



DataStax Distribution of Apache Cassandra™ 3.x Documentation

February 4, 2016

Apache, Apache Cassandra, Apache Hadoop, Hadoop and the eye logo are trademarks of the Apache Software Foundation

© 2016 DataStax, Inc. All rights reserved.

Contents

About DataStax Distribution of Apache Cassandra.....	8
What's new?.....	9
Understanding the architecture.....	11
Architecture in brief.....	11
Internode communications (gossip).....	13
Failure detection and recovery.....	14
Data distribution and replication.....	14
Consistent hashing.....	14
Virtual nodes.....	16
Data replication.....	17
Partitioners.....	18
Murmur3Partitioner.....	19
RandomPartitioner.....	19
ByteOrderedPartitioner.....	19
Snitches.....	20
Dynamic snitching.....	20
SimpleSnitch.....	20
RackInferringSnitch.....	20
PropertyFileSnitch.....	21
GossipingPropertyFileSnitch.....	22
Ec2Snitch.....	22
Ec2MultiRegionSnitch.....	23
GoogleCloudSnitch.....	24
CloudstackSnitch.....	25
Database internals.....	25
Storage engine.....	25
How Cassandra reads and writes data.....	26
How is data written?.....	26
How is data maintained?.....	27
How is data updated?.....	30
How is data deleted?.....	30
How are indexes stored and updated?.....	31
How is data read?.....	31
How do write patterns affect reads?.....	34
Data consistency.....	34
How are consistent read and write operations handled?.....	34
How are Cassandra transactions different from RDBMS transactions?.....	36
How do I accomplish lightweight transactions with linearizable consistency?.....	37
How do I discover consistency level performance?.....	38
How is the consistency level configured?.....	38
How is the serial consistency level configured?.....	41
How are read requests accomplished?.....	41
How are write requests accomplished?.....	51

Planning a cluster deployment.....	54
Selecting hardware for enterprise implementations.....	54
Planning an Amazon EC2 cluster.....	56
Calculating partition size.....	58
Calculating usable disk capacity.....	59
Calculating user data size.....	59
Anti-patterns in Cassandra.....	60
Installing.....	63
Installing the DataStax Distribution of Apache Cassandra 3.x on RHEL-based systems.....	63
Installing DataStax Distribution of Apache Cassandra 3.x on Debian-based systems.....	64
Installing from the binary tarball.....	65
Installing earlier releases of DataStax Distribution of Apache Cassandra 3.x.....	66
Uninstalling the DataStax Distribution of Apache Cassandra 3.x.....	67
Installing on cloud providers.....	68
Installing the JDK.....	68
Installing Oracle JDK on RHEL-based Systems.....	69
Installing Oracle JDK on Debian or Ubuntu Systems.....	70
Installing OpenJDK on RHEL-based Systems.....	70
Installing OpenJDK on Debian-based Systems.....	71
Recommended production settings for Linux.....	71
Install locations.....	74
Tarball installation directories.....	74
Package installation directories.....	75
Configuration.....	76
cassandra.yaml configuration file.....	76
Cassandra include file.....	94
Security.....	94
Securing Cassandra.....	94
SSL encryption.....	95
Internal authentication.....	99
Internal authorization.....	101
Configuring firewall port access.....	102
Enabling JMX authentication.....	103
Configuring gossip settings.....	104
Configuring the heap dump directory.....	105
Configuring virtual nodes.....	106
Enabling virtual nodes on a new cluster.....	106
Enabling virtual nodes on an existing production cluster.....	106
Using multiple network interfaces.....	107
Configuring logging.....	109
Commit log archive configuration.....	110
Generating tokens.....	111
Hadoop support.....	112
Initializing a cluster.....	114
Initializing a multiple node cluster (single data center).....	114
Initializing a multiple node cluster (multiple data centers).....	116
Starting and stopping Cassandra.....	119
Starting Cassandra as a service.....	119
Starting Cassandra as a stand-alone process.....	119

Stopping Cassandra as a service.....	120
Stopping Cassandra as a stand-alone process.....	120
Clearing the data as a service.....	120
Clearing the data as a stand-alone process.....	120
Operations.....	121
Adding or removing nodes, data centers, or clusters.....	121
Adding nodes to an existing cluster.....	121
Adding a data center to a cluster.....	122
Replacing a dead node or dead seed node.....	124
Replacing a running node.....	125
Moving a node from one rack to another.....	126
Decommissioning a data center.....	126
Removing a node.....	127
Switching snitches.....	127
Changing keyspace strategy.....	129
Edge cases for transitioning or migrating a cluster.....	129
Adding or replacing single-token nodes.....	130
Backing up and restoring data.....	132
About snapshots.....	132
Taking a snapshot.....	132
Deleting snapshot files.....	133
Enabling incremental backups.....	133
Restoring from a snapshot.....	133
Restoring a snapshot into a new cluster.....	135
Recovering from a single disk failure using JBOD.....	136
Repairing nodes.....	137
Hinted Handoff: repair during write path.....	137
Read Repair: repair during read path.....	140
Manual repair: Anti-entropy repair.....	140
Migrating to incremental repairs.....	145
Monitoring Cassandra.....	146
Monitoring a Cassandra cluster.....	146
Tuning Java resources.....	150
Data caching.....	152
Configuring data caches.....	152
Monitoring and adjusting caching.....	154
Configuring memtable throughput.....	154
Configuring compaction.....	155
Compression.....	156
When to compress data.....	156
Configuring compression.....	157
Testing compaction and compression.....	157
Tuning Bloom filters.....	158
Moving data to or from other databases.....	159
Purging gossip state on a node.....	159
Cassandra tools.....	160
The nodetool utility.....	160
assassinate.....	161
bootstrap.....	162
cfhistograms.....	163
cfstats.....	163
cleanup.....	163

clearsnapshot.....	164
compact.....	165
compactionhistory.....	166
compactionstats.....	168
decommission.....	169
describecluster.....	169
describing.....	170
disableautocompaction.....	171
disablebackup.....	172
disablebinary.....	172
disablegossip.....	173
disablehandoff.....	173
disablehintsfordc.....	174
disablethrift.....	175
drain.....	176
enableautocompaction.....	176
enablebackup.....	177
enablebinary.....	177
enablegossip.....	178
enablehandoff.....	178
enablehintsfordc.....	179
enablethrift.....	180
flush.....	180
gcstats.....	181
getcompactionthreshold.....	182
getcompactionthroughput.....	182
getendpoints.....	183
getlogginglevels.....	184
getsstables.....	184
getstreamthroughput.....	185
gettraceprobability.....	185
gossipinfo.....	186
help.....	186
info.....	187
invalidatecountercache.....	188
invalidatekeycache.....	189
invalidaterowcache.....	189
join.....	190
listsnapshots.....	190
move.....	191
netstats.....	192
pausehandoff.....	193
proxyhistograms.....	193
rangekeysampl.....	194
rebuild.....	194
rebuild_index.....	195
refresh.....	196
reloadtriggers.....	196
removenode.....	197
repair.....	198
replaybatchlog.....	201
resetlocalschema.....	202
resumehandoff.....	203
ring.....	203
scrub.....	204
setcachecapacity.....	205

setcachekeystosave.....	206
setcompactionthreshold.....	206
setcompactionthroughput.....	207
sethintedhandoffthrottlekb.....	208
setlogginglevel.....	208
setstreamthroughput.....	209
settraceprobability.....	210
snapshot.....	211
status.....	213
statusbackup.....	214
statusbinary.....	215
statusgossip.....	215
statushandoff.....	216
statusthrift.....	216
stop.....	217
stopdaemon.....	218
tablehistograms.....	218
tablestats.....	219
toppartitions.....	225
tpstats.....	226
truncatehints.....	230
upgradesstables.....	231
verify.....	231
version.....	232
The cassandra utility.....	233
The cassandra-stress tool.....	235
Using the Daemon Mode.....	242
Interpreting the output of cassandra-stress.....	243
SSTable utilities.....	244
sstableexpiredblockers.....	244
sstablekeys.....	245
sstablelevelreset.....	245
sstableloader (Cassandra bulk loader).....	246
sstablemetadata.....	248
sstableofflinerelevel.....	251
sstablerepairedset.....	252
sstablescrub.....	253
sstablesplit.....	253
sstableupgrade.....	254
sstableutil.....	254
sstableverify.....	255
Troubleshooting.....	255
Peculiar Linux kernel performance problem on NUMA systems.....	255
Reads are getting slower while writes are still fast.....	256
Nodes seem to freeze after some period of time.....	256
Nodes are dying with OOM errors.....	256
Nodetool or JMX connections failing on remote nodes.....	257
Handling schema disagreements.....	257
View of ring differs between some nodes.....	257
Java reports an error saying there are too many open files.....	257
Insufficient user resource limits errors.....	258
Cannot initialize class org.xerial.snappy.Snappy.....	259
Lost communication due to firewall timeouts.....	260

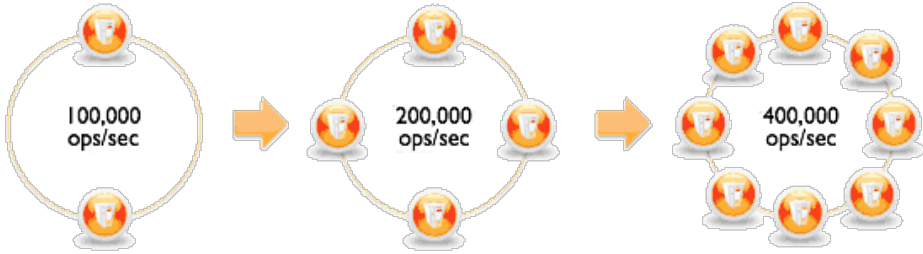
Release notes.....	260
Using the docs.....	261

About DataStax Distribution of Apache Cassandra

Apache Cassandra™ is a massively scalable open source NoSQL database. Cassandra is perfect for managing large amounts of structured, semi-structured, and unstructured data across multiple data centers and the cloud. Cassandra delivers continuous availability, linear scalability, and operational simplicity across many commodity servers with no single point of failure, along with a powerful dynamic data model designed for maximum flexibility and fast response times.

How does Cassandra work?

Cassandra's built-for-scale architecture means that it is capable of handling petabytes of information and thousands of concurrent users/operations per second.

Cassandra is a partitioned row store database	Cassandra's architecture allows any authorized user to connect to any node in any data center and access data using the CQL language. For ease of use, CQL uses a similar syntax to SQL. The most basic way to interact with Cassandra is using the CQL shell, cqlsh . Using cqlsh , you can create keyspaces and tables, insert and query tables, plus much more. If you prefer a graphical tool, you can use DataStax DevCenter . For production, DataStax supplies a number drivers so that CQL statements can be passed from client to cluster and back.
Automatic data distribution	Cassandra provides automatic data distribution across all nodes that participate in a <i>ring</i> or database cluster. There is nothing programmatic that a developer or administrator needs to do or code to distribute data across a cluster because data is transparently partitioned across all nodes in a cluster.
Built-in and customizable replication	Cassandra also provides built-in and customizable replication, which stores redundant copies of data across nodes that participate in a Cassandra ring. This means that if any node in a cluster goes down, one or more copies of that node's data is available on other machines in the cluster. Replication can be configured to work across one data center, many data centers, and multiple cloud availability zones.
Cassandra supplies linear scalability	Cassandra supplies linear scalability, meaning that capacity may be easily added simply by adding new nodes online. For example, if 2 nodes can handle 100,000 transactions per second, 4 nodes will support 200,000 transactions/sec and 8 nodes will tackle 400,000 transactions/sec:
	

How is Cassandra different from relational databases?

Cassandra is designed from the ground up as a distributed database with peer-to-peer communication. As a best practice, queries should be one per table. Data is denormalized to make this possible. For this

reason, the concept of JOINS between tables does not exist, although client-side joins can be used in applications.

What is NoSQL?

Most common translation is "Not only SQL", meaning a database that uses a method of storage different from a relational, or SQL, database. There are many different types of NoSQL databases, so a direct comparison of even the most used types is not useful. Database administrators today must be polyglot-friendly, meaning they must know how to work with many different RDBMS and NoSQL databases.

What is CQL?

[Cassandra Query Language \(CQL\)](#) is the primary interface into the Cassandra DBMS. Using CQL is similar to using SQL (Structured Query Language). CQL and SQL share the same abstract idea of a table constructed of columns and rows. The main difference from SQL is that Cassandra does not support joins or subqueries. Instead, Cassandra emphasizes denormalization through CQL features like collections and clustering specified at the schema level.

CQL is the recommended way to interact with Cassandra. Performance and the simplicity of reading and using CQL is an advantage of modern Cassandra over older Cassandra APIs.

The [CQL documentation](#) contains a [data modeling topic](#), examples, and command reference.

How do I interact with Cassandra?

The most basic way to interact with Cassandra is using the CQL shell, `cqlsh`. [Using cqlsh](#), you can create keyspaces and tables, insert and query tables, plus much more. If you prefer a graphical tool, you can use [DevCenter](#). For production, DataStax supplies a number of [drivers](#) in various programming languages, so that CQL statements can be passed from client to cluster and back.

How can I move data to/from Cassandra?

Data is inserted using the CQL INSERT command, the CQL COPY command and CSV files, or [sstableloader](#). But in reality, you need to consider how your client application will query the tables, and do data modeling first. The paradigm shift between relational and NoSQL means that a straight move of data from an RDBMS database to Cassandra will be doomed to failure.

