to be reflected.
- This can also be increased using setcompactionthreshold, but it would be reset when Cassandra is restarted.
- But it can be used as a quick fix during issues with nodes or while bootstrapping a node:

```
$CASSANDRA_HOME/bin/nodetool getcompactionthreshold  
<keyspaceName> <tableName>
```

The nodetool utility

- The output is as follows:

```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool getcompactionthreshold system_auth roles
Current compaction thresholds for system_auth/roles:
min = 4, max = 32
[root@csapp1-0 /]#
```

The nodetool utility

getcompactionthroughput

- This provides the current throughput for all compactions, irrespective of the table and keyspace.
- This can be configured but would require a restart of Cassandra for that config to be reflected.
- This can also be increased using `setcompactionthroughput`, but it would be reset when Cassandra is restarted, so it can be used as a quick fix during issues with nodes or while bootstrapping a node:

```
$CASSANDRA_HOME/bin/nodetool  
getcompactionthroughput
```



The nodetool utility

- The output is as follows:



```
@csapp1-0:/ — docker — bash
[[root@csapp1-0 ~]# nodetool getcompactionthroughput
Current compaction throughput: 16 MB/s
[[root@csapp1-0 ~]#]
```

The nodetool utility

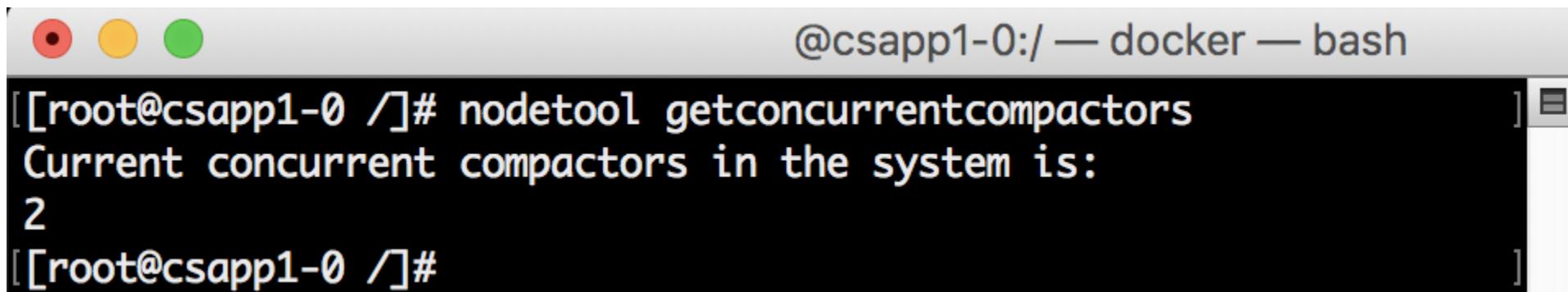
Getconcurrentcompactors

- This provides the number of possible concurrent compactions for Cassandra.
- This can be configured, but would require a restart of Cassandra for that config to be reflected.
- This can also be increased using `setconcurrentcompactors`, but it will also be reset when Cassandra is restarted.
- It can be used as a quick fix during issues with nodes or while bootstrapping a node:

```
$CASSANDRA_HOME/bin/nodetool getconcurrentcompactors
```

The nodetool utility

- The output of the preceding command is as follows:



The screenshot shows a terminal window with three colored status indicators (red, yellow, green) at the top left. The title bar reads '@csapp1-0:/ — docker — bash'. The terminal content is as follows:

```
[root@csapp1-0 ~]# nodetool getconcurrentcompactors
Current concurrent compactors in the system is:
2
[root@csapp1-0 ~]#
```

The nodetool utility

Getendpoints

- This provides all the endpoints owned by a partition key for a table in a keyspace across the Cassandra cluster.
- During inconsistencies across a Cassandra cluster, it provides an easier way to find a faulty node for random partitions by checking the LOCAL_ONEconsistency on all those replicas, and resolve the issue by running a repair only on that node or datacenter:

```
$CASSANDRA_HOME/bin/nodetool getendpoints  
<keyspaceName> <tableName> <partitionKey>
```