What other tools come with Cassandra?

Cassandra automatically installs [nodetool](#), a useful command-line management tool for Cassandra. A tool for load-stressing and basic benchmarking, [cassandra-stress](#), is also installed by default.

What kind of hardware/cloud environment do I need to run Cassandra?

Cassandra is designed to run on [commodity hardware](#) with common specifications. In the [cloud](#), Cassandra is adapted for most common offerings.

What's new in DataStax Distribution of Apache Cassandra 3.x

Note: Cassandra is now releasing on a tick-tock schedule. For more information, see [Cassandra 2.2, 3.0, and beyond](#).

The latest version of DataStax Distribution of Apache Cassandra 3.x is 3.2.

The [CHANGES.txt](#) describes the changes in detail. You can view all version changes by branch or tag in the drop-down list on the changes page.

New features Cassandra 3.2

-graph option for cassandra-stress	<code>cassandra-stress</code> results can be automatically graphed for data visualization.
TTL for COPY FROM	A TTL value can be specified when copying from CSV files.
bulkloader can use third party authentication	The bulkloader has an option <code>-ap</code> for third party authentication.
CREATE TABLE WITH ID	If a table is accidentally dropped, it can be recreated with its ID and the commitlog replayed to regain data.

New features released in Cassandra 3.0

Storage engine refactored	The Storage Engine has been refactored.
Materialized Views	Materialized views handle automated server-side denormalization, with consistency between base and view data.
Support for Windows	Support for Windows 7, Windows 8, Windows Server 2008, and Windows Server 2012. See DataStax Cassandra 2.2 Windows Documentation .
Operations improvements	
Addition of MAX_WINDOW_SIZE_SECONDS to DTCS compaction settings	Allow DTCS compaction governance based on maximum window size rather than Table age.
File-based Hint Storage and Improved Replay	Hints are now stored in files and replay is improved.
Default garbage collector is changed to G1	Default garbage collector is changed from Concurrent-Mark-Sweep (CMS) to G1. G1 performance is better for nodes with heap size of 4GB or greater.
Changed syntax for CREATE TABLE compression options	Made the compression options more consistent for <code>CREATE TABLE</code> .
Add nodetool command to force blocking batchlog replay	BatchlogManager can force batchlog replay using nodetool.
Nodetool over SSL	Nodetool can connect using SSL like <code>cqlsh</code> .
New nodetool options for hinted handoffs	Nodetool options <code>disablehintsfordc</code> and <code>enablehintsfordc</code> added. to selectively disable or enable hinted handoffs for a data center.

nodetool stop	Nodetool option added to stop compactions.
Other notable changes	
Requires Java 8	Java 8 is now required.
nodetool cfstats and nodetool cfhistograms renamed	Renamed <code>nodetool cfstats</code> to <code>nodetool tablestats</code> . Renamed <code>nodetool cfhistograms</code> to <code>nodetool tablehistograms</code> .
Native protocol v1 and v2 are dropped	Native protocol v1 and v2 are dropped in Cassandra 3.0.
DataStax AMI does not install Cassandra 2.2 and later	<p>You can install Cassandra 2.1 and earlier versions on Amazon EC2 using the DataStax AMI (Amazon Machine Image) as described in the AMI documentation for Cassandra 2.1.</p> <p>To install Cassandra 2.2 and later on Amazon EC2, use a trusted AMI for your platform and the appropriate install method for that platform.</p>

Understanding the architecture

Architecture in brief

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure. Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. Each node frequently exchanges state information about itself and other nodes across the cluster using peer-to-peer gossip communication protocol. A sequentially written commit log on each node captures write activity to ensure data durability. Data is then indexed and written to an in-memory structure, called a memtable, *memtable* which resembles a write-back cache. Each time the memory structure is full, the data is written to disk in an SSTable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates SSTables using a process called [compaction](#), discarding obsolete data marked for deletion with a tombstone. To ensure all data across the cluster stays consistent, various [repair](#) mechanisms are employed.

Cassandra is a partitioned row store database, where rows are organized into tables with a required primary key. Cassandra's architecture allows any [authorized user](#) to connect to any node in any data center and access data using the CQL language. For ease of use, CQL uses a similar syntax to SQL and works with table data. Developers can access CQL through `cqlsh`, DevCenter, and via drivers for application languages. Typically, a cluster has one keyspace per application composed of many different tables.

Client read or write requests can be sent to any node in the cluster. When a client connects to a node with a request, that node serves as the coordinator for that particular client operation. The coordinator acts as a proxy between the client application and the nodes that own the data being requested. The coordinator determines which nodes in the ring should get the request based on how the cluster is configured.

Key structures

- **Node**
Where you store your data. It is the basic infrastructure component of Cassandra.
- **Data center**
A collection of related nodes. A data center can be a physical data center or virtual data center. Different workloads should use separate data centers, either physical or virtual. Replication is set by data center. Using separate data centers prevents Cassandra transactions from being impacted by other workloads and keeps requests close to each other for lower latency. Depending on the replication factor, data can be written to multiple data centers. Data centers must never span physical locations.
- **Cluster**
A cluster contains one or more data centers. It can span physical locations.
- **Commit log**
All data is written first to the commit log for durability. After all its data has been flushed to SSTables, it can be archived, deleted, or recycled.
- **SSTable**
A sorted string table (SSTable) is an immutable data file to which Cassandra writes memtables periodically. SSTables are append only and stored on disk sequentially and maintained for each Cassandra table.
- **CQL Table**
A collection of ordered columns fetched by table row. A table consists of columns and has a primary key.

Key components for configuring Cassandra

- **Gossip**
A peer-to-peer communication protocol to discover and share location and state information about the other nodes in a Cassandra cluster. Gossip information is also persisted locally by each node to use immediately when a node restarts.
- **Partitioner**
A partitioner determines how to distribute the data across the nodes in the cluster and which node to place the first copy of data on. Basically, a partitioner is a hash function for computing the token of a partition key. Each row of data is uniquely identified by a partition key and distributed across the cluster by the value of the token. The [Murmur3Partitioner](#) is the default partitioning strategy for new Cassandra clusters and the right choice for new clusters in almost all cases.

You must set the partitioner and assign the node a [num_tokens](#) value for each node. The number of tokens you assign depends on the [hardware capabilities](#) of the system. If not using virtual nodes (vnodes), use the [initial_token](#) setting instead.
- **Replication factor**
The total number of replicas across the cluster. A replication factor of 1 means that there is only one copy of each row on one node. A replication factor of 2 means two copies of each row, where each copy is on a different node. All replicas are equally important; there is no primary or master replica. You define the replication factor for each data center. Generally you should set the replication strategy greater than one, but no more than the number of nodes in the cluster.
- **Replica placement strategy**
Cassandra stores copies (replicas) of data on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines which nodes to place replicas on. The first replica of data is simply the first copy; it is not unique in any sense. The [NetworkTopologyStrategy](#) is highly recommended for most deployments because it is much easier to expand to multiple data centers when required by future expansion.

When creating a keyspace, you must define the replica placement strategy and the number of replicas you want.

- [Snitch](#)

A snitch defines groups of machines into data centers and racks (the topology) that the replication strategy uses to place replicas.

You must configure a [snitch](#) when you create a cluster. All snitches use a dynamic snitch layer, which monitors performance and chooses the best replica for reading. It is enabled by default and recommended for use in most deployments. Configure dynamic snitch thresholds for each node in the `cassandra.yaml` configuration file.

The default [SimpleSnitch](#) does not recognize data center or rack information. Use it for single-data center deployments or single-zone in public clouds. The [GossipingPropertyFileSnitch](#) is recommended for production. It defines a node's data center and rack and uses gossip for propagating this information to other nodes.

- [The `cassandra.yaml` configuration file](#)

The main configuration file for setting the initialization properties for a cluster, caching parameters for tables, properties for tuning and resource utilization, timeout settings, client connections, backups, and security.

By default, a node is configured to store the data it manages in a directory set in the `cassandra.yaml` file.

In a production cluster deployment, you can change the [commitlog-directory](#) to a different disk drive from the [data_file_directories](#).

- [System keyspace table properties](#)

You set storage configuration attributes on a per-keyspace or per-table basis programmatically or using a client application, such as CQL.

Related reference

[cassandra.yaml configuration file](#) on page 76

Related information

[Install locations](#) on page 74

Internode communications (gossip)

Gossip is a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about. The gossip process runs every second and exchanges state messages with up to three other nodes in the cluster. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster. A gossip message has a version associated with it, so that during a gossip exchange, older information is overwritten with the most current state for a particular node.

To prevent problems in gossip communications, use the same list of seed nodes for all nodes in a cluster. This is most critical the first time a node starts up. By default, a node remembers other nodes it has gossiped with between subsequent restarts. The seed node designation has no purpose other than bootstrapping the gossip process for new nodes joining the cluster. Seed nodes are *not* a single point of failure, nor do they have any other special purpose in cluster operations beyond the bootstrapping of nodes.

Attention: In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is **not** recommended because of increased maintenance and reduced

gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).

Failure detection and recovery

Failure detection is a method for locally determining from gossip state and history if another node in the system is down or has come back up. Cassandra uses this information to avoid routing client requests to unreachable nodes whenever possible. (Cassandra can also avoid routing requests to nodes that are alive, but performing poorly, through the [dynamic snitch](#).)

The gossip process tracks state from other nodes both directly (nodes gossiping directly to it) and indirectly (nodes communicated about secondhand, third-hand, and so on). Rather than have a fixed threshold for marking failing nodes, Cassandra uses an accrual detection mechanism to calculate a per-node threshold that takes into account network performance, workload, and historical conditions. During gossip exchanges, every node maintains a sliding window of inter-arrival times of gossip messages from other nodes in the cluster. Configuring the [phi_convict_threshold](#) property adjusts the sensitivity of the failure detector. Lower values increase the likelihood that an unresponsive node will be marked as down, while higher values decrease the likelihood that transient failures causing node failure. Use the default value for most situations, but increase it to 10 or 12 for Amazon EC2 (due to frequently encountered network congestion). In unstable network environments (such as EC2 at times), raising the value to 10 or 12 helps prevent false failures. Values higher than 12 and lower than 5 are not recommended.

Node failures can result from various causes such as hardware failures and network outages. Node outages are often transient but can last for extended periods. Because a node outage rarely signifies a permanent departure from the cluster it does not automatically result in permanent removal of the node from the ring. Other nodes will periodically try to re-establish contact with failed nodes to see if they are back up. To permanently change a node's membership in a cluster, administrators must explicitly add or remove nodes from a Cassandra cluster using the [nodetool utility](#).

When a node comes back online after an outage, it may have missed writes for the replica data it maintains. [Repair mechanisms](#) exist to recover missed data, such as hinted handoffs and manual repair with [nodetool repair](#). The length of the outage will determine which repair mechanism is used to make the data consistent.

Data distribution and replication

In Cassandra, data distribution and replication go together. Data is organized by table and identified by a primary key, which determines which node the data is stored on. Replicas are copies of rows. When data is first written, it is also referred to as a replica.

Factors influencing replication include:

- [Virtual nodes](#): assigns data ownership to physical machines.
- [Partitioner](#): partitions the data across the cluster.
- [Replication strategy](#): determines the replicas for each row of data.
- [Snitch](#): defines the topology information that the replication strategy uses to place replicas.

Consistent hashing

Consistent hashing allows distribution of data across a cluster to minimize reorganization when nodes are added or removed. Consistent hashing partitions data based on the partition key. (For an explanation of partition keys and primary keys, see the [Data modeling example](#) in *CQL for Cassandra 2.2*.)

For example, if you have the following data:

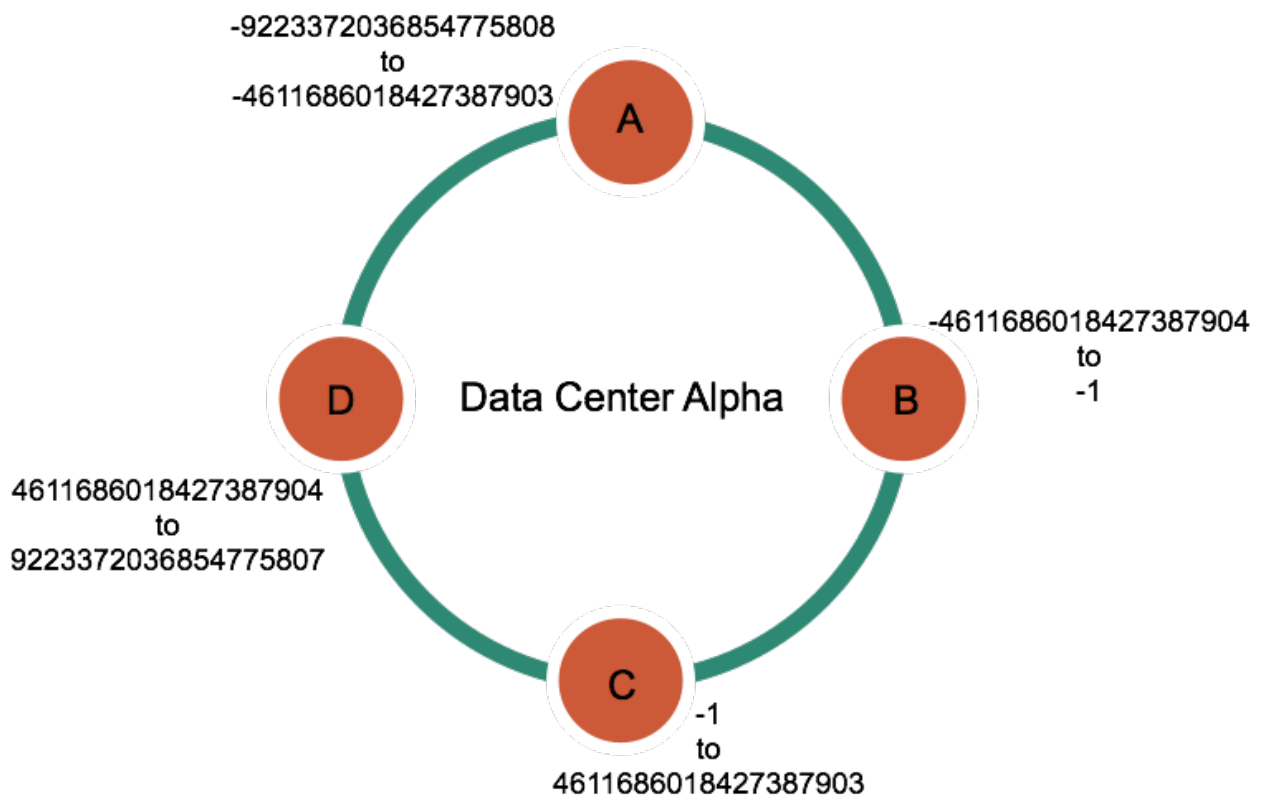
name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F

Cassandra assigns a hash value to each partition key:

Partition key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Each node in the cluster is responsible for a range of data based on the hash value.

Figure: Hash values in a four node cluster



Cassandra places the data on each node according to the value of the partition key and the range that the node is responsible for. For example, in a four node cluster, the data in this example is distributed as follows:

Node	Start range	End range	Partition key	Hash value
A	-9223372036854775808	-4611686018427387903	johnny	-6723372854036780875

Node	Start range	End range	Partition key	Hash value
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

Virtual nodes

Virtual nodes, known as Vnodes, distribute data across nodes at a finer granularity than can be easily achieved if calculated tokens are used. Vnodes simplify many tasks in Cassandra:

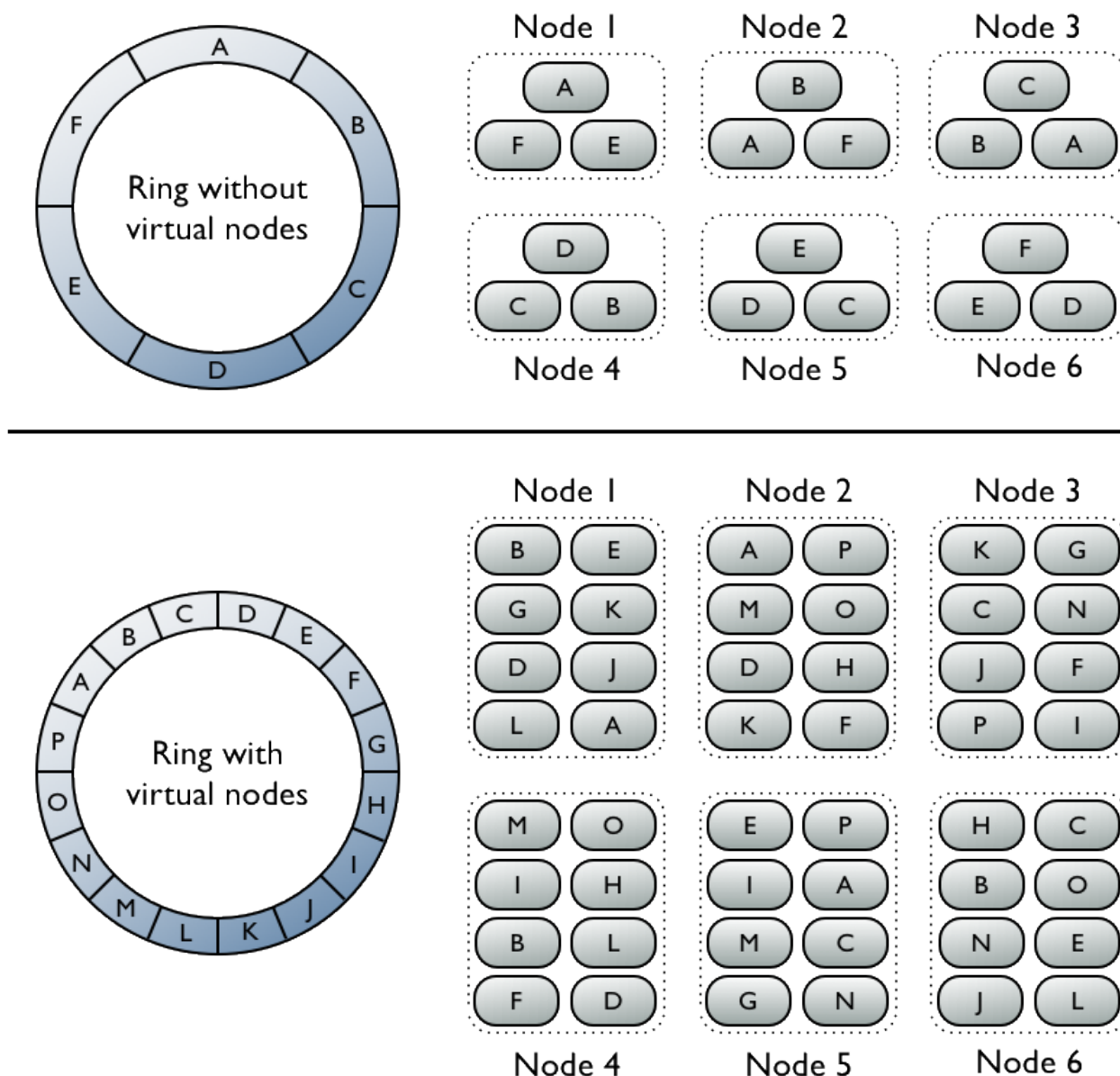
- Tokens are automatically calculated and assigned to each node.
- Rebalancing a cluster is automatically accomplished when adding or removing nodes. When a node joins the cluster, it assumes responsibility for an even portion of data from the other nodes in the cluster. If a node fails, the load is spread evenly across other nodes in the cluster.
- Rebuilding a dead node is faster because it involves every other node in the cluster.
- The proportion of vnodes assigned to each machine in a cluster can be assigned, so smaller and larger computers can be used in building a cluster.

For more information, see the article [Virtual nodes in Cassandra 1.2](#). To convert an existing cluster to vnodes, see [Enabling virtual nodes on an existing production cluster](#) on page 106.

How data is distributed across a cluster (using virtual nodes)

Prior to Cassandra 1.2, you had to calculate and assign a single [token](#) to each node in a cluster. Each token determined the node's position in the ring and its portion of data according to its hash value. In Cassandra 1.2 and later, each node is allowed many tokens. The new paradigm is called virtual nodes (vnodes). Vnodes allow each node to own a large number of small partition ranges distributed throughout the cluster. Vnodes also use consistent hashing to distribute data but using them doesn't require token generation and assignment.

Figure: Virtual vs single-token architecture



The top portion of the graphic shows a cluster without vnodes. In this paradigm, each node is assigned a single token that represents a location in the ring. Each node stores data determined by mapping the partition key to a token value within a range from the previous node to its assigned value. Each node also contains copies of each row from other nodes in the cluster. For example, if the replication factor is 3, range E replicates to nodes 5, 6, and 1. Notice that a node owns exactly one contiguous partition range in the ring space.

The bottom portion of the graphic shows a ring with vnodes. Within a cluster, virtual nodes are randomly selected and non-contiguous. The placement of a row is determined by the hash of the partition key within many smaller partition ranges belonging to each node.

Data replication

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines the nodes where replicas are placed. The total number of replicas across the cluster is referred to as the replication factor. A replication factor of 1 means that there is only one copy of each row in the cluster. If the node containing the row goes down, the row cannot be retrieved. A replication factor of 2 means two copies of each row, where each copy is on a different node. All replicas are equally important;

there is no primary or master replica. As a general rule, the replication factor should not exceed the number of nodes in the cluster. However, you can increase the replication factor and then add the desired number of nodes later.

Two replication strategies are available:

- `SimpleStrategy`: Use only for a single data center and one rack. If you ever intend more than one data center, use the `NetworkTopologyStrategy`.
- `NetworkTopologyStrategy`: Highly recommended for most deployments because it is much easier to expand to multiple data centers when required by future expansion.

SimpleStrategy

Use only for a single data center and one rack. `SimpleStrategy` places the first replica on a node determined by the partitioner. Additional replicas are placed on the next nodes clockwise in the ring without considering topology (rack or data center location).

NetworkTopologyStrategy

Use `NetworkTopologyStrategy` when you have (or plan to have) your cluster deployed across multiple data centers. This strategy specifies how many replicas you want in each data center.

`NetworkTopologyStrategy` places replicas in the same data center by walking the ring clockwise until reaching the first node in another rack. `NetworkTopologyStrategy` attempts to place replicas on distinct racks because nodes in the same rack (or similar physical grouping) often fail at the same time due to power, cooling, or network issues.

When deciding how many replicas to configure in each data center, the two primary considerations are (1) being able to satisfy reads locally, without incurring cross data-center latency, and (2) failure scenarios. The two most common ways to configure multiple data center clusters are:

- Two replicas in each data center: This configuration tolerates the failure of a single node per replication group and still allows local reads at a consistency level of `ONE`.
- Three replicas in each data center: This configuration tolerates either the failure of one node per replication group at a strong consistency level of `LOCAL_QUORUM` or multiple node failures per data center using consistency level `ONE`.

Asymmetrical replication groupings are also possible. For example, you can have three replicas in one data center to serve real-time application requests and use a single replica elsewhere for running analytics.

Replication strategy is defined per keyspace, and is set during keyspace creation. To set up a keyspace, see [creating a keyspace](#).

Partitioners

A partitioner determines how data is distributed across the nodes in the cluster (including replicas). Basically, a partitioner is a function for deriving a token representing a row from its partition key, typically by hashing. Each row of data is then distributed across the cluster by the value of the token.

Both the `Murmur3Partitioner` and `RandomPartitioner` use tokens to help assign equal portions of data to each node and evenly distribute data from all the tables throughout the ring or other grouping, such as a keyspace. This is true even if the tables use different partition keys, such as usernames or timestamps. Moreover, the read and write requests to the cluster are also evenly distributed and load balancing is simplified because each part of the hash range receives an equal number of rows on average. For more detailed information, see [Consistent hashing](#) on page 14.

The main difference between the two partitioners is how each generates the token hash values. The `RandomPartitioner` uses a cryptographic hash that takes longer to generate than the `Murmur3Partitioner`. Cassandra doesn't really need a cryptographic hash, so using the `Murmur3Partitioner` results in a 3-5 times improvement in performance.

Cassandra offers the following partitioners that can be set in the [cassandra.yaml](#) file.

- `Murmur3Partitioner` (default): uniformly distributes data across the cluster based on `MurmurHash` hash values.
- `RandomPartitioner`: uniformly distributes data across the cluster based on MD5 hash values.
- `ByteOrderedPartitioner`: keeps an ordered distribution of data lexically by key bytes

The `Murmur3Partitioner` is the default partitioning strategy for Cassandra 1.2 and later new clusters and the right choice for new clusters in almost all cases. However, the partitioners are not compatible and data partitioned with one partitioner cannot be easily converted to the other partitioner.

Note: If using virtual nodes (vnodes), you do **not** need to calculate the tokens. If not using vnodes, you **must** calculate the tokens to assign to the `initial_token` parameter in the `cassandra.yaml` file. See [Generating tokens](#) on page 111 and use the method for the type of partitioner you are using.

Related information

[Install locations](#) on page 74

Murmur3Partitioner

The `Murmur3Partitioner` is the default partitioner. The `Murmur3Partitioner` provides faster hashing and improved performance than the `RandomPartitioner`. The `Murmur3Partitioner` can be used with vnodes. However, if you don't use vnodes, you must calculate the tokens, as described in [Generating tokens](#).

Use `Murmur3Partitioner` for new clusters; you cannot change the partitioner in existing clusters that use a different partitioner. The `Murmur3Partitioner` uses the `MurmurHash` function. This hashing function creates a 64-bit hash value of the partition key. The possible range of hash values is from -2^{63} to $+2^{63}-1$.

When using the `Murmur3Partitioner`, you can page through all rows using the `token function` in a CQL query.

RandomPartitioner

The `RandomPartitioner` was the default partitioner prior to Cassandra 1.2. It is included for backwards compatibility. The `RandomPartitioner` can be used with virtual nodes (vnodes). However, if you don't use vnodes, you must calculate the tokens, as described in [Generating tokens](#). The `RandomPartitioner` distributes data evenly across the nodes using an MD5 hash value of the row key. The possible range of hash values is from 0 to $2^{127}-1$.

When using the `RandomPartitioner`, you can page through all rows using the `token function` in a CQL query.

ByteOrderedPartitioner

Cassandra provides the `ByteOrderedPartitioner` for ordered partitioning. It is included for backwards compatibility. This partitioner orders rows lexically by key bytes. You calculate tokens by looking at the actual values of your partition key data and using a hexadecimal representation of the leading character(s) in a key. For example, if you wanted to partition rows alphabetically, you could assign an A token using its hexadecimal representation of 41.