The nodetool utility

- The output of the preceding command is as follows:

```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool getendpoints system_auth roles cassandra
127.0.0.1
[root@csapp1-0 /]#
```

The nodetool utility

getloginglevels

- This provides all the logging levels set for the current Cassandra node.
- This can be configured, but would require a restart of Cassandra for that config to be reflected.
- It can also be increased using `setloginglevel`, but it would be reset when Cassandra is restarted, so it can be used as a quick fix during issues with nodes or while troubleshooting a node:

```
$CASSANDRA_HOME/bin/nodetool getloginglevels
```

The nodetool utility

- The output of the preceding code is as follows:

```
@csapp1-0:/ — docker — bash
[ [root@csapp1-0 /]# nodetool getlogginglevels
Logger Name                                     Log Level
ROOT                                         INFO
com.thinkaurelius.thrift                      ERROR
org.apache.cassandra                           DEBUG
[root@csapp1-0 /]# ]
```

The nodetool utility

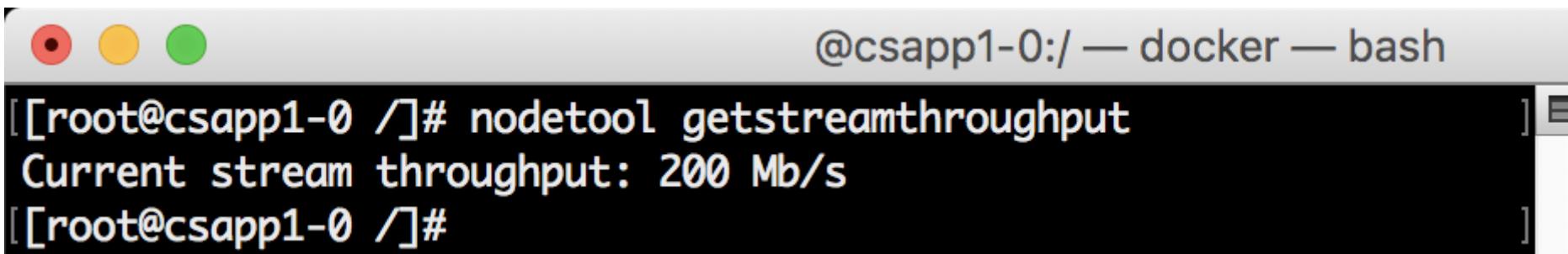
getstreamthroughput

- This provides the Mbps (megabits per second) limit for any streaming on the current Cassandra node.
- This can be configured, but would require a restart of Cassandra for that config to be reflected.
- It can also be increased using setsteamthroughput, but it would be reset when Cassandra is restarted, so it can be used as a quick fix during issues with node or while bootstrapping a node:

```
$CASSANDRA_HOME/bin/nodetool getstreamthroughput
```

The nodetool utility

- The output of the preceding command is as follows:



```
@csapp1-0:/ — docker — bash
[[root@csapp1-0 ~]# nodetool getstreamthroughput
Current stream throughput: 200 Mb/s
[[root@csapp1-0 ~]#]
```

The nodetool utility

gettimeout

- This provides the timeout in milliseconds for read, range, write, counterwrite, cascontention, truncate, steamingsocket, and rpc on the current Cassandra node.
- This can be configured, but would require a restart of Cassandra for that config to be reflected.
- This can also be increased using settimeout, but it would be reset when Cassandra is restarted.
- However, it can be used as a quick fix during issues with nodes or while bootstrapping a node:

```
$CASSANDRA_HOME/bin/nodetool gettimeout <timeoutType>
```