Using the ordered partitioner allows ordered scans by primary key. This means you can scan rows as though you were moving a cursor through a traditional index. For example, if your application has user names as the partition key, you can scan rows for users whose names fall between Jake and Joe. This type of query is not possible using randomly partitioned partition keys because the keys are stored in the order of their MD5 hash (not sequentially).

Although having the capability to do range scans on rows sounds like a desirable feature of ordered partitioners, there are ways to achieve the same functionality using [table indexes](#).

Using an ordered partitioner is not recommended for the following reasons:

Difficult load balancing

More administrative overhead is required to load balance the cluster. An ordered partitioner requires administrators to manually calculate partition ranges based on their estimates of the partition key distribution. In practice, this requires actively moving node tokens around to accommodate the actual distribution of data once it is loaded.

Sequential writes can cause hot spots

If your application tends to write or update a sequential block of rows at a time, then the writes are not be distributed across the cluster; they all go to one node. This is frequently a problem for applications dealing with timestamped data.

Uneven load balancing for multiple tables

If your application has multiple tables, chances are that those tables have different row keys and different distributions of data. An ordered partitioner that is balanced for one table may cause hot spots and uneven distribution for another table in the same cluster.

Related information

[Install locations](#) on page 74

Snitches

A snitch determines which data centers and racks nodes belong to. They inform Cassandra about the network topology so that requests are routed efficiently and allows Cassandra to distribute replicas by grouping machines into data centers and racks. Specifically, the replication strategy places the replicas based on the information provided by the new snitch. All nodes must return to the same rack and data center. Cassandra does its best not to have more than one replica on the same rack (which is not necessarily a physical location).

Note: If you change snitches, you may need to perform additional steps because the snitch affects where replicas are placed. See [Switching snitches](#) on page 127.

Dynamic snitching

By default, all snitches also use a dynamic snitch layer that monitors read latency and, when possible, routes requests away from poorly-performing nodes. The dynamic snitch is enabled by default and is recommended for use in most deployments. For information on how this works, see [Dynamic snitching in Cassandra: past, present, and future](#). Configure dynamic snitch thresholds for each node in the `cassandra.yaml` configuration file.

For more information, see the properties listed under [Failure detection and recovery](#) on page 14.

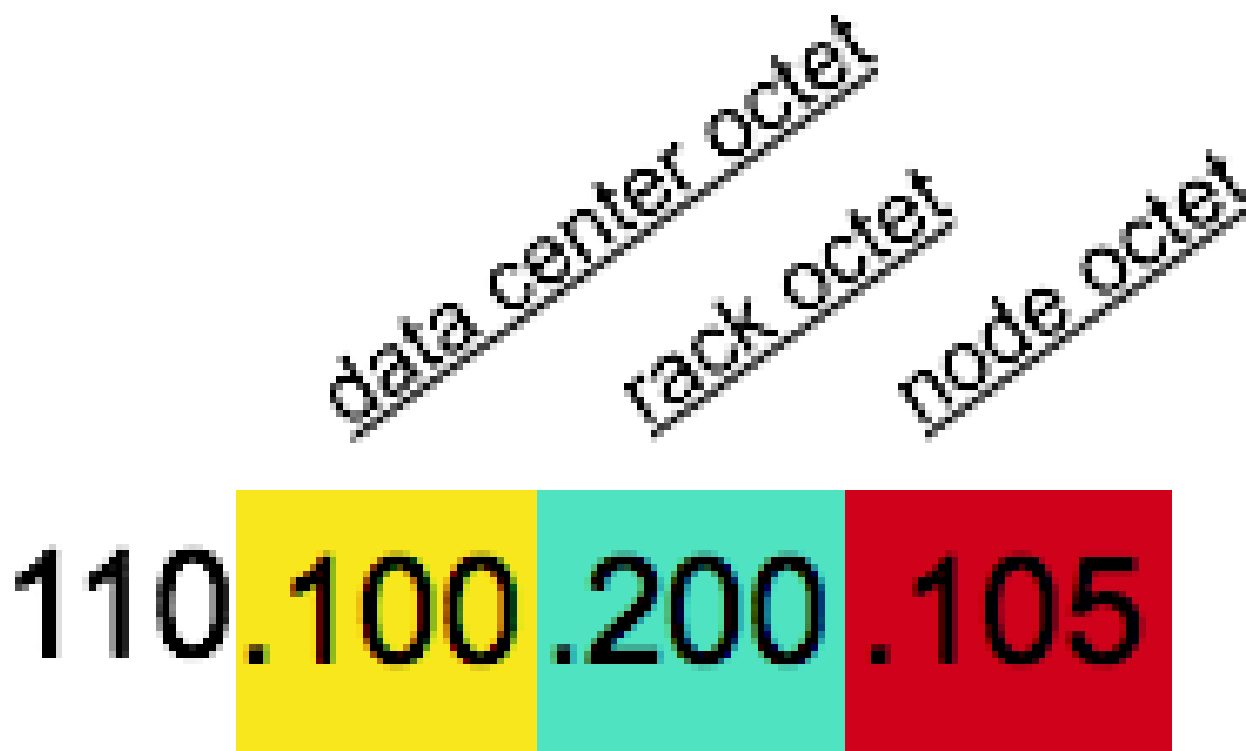
SimpleSnitch

The SimpleSnitch (default) is used only for single-data center deployments. It does not recognize data center or rack information and can be used only for single-data center deployments or single-zone in public clouds. It treats strategy order as proximity, which can improve cache locality when disabling read repair.

Using a SimpleSnitch, you [define the keyspace](#) to use SimpleStrategy and specify a replication factor.

RackInferringSnitch

The RackInferringSnitch determines the proximity of nodes by rack and data center, which are assumed to correspond to the 3rd and 2nd octet of the node's IP address, respectively. This snitch is best used as an example for writing a custom snitch class (unless this happens to match your deployment conventions).



PropertyFileSnitch

This snitch determines proximity as determined by rack and data center. It uses the network details located in the `cassandra-topology.properties` file. When using this snitch, you can define your data center names to be whatever you want. Make sure that the data center names correlate to the name of your data centers in the [keyspace definition](#). Every node in the cluster should be described in the `cassandra-topology.properties` file, and this file should be exactly the same on every node in the cluster.

Procedure

If you had non-uniform IPs and two physical data centers with two racks in each, and a third logical data center for replicating analytics data, the `cassandra-topology.properties` file might look like this:

Note: Data center and rack names are case-sensitive.

```
# Data Center One

175.56.12.105=DC1:RAC1
175.50.13.200=DC1:RAC1
175.54.35.197=DC1:RAC1

120.53.24.101=DC1:RAC2
120.55.16.200=DC1:RAC2
120.57.102.103=DC1:RAC2

# Data Center Two

110.56.12.120=DC2:RAC1
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1
```

```
50.33.23.120=DC2:RAC2
50.45.14.220=DC2:RAC2
50.17.10.203=DC2:RAC2

# Analytics Replication Group

172.106.12.120=DC3:RAC1
172.106.12.121=DC3:RAC1
172.106.12.122=DC3:RAC1

# default for unknown nodes
default =DC3:RAC1
```

GossipingPropertyFileSnitch

This snitch is recommended for production. It uses rack and data center information for the local node defined in the `cassandra-rackdc.properties` file and propagates this information to other nodes via gossip.

The `cassandra-rackdc.properties` file defines the default data center and rack used by this snitch:

Note: Data center and rack names are case-sensitive.

```
dc=DC1
rack=RAC1
```

To save bandwidth, add the `prefer_local=true` option. This option tells Cassandra to use the local IP address when communication is not across different data centers.

To allow migration from the `PropertyFileSnitch`, the `GossipingPropertyFileSnitch` uses the `cassandra-topology.properties` file when present.

The location of the `cassandra-rackdc.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-rackdc.properties</code>
Tarball installations	<code>install_location/conf/cassandra-rackdc.properties</code>
Windows installations	

Ec2Snitch

Use the `Ec2Snitch` for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region.

In EC2 deployments, the region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the `us-east-1` region, `us-east` is the data center name and `1` is the rack location. (Racks are important for distributing replicas, but not for data center naming.) Because private IPs are used, this snitch does not work across multiple regions.

If you are using only a single data center, you do not need to specify any properties.

If you need multiple data centers, set the `dc_suffix` options in the `cassandra-rackdc.properties` file. Any other lines are ignored.

For example, for each node within the `us-east` region, specify the data center in its `cassandra-rackdc.properties` file:

Note: Data center names are case-sensitive.

- **node0**
`dc_suffix=_1_cassandra`
- **node1**
`dc_suffix=_1_cassandra`
- **node2**
`dc_suffix=_1_cassandra`
- **node3**
`dc_suffix=_1_cassandra`
- **node4**
`dc_suffix=_1_analytics`
- **node5**
`dc_suffix=_1_search`

This results in three data centers for the region:

```
us-east-1_cassandra
us-east-1_analytics
us-east-1_search
```

Note: The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200.

Keyspace strategy options

When defining your [keyspace strategy options](#), use the EC2 region name, such as ``us-east``, as your data center name.

Ec2MultiRegionSnitch

Use the Ec2MultiRegionSnitch for deployments on Amazon EC2 where the cluster spans multiple regions.

You must configure settings in both the `cassandra.yaml` file and the property file (`cassandra-rackdc.properties`) used by the Ec2MultiRegionSnitch.

Configuring `cassandra.yaml` for cross-region communication

The Ec2MultiRegionSnitch uses public IP designated in the `broadcast_address` to allow cross-region connectivity. Configure each node as follows:

1. In the `cassandra.yaml`, set the [listen_address](#) to the *private* IP address of the node, and the [broadcast_address](#) to the *public* IP address of the node.

This allows Cassandra nodes in one EC2 region to bind to nodes in another region, thus enabling multiple data center support. For intra-region traffic, Cassandra switches to the private IP after establishing a connection.

2. Set the addresses of the seed nodes in the `cassandra.yaml` file to that of the *public* IP. Private IP are not routable between networks. For example:

```
seeds: 50.34.16.33, 60.247.70.52
```

To find the public IP address, from each of the seed nodes in EC2:

```
$ curl http://instance-data/latest/meta-data/public-ipv4
```

Note: Do not make all nodes seeds, see [Internode communications \(gossip\)](#) on page 13.

3. Be sure that the `storage_port` or `ssl_storage_port` is open on the public IP firewall.

Configuring the snitch for cross-region communication

In EC2 deployments, the region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the us-east-1 region, us-east is the data center name and 1 is the rack location. (Racks are important for distributing replicas, but not for data center naming.)

For each node, specify its data center in the `cassandra-rackdc.properties`. The `dc_suffix` option defines the data centers used by the snitch. Any other lines are ignored.

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

Region: us-east	Region: us-west
<div>Node and data center:</div> <ul style="list-style-type: none">node0 dc_suffix=_1_cassandranode1 dc_suffix=_1_cassandranode2 dc_suffix=_2_cassandranode3 dc_suffix=_2_cassandranode4 dc_suffix=_1_analyticsnode5 dc_suffix=_1_search <div>This results in four us-east data centers:</div> <div>us-east_1_cassandra us-east_2_cassandra us-east_1_analytics us-east_1_search</div>	<div>Node and data center:</div> <ul style="list-style-type: none">node0 dc_suffix=_1_cassandranode1 dc_suffix=_1_cassandranode2 dc_suffix=_2_cassandranode3 dc_suffix=_2_cassandranode4 dc_suffix=_1_analyticsnode5 dc_suffix=_1_search <div>This results in four us-west data centers:</div> <div>us-west_1_cassandra us-west_2_cassandra us-west_1_analytics us-west_1_search</div>

Keyspace strategy options

When defining your `keyspace strategy options`, use the EC2 region name, such as ``us-east``, as your data center name.

Related information

[Install locations](#) on page 74

GoogleCloudSnitch

Use the GoogleCloudSnitch for Cassandra deployments on [Google Cloud Platform](#) across one or more regions. The region is treated as a data center and the availability zones are treated as racks within the data center. All communication occurs over private IP addresses within the same logical network.

The region name is treated as the data center name and zones are treated as racks within a data center. For example, if a node is in the us-central1-a region, us-central1 is the data center name and a is the rack location. (Racks are important for distributing replicas, but not for data center naming.) This snitch can work across multiple regions without additional configuration.

If you are using only a single data center, you do not need to specify any properties.

If you need multiple data centers, set the `dc_suffix` options in the `cassandra-rackdc.properties` file. Any other lines are ignored.

For example, for each node within the us-central1 region, specify the data center in its `cassandra-rackdc.properties` file:

Note: Data center names are case-sensitive.

- **node0**
`dc_suffix=_a_cassandra`
- **node1**
`dc_suffix=_a_cassandra`
- **node2**
`dc_suffix=_a_cassandra`
- **node3**
`dc_suffix=_a_cassandra`
- **node4**
`dc_suffix=_a_analytics`
- **node5**
`dc_suffix=_a_search`

Note: Data center and rack names are case-sensitive.

CloudstackSnitch

Use the CloudstackSnitch for [Apache Cloudstack](#) environments. Because zone naming is free-form in Apache Cloudstack, this snitch uses the widely-used <country> <location> <az> notation.

Database internals

Storage engine

Cassandra uses a storage structure similar to a [Log-Structured Merge Tree](#), unlike a typical relational database that uses a [B-Tree](#). Cassandra avoids reading before writing. Read-before-write, especially in a large distributed system, can produce stalls in read performance and other problems. For example, two clients read at the same time, one overwrites the row to make update A, and the other overwrites the row to make update B, removing update A. Reading before writing also corrupts caches and increases IO requirements. To avoid a read-before-write condition, the storage engine groups inserts/updates to be made, and sequentially writes only the updated parts of a row in append mode. Cassandra never re-writes or re-reads existing data, and never overwrites the rows in place.

A log-structured engine that avoids overwrites and uses sequential IO to update data is essential for writing to solid-state disks (SSD) and hard disks (HDD) On HDD, writing randomly involves a higher number of

seek operations than sequential writing. The seek penalty incurred can be substantial. Using sequential IO (thereby avoiding [write amplification](#) and disk failure), Cassandra accommodates inexpensive, consumer SSDs extremely well.

How Cassandra reads and writes data

To manage and access data in Cassandra, it is important to understand how Cassandra stores data. The hinted handoff feature plus Cassandra conformance and non-conformance to the ACID (atomic, consistent, isolated, durable) database properties are key concepts to understand reads and writes. In Cassandra, consistency refers to how up-to-date and synchronized a row of data is on all of its replicas.

Client utilities and application programming interfaces (APIs) for developing applications for data storage and retrieval are [available](#).

How is data written?

Cassandra processes data at several stages on the write path, starting with the immediate logging of a write and ending in with a write of data to disk:

- Logging data in the commit log
- Writing data to the memtable
- Flushing data from the memtable
- Storing data on disk in SSTables

Logging writes and memtable storage

When a write occurs, Cassandra stores the data in a memory structure called memtable, and to provide [configurable durability](#), it also appends writes to the commit log on disk. The commit log receives every write made to a Cassandra node, and these durable writes survive permanently even if power fails on a node. The memtable is a write-back cache of data partitions that Cassandra looks up by key. The memtable stores writes until reaching a configurable limit, and then is flushed.

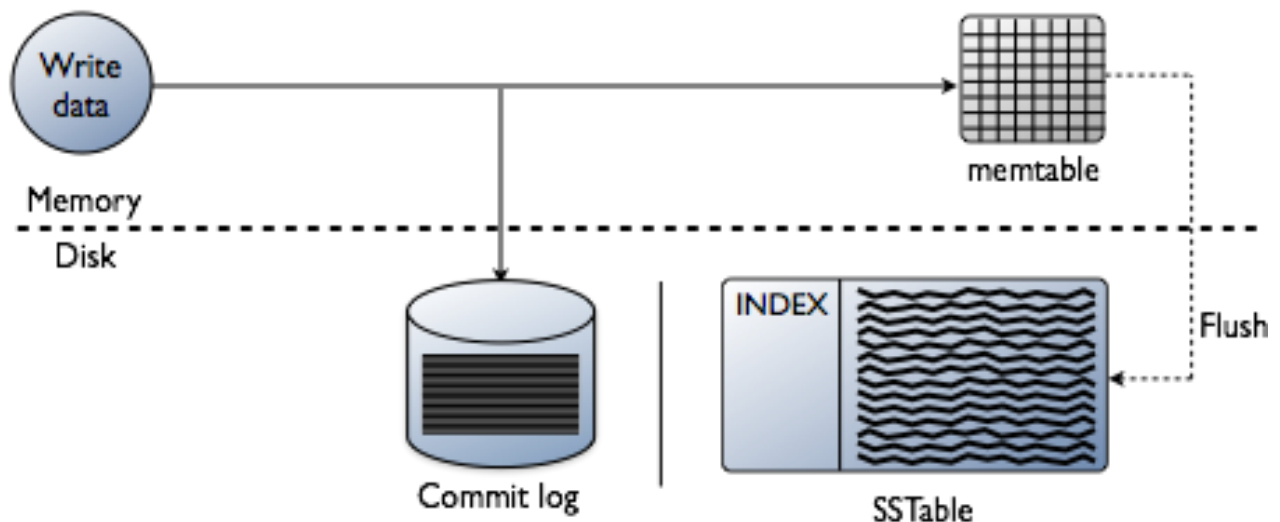
Flushing data from the memtable

To flush the data, Cassandra sorts memtables by token and then writes the data to disk sequentially. A partition index is also created on the disk that maps the tokens to a location on disk. When the memtable content exceeds the [configurable threshold](#), the memtable is put in a queue that is flushed to disk. The queue can be configured with the `memtable_heap_space_in_mb` or `memtable_offheap_space_in_mb` setting in the `cassandra.yaml` file. If the data to be flushed exceeds the queue size, Cassandra blocks writes until the next flush succeeds. You can manually flush a table using [nodetool flush](#). To reduce the commit log replay time, the recommended best practice is to flush the memtable before you restart the nodes. Commit log replay is the process of reading the commit log to recover lost writes in the event of interrupted operations.

Data in the commit log is purged after its corresponding data in the memtable is flushed to an SSTable on disk.

Storing data on disk in SSTables

Memtables and SSTables are maintained per table. SSTables are immutable, not written to again after the memtable is flushed. Consequently, a partition is typically stored across multiple SSTable files. A number of other SSTable structures exist to assist read operations:



For each SSTable, Cassandra creates these structures:

- Partition index
A list of partition keys and the start position of rows in the data file written on disk
- Partition summary
A sample of the partition index stored in memory
- Bloom filter
A structure stored in memory that checks if row data exists in the memtable before accessing SSTables on disk

The SSTables are files stored on disk. The naming convention for SSTable files has changed with Cassandra 2.2 and later to shorten the file path. The data files are stored in a data directory that varies with installation. For each keyspace, a directory within the data directory stores each table. For example, `/data/data/ks1/cf1-5be396077b811e3a3ab9dc4b9ac088d/1a-1-big-Data.db` represents a data file. `ks1` represents the keyspace name to distinguish the keyspace for streaming or bulk loading data. A hexadecimal string, `5be396077b811e3a3ab9dc4b9ac088d` in this example, is appended to table names to represent unique table IDs.

Several files are written to store the data, partition summary, statistics, and other information.

Cassandra creates a subdirectory for each table, which allows you to symlink a table to a chosen physical drive or data volume. This provides the capability to move very active tables to faster media, such as SSDs for better performance, and also divides tables across all attached storage devices for better I/O balance at the storage layer.

How is data maintained?

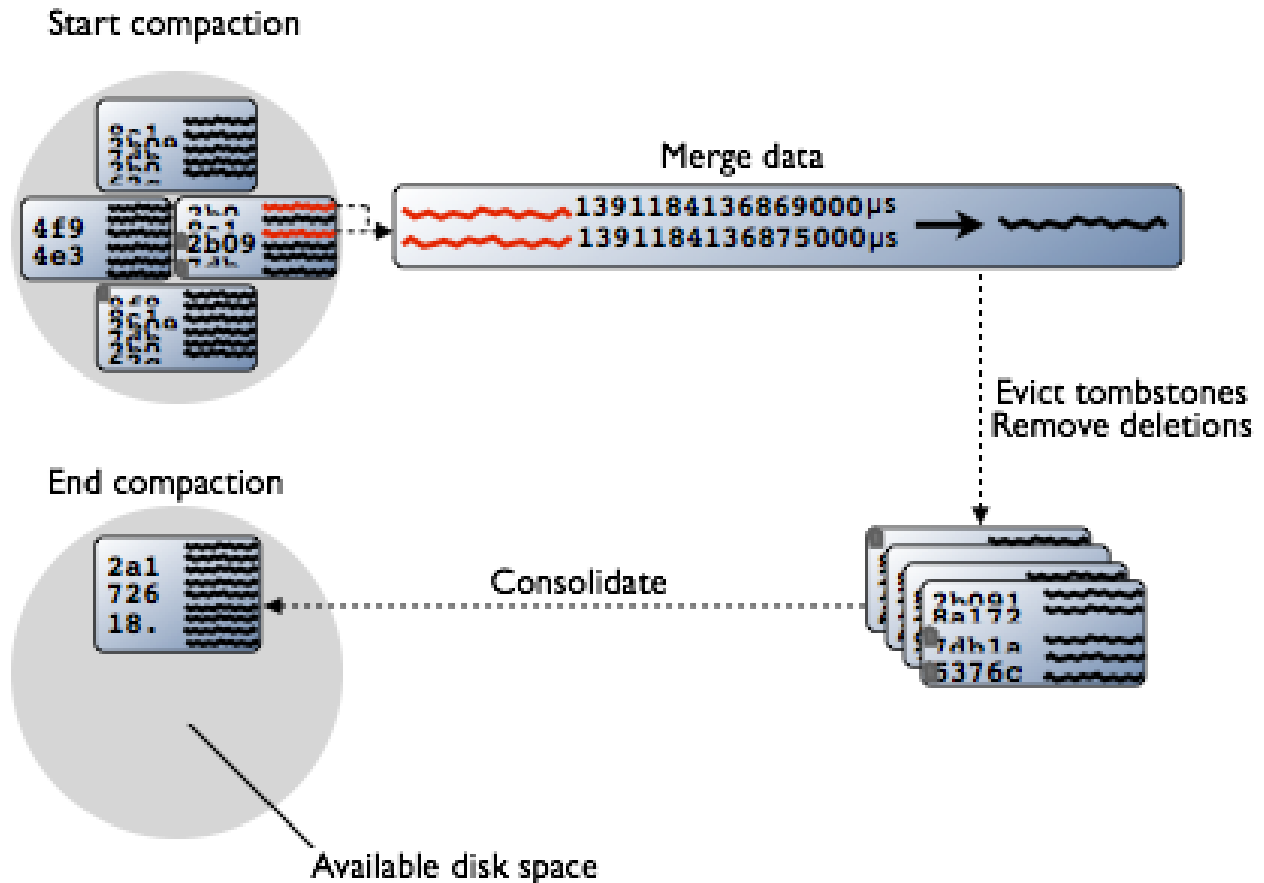
Cassandra maintains data on disk by consolidating SSTables. SSTables are immutable and accumulate on disk and must periodically be merged using compaction.

Compaction

Periodic compaction is essential to a healthy Cassandra database because Cassandra does not insert/update in place. As inserts/updates occur, instead of overwriting the rows, Cassandra writes a new timestamped version of the inserted or updated data in another SSTable. Cassandra also does not delete in place because SSTables are immutable. Instead, Cassandra marks data to be deleted using a tombstone. Tombstones exist for a configured time period defined by the `gc_grace_seconds` value set on the table.

Over time, many versions of a row might exist in different SSTables. Each version has a different set of columns stored. As SSTables accumulate, more and more SSTables must be read in order to retrieve an entire row of data.

Compaction merges the data in each SSTable by partition key, selecting the latest data for storage based on its timestamp. Because rows are sorted by partition key within each SSTable, the merge process does not use random I/O and is performant. After evicting tombstones and removing deleted data, columns, and rows, the compaction process consolidates SSTables into a new single SSTable file. The old SSTable files are deleted as soon as any pending reads finish using the files.



During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. Disk space occupied by old SSTables becomes available for reuse when the new SSTable is ready. Cassandra 2.1 and later improves read performance after compaction because of incremental replacement of compacted SSTables. Instead of waiting for the entire compaction to finish and then throwing away the old SSTable, Cassandra can read data directly from the new SSTable even before it finishes writing.

As data is written to the new SSTable and reads are directed to it, the corresponding data in the old SSTables is no longer accessed and is evicted from the page cache. Thus begins an incremental process of caching the new SSTable, while directing reads away from the old one, thus avoiding the dramatic cache miss. Cassandra provides predictable high performance even under heavy load.

Types of compaction

Different compaction strategies have strengths and weaknesses. Understanding how each type works is vital to making the right choice for your application workload. `SizeTieredCompactionStrategy` (STCS) is recommended for write-intensive workloads. `LeveledCompactionStrategy` (LCS) is recommended for read-intensive workloads. `DateTieredCompactionStrategy` (DTCS) is recommended for time series data and expiring TTL data.

SizeTieredCompactionStrategy (STCS)

Recommended for write-intensive workloads.

Pros: Compacts write-intensive workload very well.

Cons: Might hold onto stale data too long. Amount of memory needed increases over time.

The SizeTieredCompactionStrategy (STCS) initiates compaction when a set number (default is 4) of similar-sized SSTables have accumulated. Compaction merges the SSTables to create one larger SSTable. As larger SSTables accumulate, the same process occurs, merging the larger SSTables into an even larger SSTable. At any given time, several SSTables of varying sizes are present. While this strategy works quite well to compact a write-intensive workload, when reads are needed, several SSTables still must be retrieved to find all the data for a row. There is no guarantee that a row's data will be restricted to a small number of SSTables. Also, predicting the eviction of deleted data is uneven, because SSTable size is the trigger for compaction, and SSTables might not grow quickly enough to merge and evict old data. As the largest SSTables grow in size, the amount of memory needed for compaction to hold both the new and old SSTables simultaneously can outstrip a typical amount of RAM on a node.

LeveledCompactionStrategy (LCS)

Recommended for read-intensive workloads.

Pros: Memory requirements are simple to predict. Read operations more predictable in latency. Stale data is evicted more frequently.

Cons: Much higher I/O utilization that can impact operation latency.