The nodetool utility

- The output of the preceding code is as follows:

```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool gettimeout
nodetool: gettimeout requires a timeout type, one of (read, range, write, counterwrite, casc
ontention, truncate, streamingsocket, misc (general rpc_timeout_in_ms))
See 'nodetool help' or 'nodetool help <command>'.
[root@csapp1-0 /]# nodetool gettimeout read
Current timeout for type read: 5000 ms
[root@csapp1-0 /]#
```

The nodetool utility

Gossipinfo

- This provides all the gossip information for a Cassandra node, and contains the status, load, schema version, dc, rack, Cassandra version, listen, and RPC addresses:

```
$CASSANDRA_HOME/bin/nodetool gossipinfo
```



The nodetool utility

- The output of the preceding code is as follows:

```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool gossipinfo
/127.0.0.1
generation:1537802399
heartbeat:4276
STATUS:20:NORMAL,-1809464237749553735
LOAD:4242:130137.0
SCHEMA:23:ea63e099-37c5-3d7b-9ace-32f4c833653d
DC:8:dc1
RACK:10:rack1
RELEASE_VERSION:4:3.11.2
INTERNAL_IP:6:127.0.0.1
RPC_ADDRESS:3:127.0.0.1
NET_VERSION:1:11
HOST_ID:2:dee2a308-1946-40f6-b22b-60effdb9d99c
RPC_READY:25:true
TOKENS:19:<hidden>

[root@csapp1-0 /]#
```

The nodetool utility

Info

- This provides all the internal information of a Cassandra node, and contains ID, gossip, thrift, native transport, and other statuses, along with key and row-cache details:

```
$CASSANDRA_HOME/bin/nodetool info
```



The nodetool utility

- The output of the preceding code is as follows:

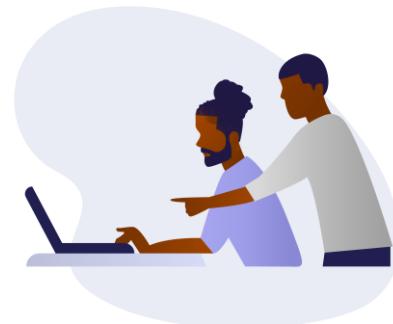
```
@csapp1-0:/ — docker — bash
[root@csapp1-0 /]# nodetool info
ID : 61bc005d-be87-45ab-8180-488b32007e3d
Gossip active : true
Thrift active : true
Native Transport active: true
Load : 99.28 KiB
Generation No : 1537770324
Uptime (seconds) : 2582
Heap Memory (MB) : 271.34 / 1000.00
Off Heap Memory (MB) : 0.00
Data Center : dc1
Rack : rack1
Exceptions : 0
Key Cache : entries 14, size 1.08 KiB, capacity 50 MiB, 78 hits, 98 requests, 0.796 recent hit rate, 14400 save period in seconds
Row Cache : entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0 requests, NaN recent hit rate, 0 save period in seconds
Counter Cache : entries 0, size 0 bytes, capacity 25 MiB, 0 hits, 0 requests, NaN recent hit rate, 7200 save period in seconds
Chunk Cache : entries 15, size 960 KiB, capacity 218 MiB, 27 misses, 145 requests, 0.814 recent hit rate, NaN microseconds miss latency
Percent Repaired : 100.0%
Token : (invoke with -T/--tokens to see all 16 tokens)
[root@csapp1-0 /]#
```

The nodetool utility

Netstats

- This provides all network information for a Cassandra node, and contains Mode, Large messages, Small messages, Gossip messages that are Active, Pending, Completed, and Dropped:

```
$CASSANDRA_HOME/bin/nodetool netstats
```



The nodetool utility

- The output is as follows:

```
[root@csapp1-0 /]# nodetool netstats
Mode: NORMAL
Not sending any streams.
Read Repair Statistics:
Attempted: 0
Mismatch (Blocking): 0
Mismatch (Background): 0
Pool Name          Active  Pending  Completed  Dropped
Large messages      n/a      0          0          0
Small messages      n/a      0          2          0
Gossip messages     n/a      0          0          0
[root@csapp1-0 /]#
```