The LeveledCompactionStrategy (LCS) is intended to alleviate some of the read operation issues with the SizeTieredCompactionStrategy (STCS). As SSTables reach a certain small fixed size (default is 5MB), they are written into the first level, L0, and also merged into the first level, L1. In each level starting with L1, all SSTables are guaranteed to have non-overlapping data. Because no data is overlapping, the LeveledCompactionStrategy sometimes splits SSTables as well as merging them, to keep the files similarly sized. Each level is 10X the size of the last level, so level L1 has 10X as many SSTables as L0, and level L2 has 100X. Level L2 will start filling when L1 has been filled. Because a level contains no overlapping data, a read can be accomplished quite efficiently with very few SSTables retrieved. For many read operations, only one or two SSTables will be read. In fact, 90% of all reads will be satisfied from reading one SSTable. The worst case is one SSTable per level must be read. Less memory will be required for compacting using this strategy, with 10X the fixed size of the SSTable required. Obsolete data will be evicted more often, so deleted data will occupy a much smaller portion of the SSTables on disk. However, the compaction operations for the LeveledCompactionStrategy (LCS) take place more often and place more I/O burden on the node. For write-intensive workloads, the payoff using this strategy is generally not worth the performance loss to I/O operations. In Cassandra 2.2 and later, performance improvements have been implemented that bypass compaction operations when bootstrapping a new node using LCS into a cluster. The original data is directly moved to the correct level because there is no existing data, so no partition overlap per level is present. For more information, see [Apache Cassandra 2.2 - Bootstrapping Performance Improvements for Leveled Compaction](#).

DateTieredCompactionStrategy (DTCS)

Recommended for time series and expiring TTL workloads.

Pros: Specifically designed for time series data.

Cons: Out of order data injections can cause errors. Read repair must be turned off for DTCS.

The DateTieredCompactionStrategy (DTCS) acts similarly to STCS, but instead of compacting based on SSTable size, DTCS compacts based on SSTable age. Making the time window configurable ensures that new and old data will not be mixed in merged SSTables. In fact, using Time-To-Live (TTL) timestamps, DateTieredCompactionStrategy (DTCS) often ejects whole SSTables for old data that has expired. This strategy often results in similar-sized SSTables, too, if time series data is ingested at a steady rate. SSTables are merged when a certain minimum threshold of number of SSTables is reached within a configurable time interval. SSTables will still be merged into larger tables, like in size tiered compaction, if the required

number of SSTables falls within the time interval. However, SSTables are not compacted after reaching a configurable age, reducing the number of times data will be rewritten. SSTables compacted using this strategy can be read, especially for queries that ask for the "last hour's worth of data", very efficiently. One issue that can cause difficulty with this strategy is out-of-order writing, where a timestamped record is written for a past timestamp, for example. Read repairs can inject an out-of-order timestamping, so turn off read repairs when using the `DateTieredCompactionStrategy`. For more information about compaction strategies, see [When to Use Leveled Compaction](#) and [Leveled Compaction in Apache Cassandra](#). For `DateTieredCompactionStrategy`, see [DateTieredCompactionStrategy: Notes from the Field](#), [Date-Tiered Compaction in Cassandra](#) or [DateTieredCompactionStrategy: Compaction for Time Series Data](#).

Starting compaction

You can configure these types of compaction to run periodically:

- [SizeTieredCompactionStrategy](#)
For write-intensive workloads
- [LeveledCompactionStrategy](#)
For read-intensive workloads
- [DateTieredCompactionStrategy](#)
For [time series data](#) and [expiring \(TTL\) data](#)

You can manually start compaction using the `nodetool compact` command.

How is data updated?

Inserting a duplicate primary key is treated as an upsert. An upsert writes a new record to the database if the data didn't exist before. If the data for that primary key already exists, a new record is written with a more recent timestamp. If the data is retrieved during a read, only the most recent is retrieved; older timestamped data will be marked for deletion. The net effect is similar to swapping overwriting the old value with the new value, even though Cassandra does not overwrite data. Eventually, the updates are streamed to disk using sequential I/O and stored in a new SSTable. During an update, Cassandra timestamps and writes columns to disk using the [write path](#). If multiple versions of the column exist in the memtable, Cassandra flushes only the newer version of the column to disk, as described in the [Compaction](#) section.

How is data deleted?

Cassandra deletes data differently than a relational database does. A relational database might spend time scanning through data looking for expired data and throwing it away or an administrator might have to partition expired data by month. Data in a Cassandra column can have an optional expiration date called TTL (time to live). Use CQL to [set the TTL](#) in seconds for data. Cassandra marks TTL data with a tombstone after the requested amount of time has expired. A tombstone exists for [gc_grace_seconds](#). After data is marked with a tombstone, the data is automatically removed during normal [compaction](#).

Facts about deleted data to consider are:

- Cassandra does not immediately remove data marked for deletion from disk. The deletion occurs during compaction.
- If you use the [SizeTieredCompactionStrategy](#) or [DateTieredCompactionStrategy](#), you can drop data immediately by [manually starting the compaction process](#). Before doing so, understand the disadvantages of the process. If you force compaction, one potentially very large SSTable is created from all the data. Another compaction will not be triggered for a long time. The data in the SSTable created during the forced compaction can grow very stale during this long period of non-compaction.
- Deleted data can reappear if you do not do [repair](#) routinely.

Marking data with a tombstone signals Cassandra to retry sending a delete request to a replica that was down at the time of delete. If the replica comes back up within the grace period of time, it eventually

receives the delete request. However, if a node is down longer than the grace period, the node can miss the delete because the tombstone disappears after `gc_grace_seconds`. Cassandra always attempts to replay missed updates when the node comes back up again. After a failure, it is a best practice to run node repair to [repair inconsistencies](#) across all of the replicas when bringing a node back into the cluster. If the node doesn't come back within `gc_grace_seconds`, remove the node, delete the node's data, and bootstrap it again.

How are indexes stored and updated?

Secondary indexes are used to filter a table for data stored in non-primary key columns. For example, a table storing user IDs, names, and ages using the user ID as the primary key might have a secondary index on the age to allow queries by age. Querying to match a non-primary key column is an anti-pattern, as querying should always result in a continuous slice of data retrieved from the table. Non-primary keys play no role in ordering the data in storage, subsequently querying for a particular value of a non-primary key column results in scanning all partitions. Scanning all partitions generally results in a prohibitive read latency, and is not allowed.

Secondary indexes can be built for a column in a table. These indexes are stored locally on each node in a hidden table and built in a background process. If a secondary index is used in a query that is not restricted to a particular partition key, the query will have prohibitive read latency because all nodes will be queried. A query with these parameters is only allowed if the query option `ALLOW FILTERING` is used. This option is not appropriate for production environments. If a query includes both a partition key condition and a secondary index column condition, the query will be successful because the query can be directed to a single node partition.

This technique, however, does not guarantee trouble-free indexing, so know [when and when not to use an index](#).

As with relational databases, keeping indexes up to date uses processing time and resources, so unnecessary indexes should be avoided. When a column is updated, the index is updated as well. If the old column value still exists in the memtable, which typically occurs when updating a small set of rows repeatedly, Cassandra removes the corresponding obsolete index entry; otherwise, the old entry remains to be purged by compaction. If a read sees a stale index entry before compaction purges it, the reader thread invalidates it.

How is data read?

To satisfy a read, Cassandra must combine results from the active memtable and potentially multiple SSTables.

Cassandra processes data at several stages on the read path to discover where the data is stored, starting with the data in the memtable and finishing with SSTables:

- Check the memtable
- Check row cache, if enabled
- Checks Bloom filter
- Checks partition key cache, if enabled
- Goes directly to the compression offset map if a partition key is found in the partition key cache, or checks the partition summary if not

If the partition summary is checked, then the partition index is accessed

- Locates the data on disk using the compression offset map
- Fetches the data from the SSTable on disk

Figure: Read request flow

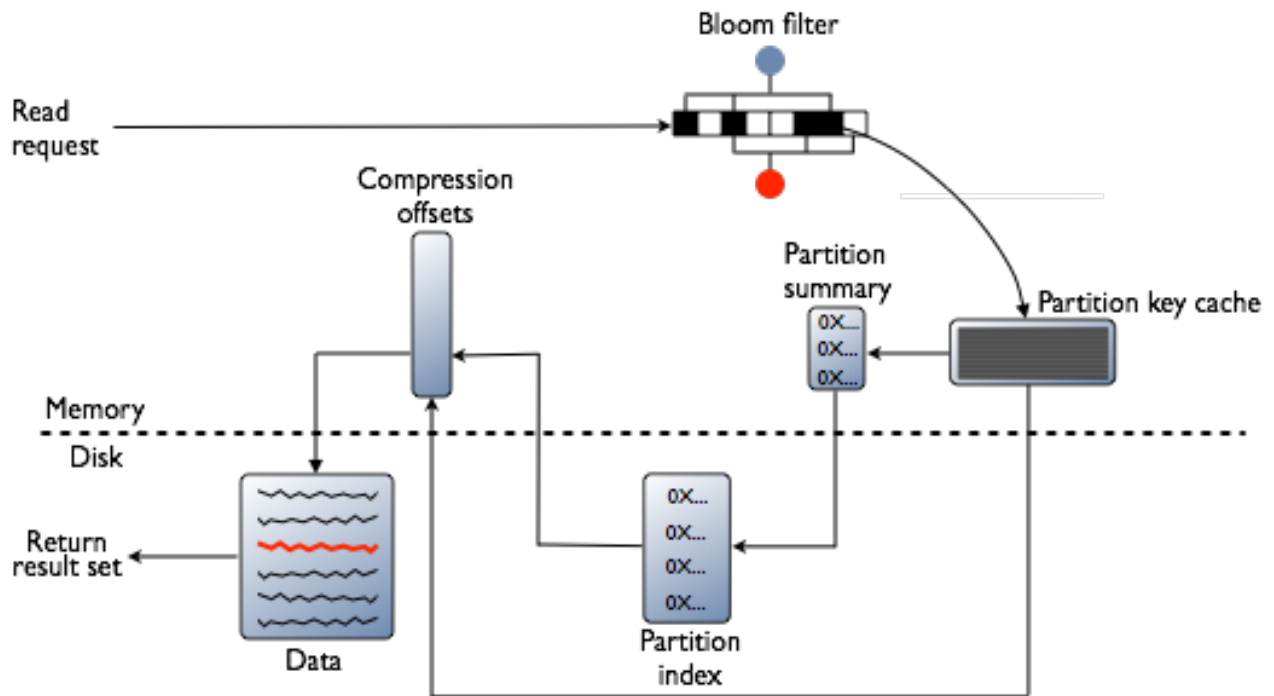
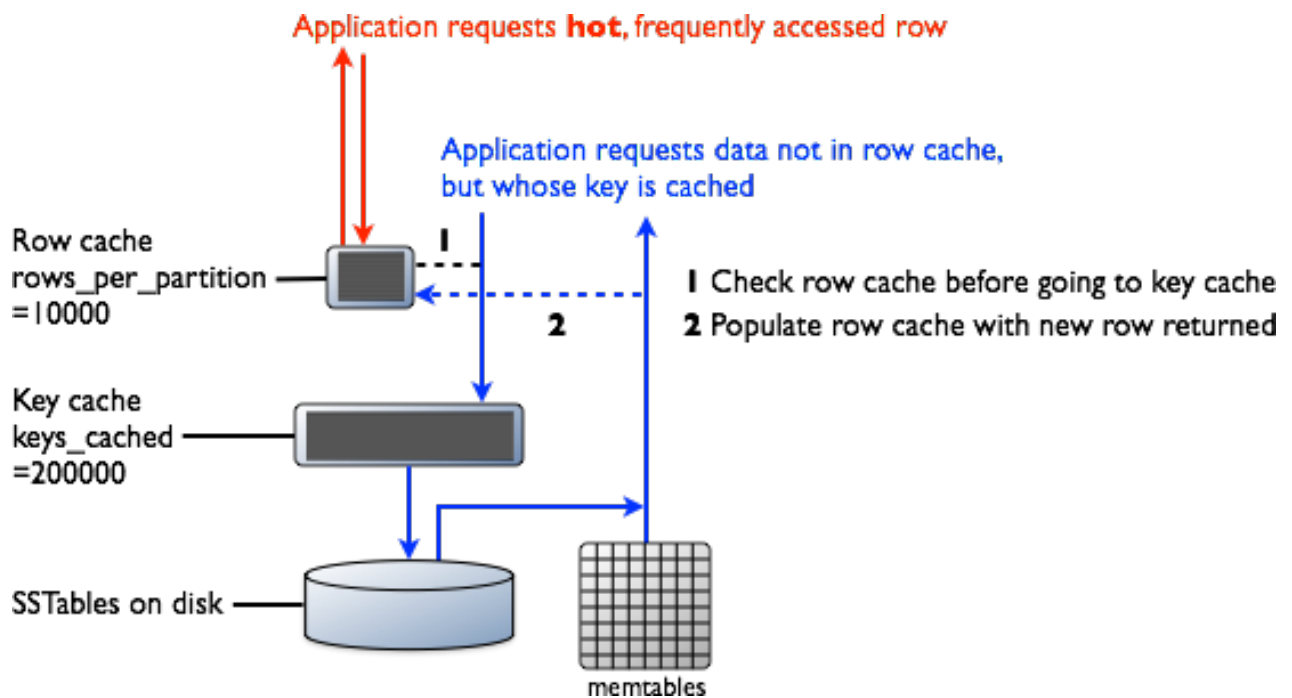


Figure: Row cache and Key cache request flow



Memtable

If the memtable has the desired partition data, then the data is read and then merged with the data from the SSTables. The SSTable data is accessed as shown in the following steps.

Row Cache

Typical of any database, reads are fastest when the most in-demand data fits into memory. The row cache, if enabled, stores a subset of the partition data stored on disk in the SSTables in memory. In Cassandra

2.2 and later, it is stored in fully off-heap memory using a new implementation that relieves garbage collection pressure in the JVM. The subset stored in the row cache use a configurable amount of memory for a specified period of time. A useful feature is that the number of rows to be stored in row cache can be configured, making a "Last 10 Items" query very fast to read. If row cache is enabled, desired partition data is read from the row cache, potentially saving two seeks to disk for the data. The rows stored in row cache are frequently accessed rows that are merged and saved to the row cache from the SSTables as they are accessed. After storage, the data is available to later queries. The row cache is not write-through. If a write comes in for the row, the cache for that row is invalidated and is not cached again until the row is read. If the desired partition data is not found in the row cache, then the Bloom filter is checked.

Note: The row cache must store an entire internal row of data in memory, so if the partition data is larger than the memory allocated for row cache, the row will not cache. The row cache uses LRU (least-recently-used) eviction to reclaim memory when the cache has filled up.

The row cache size is configurable, as is the number of rows to store.

Bloom Filter

First, Cassandra checks the Bloom filter to discover which SSTables are likely to have the request partition data. The Bloom filter is stored in off-heap memory. Each SSTable has a Bloom filter associated with it. A Bloom filter can establish that a SSTable does not contain certain partition data. A Bloom filter can also find the likelihood that partition data is stored in a SSTable. It speeds up the process of partition key lookup by narrowing the pool of keys. However, because the Bloom filter is a probabilistic function, it can result in false positives. Not all SSTables identified by the Bloom filter will have data. If the Bloom filter does not rule out an SSTable, Cassandra checks the [partition key cache](#)

The Bloom filter grows to approximately 1-2 GB per billion partitions. In the extreme case, you can have one partition per row, so you can easily have billions of these entries on a single machine. The Bloom filter is tunable if you want to trade memory for performance.

Partition Key Cache

The partition key cache, if enabled, stores a cache of the partition index in off-heap memory. The key cache uses a small, configurable amount of memory, and each "hit" saves one seek during the read operation. If a partition key is found in the key cache can go directly to the compression offset map to find the compressed block on disk that has the data. The partition key cache functions better once warmed, and can greatly improve over the performance of cold-start reads, where the key cache doesn't yet have or has purged the keys stored in the key cache. It is possible to limit the number of partition keys saved in the key cache, if memory is very limited on a node. If a partition key is not found in the key cache, then the partition summary is searched.

The partition key cache size is configurable, as are the number of partition keys to store in the key cache.

Partition Summary

The partition summary is an off-heap in-memory structure that stores a sampling of the partition index. A partition index contains all partition keys, whereas a partition summary samples every X keys, and maps the location of every Xth key's location in the index file. For example, if the partition summary is set to sample every 20 keys, it will store the location of the first key as the beginning of the SSTable file, the 20th key and its location in the file, and so on. While not as exact as knowing the location of the partition key, the partition summary can shorten the scan to find the partition data location. After finding the range of possible partition key values, the partition index is searched.

By configuring the sample frequency, you can trade memory for performance, as the more granularity the partition summary has, the more memory it will use. The sample frequency is changed using the [index interval](#) property in the table definition.

Partition Index

The partition index resides on disk and stores an index of all partition keys mapped to their offset. If the partition summary has been checked for a range of partition keys, now the search passes to the partition index to seek the location of the desired partition key. A single seek and sequential read of the columns over the passed-in range is performed. Using the information found, the partition index now goes to the compression offset map to find the compressed block on disk that has the data. If the partition index must be searched, two seeks to disk will be required to find the desired data.

Compression offset map

The compression offset map stores pointers to the exact location on disk that the desired partition data will be found. It is stored in off-heap memory and is accessed by either the partition key cache or the partition index. The desired compressed partition data is fetched from the correct SSTable(s) once the compression offset map identifies the disk location. The query receives the result set.

Note: Within a partition, all rows are not equally expensive to query. The very beginning of the partition (the first rows, clustered by your key definition) is slightly less expensive to query because there is no need to consult the partition-level index.

The compression offset map grows to 1-3 GB per terabyte compressed. The more you compress data, the greater number of compressed blocks you have and the larger the compression offset table. Compression is enabled by default even though going through the compression offset map consumes CPU resources. Having compression enabled makes the page cache more effective, and typically, almost always pays off.

How do write patterns affect reads?

It is important to consider how the write operations will affect the read operations in the cluster. The type of [compaction strategy](#) Cassandra performs on your data is configurable and can significantly affect read performance. Using the `SizeTieredCompactionStrategy` or `DateTieredCompactionStrategy` tends to cause data fragmentation when rows are frequently updated. The `LeveledCompactionStrategy` (LCS) was designed to prevent fragmentation under this condition.

Data consistency

How are consistent read and write operations handled?

Consistency refers to how up-to-date and synchronized a row of Cassandra data is on all of its replicas. Using [repair operations](#), Cassandra data will eventually be consistent in all replicas. Repairs work to decrease the variability in replica data, but at a given time, stale data can be present. Cassandra is a AP system according to the [CAP theorem](#), providing high availability and partition tolerance. Cassandra does have flexibility in its configuration, though, and can perform more like a CP (consistent and partition tolerant) system according to the CAP theorem, depending on the application requirements. Two consistency features are tunable consistency and linearizable consistency.

Tunable consistency

To ensure that data is written and read correctly, Cassandra extends the concept of [eventual consistency](#) by offering tunable consistency. Tunable consistency allows individual read or write operations to be as strongly consistent as required by the client application. The consistency level of each read or write operation can be set, so that the data returned is more or less consistent, based on need. The tradeoff between operation latency and consistency level can be tuned down to the per-operation level, or set globally for a cluster or data center. Using tunable consistency, Cassandra can act more like a CP (consistent and partition tolerant) or AP (highly available and partition tolerant) system according to the CAP theorem, depending on the application requirements.

The consistency level determines only the number of replicas that need to acknowledge the read or write operation success to the client application. For read operations, the read consistency level specifies how many replicas must respond to a read request before returning data to the client application. Read operations will use [read repair](#) to update stale data in the background if discovered during a read operation.

For write operations, the write consistency level specified how many replicas must respond to a write request before the write is considered successful. Even at low consistency levels, Cassandra writes to all replicas of the partition key, including replicas in other data centers. The write consistency level just specifies when the coordinator can report to the client application that the write operation is considered completed. Write operations will use [hinted handoffs](#) to ensure the writes are completed when replicas are down or otherwise not responsive to the write request.

Typically, a client specifies a consistency level that is less than the replication factor specified by the keyspace. Another common practice is to write at a consistency level of QUORUM and read at a consistency level of QUORUM. The choices made depend on the client application's needs, and Cassandra provides maximum flexibility for application design.

Linearizable consistency

In ACID terms, linearizable consistency (or serial consistency) is a serial (immediate) isolation level for [lightweight transactions](#). Cassandra does not use locking or transactional dependencies when concurrently updating multiple rows or tables. However, sometimes operations must be performed in sequence and not interrupted by other operations. For example, a typical use case is the creation of user accounts, where a duplication or overwrite will have serious consequences. Linearizable consistency is not required for all aspects of the user's account, but the unique identifier like the userID or email address that claims the account is treated differently. Such a serial operation is implemented in Cassandra with the Paxos consensus protocol, which uses a quorum-based algorithm. Lightweight transactions can be implemented without the need for a master database or two-phase commit process.

Lightweight transaction write operations use the serial consistency level for Paxos consensus and the regular consistency level for the write to the table. For more information, see [Lightweight Transactions](#).

Calculating consistency

Reliability of read and write operations depends on the consistency used to verify the operation. Strong consistency can be guaranteed when the following condition is true:

$$R + W > N$$

where

- R is the consistency level of read operations
- W is the consistency level of write operations
- N is the number of replicas

If the replication factor is 3, then the consistency level of the reads and writes combined must be at least 4. For example, read operations using 2 out of 3 replicas to verify the value, and write operations using 2 out of 3 replicas to verify the value will result in strong consistency. If fast write operations are required, but strong consistency is still desired, the write consistency level is lowered to 1, but now read operations have to verify a matched value on all 3 replicas. Writes will be fast, but reads will be slower.

Eventual consistency occurs if the following condition is true:

$$R + W \leq N$$

where

- R is the consistency level of read operations
- W is the consistency level of write operations

- N is the number of replicas

If the replication factor is 3, then the consistency level of the reads and writes combined are 3 or less. For example, read operations using **QUORUM** (2 out of 3 replicas) to verify the value, and write operations using **ONE** (1 out of 3 replicas) to do fast writes will result in eventual consistency. All replicas will receive the data, but read operations are more vulnerable to selecting data before all replicas write the data.

Additional consistency examples:

- You do a write at **ONE**, the replica crashes one second later. The other messages are not delivered. The data is lost.
- You do a write at **ONE**, and the operation times out. Future reads can return the old or the new value. You will not know the data is incorrect.
- You do a write at **ONE**, and one of the other replicas is down. The node comes back online. The application will get old data from that node until the node gets the correct data or a read repair occurs.
- You do a write at **QUORUM**, and then a read at **QUORUM**. One of the replicas dies. You will always get the correct data.

How are Cassandra transactions different from RDBMS transactions?

Cassandra does not use RDBMS ACID transactions with rollback or locking mechanisms, but instead offers atomic, isolated, and durable transactions with eventual/tunable consistency that lets the user decide how strong or eventual they want each transaction's consistency to be.

As a non-relational database, Cassandra does not support joins or foreign keys, and consequently does not offer consistency in the ACID sense. For example, when moving money from account A to B the total in the accounts does not change. Cassandra supports atomicity and isolation at the row-level, but trades transactional isolation and atomicity for high availability and fast write performance. Cassandra writes are durable.

Atomicity

In Cassandra, a write is atomic at the partition-level, meaning inserting or updating columns in a row is treated as one write operation. A delete operation is also performed atomically.

For example, if using a write consistency level of **QUORUM** with a replication factor of 3, Cassandra will replicate the write to all nodes in the cluster and wait for acknowledgement from two nodes. If the write fails on one of the nodes but succeeds on the other, Cassandra reports a failure to replicate the write on that node. However, the replicated write that succeeds on the other node is not automatically rolled back.

Cassandra uses client-side timestamps to determine the most recent update to a column. The latest timestamp always wins when requesting data, so if multiple client sessions update the same columns in a row concurrently, the most recent update is the one seen by readers.

Isolation

Full row-level isolation is in place, which means that writes to a row are isolated to the client performing the write and are not visible to any other use until they are complete. Delete operations are performed in isolation. All updates in a batch operation belonging to a given partition key are performed in isolation.

Durability

Writes in Cassandra are durable. All writes to a replica node are recorded both in memory and in a commit log on disk before they are acknowledged as a success. If a crash or server failure occurs before the memtables are flushed to disk, the commit log is replayed on restart to recover any lost writes. In addition to the local durability (data immediately written to disk), the replication of data on other nodes strengthens durability.

You can manage the local durability to suit your needs for consistency using the `commitlog_sync` option in the `cassandra.yaml` file. Set the option to either `periodic` or `batch`.

The location of the `cassandra.yaml` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra.yaml</code>
Tarball installations	<code>install_location/resources/cassandra/conf/cassandra.yaml</code>
Windows installations	

How do I accomplish lightweight transactions with linearizable consistency?

Distributed databases present a unique challenge when data must be strictly read and written in sequential order. In transactions for creating user accounts or transferring money, race conditions between two potential writes must be regulated to ensure that one write precedes the other. In Cassandra, the Paxos consensus protocol is used to implement lightweight transactions that can handle concurrent operations.

The Paxos protocol is implemented in Cassandra with linearizable consistency, that is sequential consistency with real-time constraints. Linearizable consistency ensures transaction isolation at a level similar to the serializable level offered by RDBMSs. This type of transaction is known as compare and set (CAS); replica data is compared and any data found to be out of date is set to the most consistent value. In Cassandra, the process combines the Paxos protocol with normal read and write operations to accomplish the compare and set operation.

The Paxos protocol is implemented as a series of phases:

1. Prepare/Promise
2. Read/Results
3. Propose/Accept
4. Commit/Acknowledge

These phases are actions that take place between a proposer and acceptors. Any node can be a proposer, and multiple proposers can be operating at the same time. For simplicity, this description will use only one proposer. A proposer prepares by sending a message to a quorum of acceptors that includes a proposal number. Each acceptor promises to accept the proposal if the proposal number is the highest they have received. Once the proposer receives a quorum of acceptors who promise, the value for the proposal is read from each acceptor and sent back to the proposer. The proposer figures out which value to use and proposes the value to a quorum of the acceptors along with the proposal number. Each acceptor accepts the proposal with a certain number if and only if the acceptor is not already promised to a proposal with a high number. The value is committed and acknowledged as a Cassandra write operation if all the conditions are met.

These four phases require four round trips between a node proposing a lightweight transaction and any cluster replicas involved in the transaction. Performance will be affected. Consequently, reserve lightweight transactions for situations where concurrency must be considered.

Lightweight transactions will block other lightweight transactions from occurring, but will not stop normal read and write operations from occurring. Lightweight transactions use a timestamping mechanism different than for normal operations and mixing LWTs and normal operations can result in errors. If lightweight transactions are used to write to a row within a partition, only lightweight transactions for both read and write operations should be used. This caution applies to all operations, whether individual or batched. For example, the following series of operations can fail:

```
DELETE ...
INSERT .... IF NOT EXISTS
SELECT ....
```

The following series of operations will work:

```
DELETE ... IF EXISTS
INSERT .... IF NOT EXISTS
SELECT .....
```

A [SERIAL consistency level](#) allows reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a SERIAL read finds an uncommitted transaction in progress, Cassandra performs a read repair as part of the commit.

How do I discover consistency level performance?

Before changing the consistency level on read and write operations, discover how your CQL commands are performing using the `TRACING` command in CQL. Using `cqlsh`, you can vary the consistency level and trace read and write operations. The tracing output includes latency times for the operations.

The CQL documentation includes a [tutorial](#) comparing consistency levels.

How is the consistency level configured?

Consistency levels in Cassandra can be configured to manage availability versus data accuracy. You can configure consistency on a cluster, data center, or per individual read or write operation. Consistency among participating nodes can be set globally and also controlled on a per-operation basis. Within `cqlsh`, use [CONSISTENCY](#), to set the consistency level for all queries in the current `cqlsh` session. For programming client applications, set the consistency level using an appropriate driver. For example, using the Java driver, call `QueryBuilder.insertInto` with `setConsistencyLevel` to set a per-insert consistency level.

The consistency level defaults to `ONE` for all write and read operations.

Write consistency levels

This table describes the write consistency levels in strongest-to-weakest order.

Table: Write Consistency Levels

Level	Description	Usage
ALL	A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition.	Provides the highest consistency and the lowest availability of any other level.
EACH_QUORUM	Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in <i>each</i> data centers.	Used in multiple data center clusters to strictly maintain consistency at the same level in each data center. For example, choose this level if you want a read to fail when a data center is down and the <code>QUORUM</code> cannot be reached on that data center.
QUORUM	A write must be written to the commit log and memtable on a quorum of replica nodes across <i>all</i> data centers.	Used in either single or multiple data center clusters to maintain strong consistency across the cluster. Use if you can tolerate some level of failure.
LOCAL_QUORUM	Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in the same data	Used in multiple data center clusters with a rack-aware replica placement strategy, such as NetworkTopologyStrategy , and a properly configured snitch. Use to maintain

Level	Description	Usage
	center as the coordinator node. Avoids latency of inter-data center communication.	consistency locally (within the single data center). Can be used with SimpleStrategy .
ONE	A write must be written to the commit log and memtable of at least one replica node.	Satisfies the needs of most users because consistency requirements are not stringent.
TWO	A write must be written to the commit log and memtable of at least two replica nodes.	Similar to ONE.
THREE	A write must be written to the commit log and memtable of at least three replica nodes.	Similar to TWO.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.	In a multiple data center clusters, a consistency level of ONE is often desirable, but cross-DC traffic is not. LOCAL_ONE accomplishes this. For security and quality reasons, you can use this consistency level in an offline data center to prevent automatic connection to online nodes in other data centers if an offline node goes down.
ANY	A write must be written to at least one node. If all replica nodes for the given partition key are down, the write can still succeed after a hinted handoff has been written. If all replica nodes are down at write time, an ANY write is not readable until the replica nodes for that partition have recovered.	Provides low latency and a guarantee that a write never fails. Delivers the lowest consistency and highest availability.

Read consistency levels

This table describes read consistency levels in strongest-to-weakest order.

Table: Read Consistency Levels

Level	Description	Usage
ALL	Returns the record after all replicas have responded. The read operation will fail if a replica does not respond.	Provides the highest consistency of all levels and the lowest availability of all levels.
EACH_QUORUM	Strong consistency. A read must be read on a quorum of replica nodes in <i>each</i> data centers.	Used in multiple data center clusters to strictly maintain consistency at the same level in each data center. For example, choose this level if you want a read to fail when a data center is down and the QUORUM cannot be reached on that data center.
QUORUM	Returns the record after a quorum of replicas from all data centers has responded.	Used in either single or multiple data center clusters to maintain strong consistency across the cluster. Ensures strong consistency if you can tolerate some level of failure.

Level	Description	Usage
LOCAL_QUORUM	Returns the record after a quorum of replicas in the current data center as the coordinator node has reported. Avoids latency of inter-data center communication.	Used in multiple data center clusters with a rack-aware replica placement strategy (<code>NetworkTopologyStrategy</code>) and a properly configured snitch. Fails when using <code>SimpleStrategy</code> .
ONE	Returns a response from the closest replica, as determined by the snitch . By default, a read repair runs in the background to make the other replicas consistent.	Provides the highest availability of all the levels if you can tolerate a comparatively high probability of stale data being read. The replicas contacted for reads may not always have the most recent write.
TWO	Returns the most recent data from two of the closest replicas.	Similar to ONE.
THREE	Returns the most recent data from three of the closest replicas.	Similar to TWO.
LOCAL_ONE	Returns a response from the closest replica in the local data center.	Same usage as described in the table about write consistency levels.
SERIAL	Allows reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a <code>SERIAL</code> read finds an uncommitted transaction in progress, it will commit the transaction as part of the read. Similar to <code>QUORUM</code> .	To read the latest value of a column after a user has invoked a lightweight transaction to write to the column, use <code>SERIAL</code> . Cassandra then checks the inflight lightweight transaction for updates and, if found, returns the latest data.
LOCAL_SERIAL	Same as <code>SERIAL</code> , but confined to the data center. Similar to <code>LOCAL_QUORUM</code> .	Used to achieve linearizable consistency for lightweight transactions.

How QUORUM is calculated

The `QUORUM` level writes to the number of nodes that make up a quorum. A quorum is calculated, and then rounded down to a whole number, as follows:

```
quorum = (sum_of_replication_factors / 2) + 1
```

The sum of all the `replication_factor` settings for each data center is the `sum_of_replication_factors`.

```
sum_of_replication_factors = datacenter1_RF + datacenter2_RF + . . . +
datacentern_RF
```

Examples:

- Using a replication factor of 3, a quorum is 2 nodes. The cluster can tolerate 1 replica down.
- Using a replication factor of 6, a quorum is 4. The cluster can tolerate 2 replicas down.
- In a two data center cluster where each data center has a replication factor of 3, a quorum is 4 nodes. The cluster can tolerate 2 replica nodes down.
- In a five data center cluster where two data centers have a replication factor of 3 and three data centers have a replication factor of 2, a quorum is 6 nodes.

The more data centers, the higher number of replica nodes need to respond for a successful operation.

Similar to `QUORUM`, the `LOCAL_QUORUM` level is calculated based on the replication factor of the same data center as the coordinator node. That is, even if the cluster has more than one data center, the quorum is calculated only with local replica nodes.

In `EACH_QUORUM`, every data center in the cluster must reach a quorum based on that data center's replication factor in order for the read or write request to succeed. That is, for every data center in the cluster a quorum of replica nodes must respond to the coordinator node in order for the read or write request to succeed.

Configuring client consistency levels

You can use a `cqlsh` command, `CONSISTENCY`, to set the consistency level for queries in the current `cqlsh` session. For programming client applications, set the consistency level using an appropriate driver. For example, call `QueryBuilder.insertInto` with a `setConsistencyLevel` argument using the Java driver.

How is the serial consistency level configured?

Serial consistency levels in Cassandra can be configured to manage lightweight transaction isolation. Lightweight transactions have two consistency levels defined. The serial consistency level defines the consistency level of the serial phase, or Paxos phase, of lightweight transactions. The learn phase, which defines what read operations will be guaranteed to complete immediately if lightweight writes are occurring uses a normal consistency level. The serial consistency level is ignore for any query that is not a conditional update.

Serial consistency levels

Table: Serial Consistency Levels

Level	Description	Usage
<code>SERIAL</code>	Achieves linearizable consistency for lightweight transactions by preventing unconditional updates.	This consistency level is only for use with lightweight transaction. Equivalent to <code>QUORUM</code> .
<code>LOCAL_SERIAL</code>	Same as <code>SERIAL</code> but confined to the data center. A conditional write must be written to the commit log and memtable on a quorum of replica nodes in the same data center.	Same as <code>SERIAL</code> but used to maintain consistency locally (within the single data center). Equivalent to <code>LOCAL_QUORUM</code> .

How are read requests accomplished?

There are three types of read requests that a coordinator can send to a replica:

- A direct read request
- A digest request
- A background read repair request

The coordinator node contacts one replica node with a direct read request. Then the coordinator sends a digest request to a number of replicas determined by the [consistency level](#) specified by the client. The digest request checks the data in the replica node to make sure it is up to date. Then the coordinator sends a digest request to all remaining replicas. If any replica nodes have out of date data, a background [read repair](#) request is sent. Read repair requests ensure that the requested row is made consistent on all replicas.

For a digest request the coordinator first contacts the replicas specified by the consistency level. The coordinator sends these requests to the replicas that are currently responding the fastest. The nodes

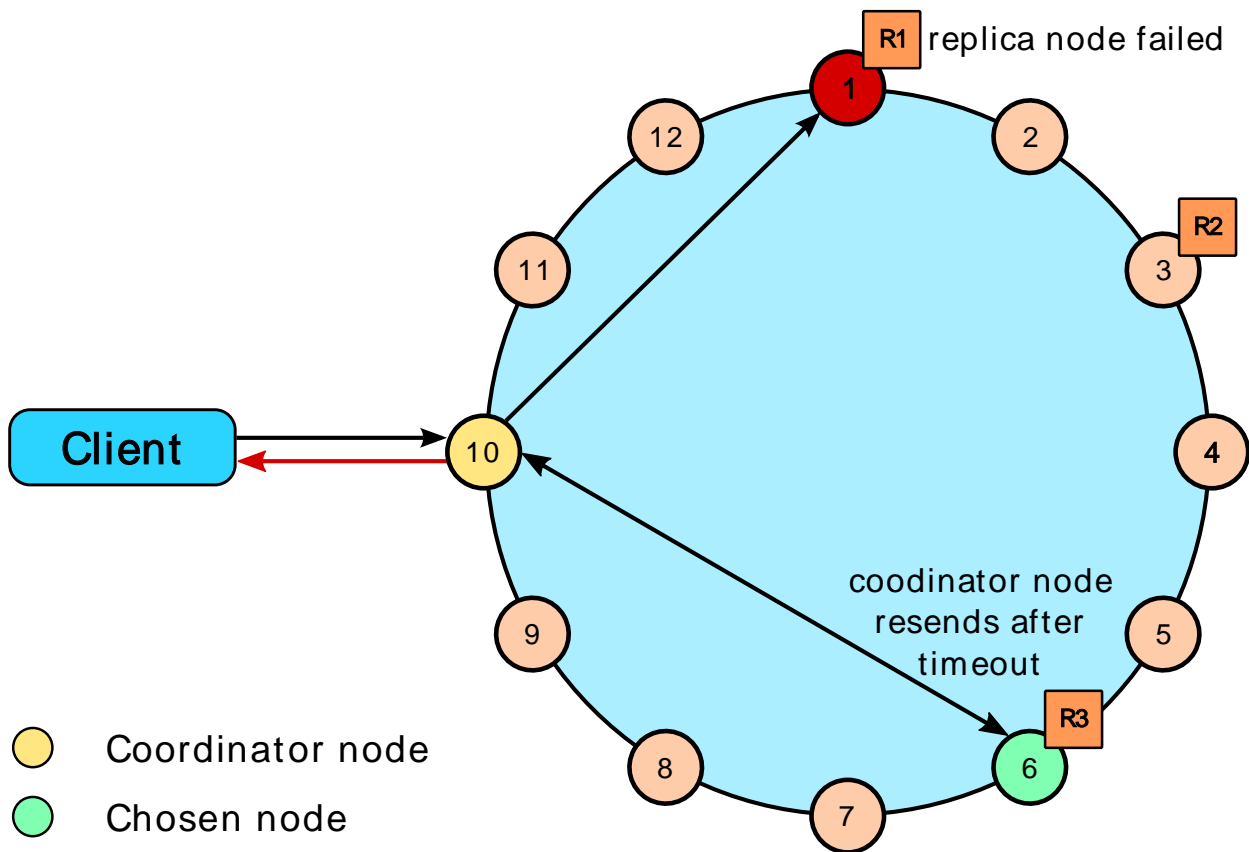
contacted respond with a digest of the requested data; if multiple nodes are contacted, the rows from each replica are compared in memory to see if they are consistent. If they are not, then the replica that has the most recent data (based on the timestamp) is used by the coordinator to forward the result back to the client. To ensure that all replicas have the most recent version of the data, read repair is carried out to update out-of-date replicas.

For illustrated examples of read requests, see the [examples of read consistency levels](#).

Rapid read protection using speculative_retry

Rapid read protection allows Cassandra to still deliver read requests when the originally selected replica nodes are either down or taking too long to respond. If the table has been configured with the [speculative_retry](#) property, the coordinator node for the read request will retry the request with another replica node if the original replica node exceeds a configurable timeout value to complete the read request.

Figure: Recovering from replica node failure with rapid read protection



Examples of read consistency levels

The following diagrams show examples of read requests using these consistency levels:

- [QUORUM](#) in a single data center
- [ONE](#) in a single data center
- [QUORUM](#) in two data centers
- [LOCAL_QUORUM](#) in two data centers
- [ONE](#) in two data centers

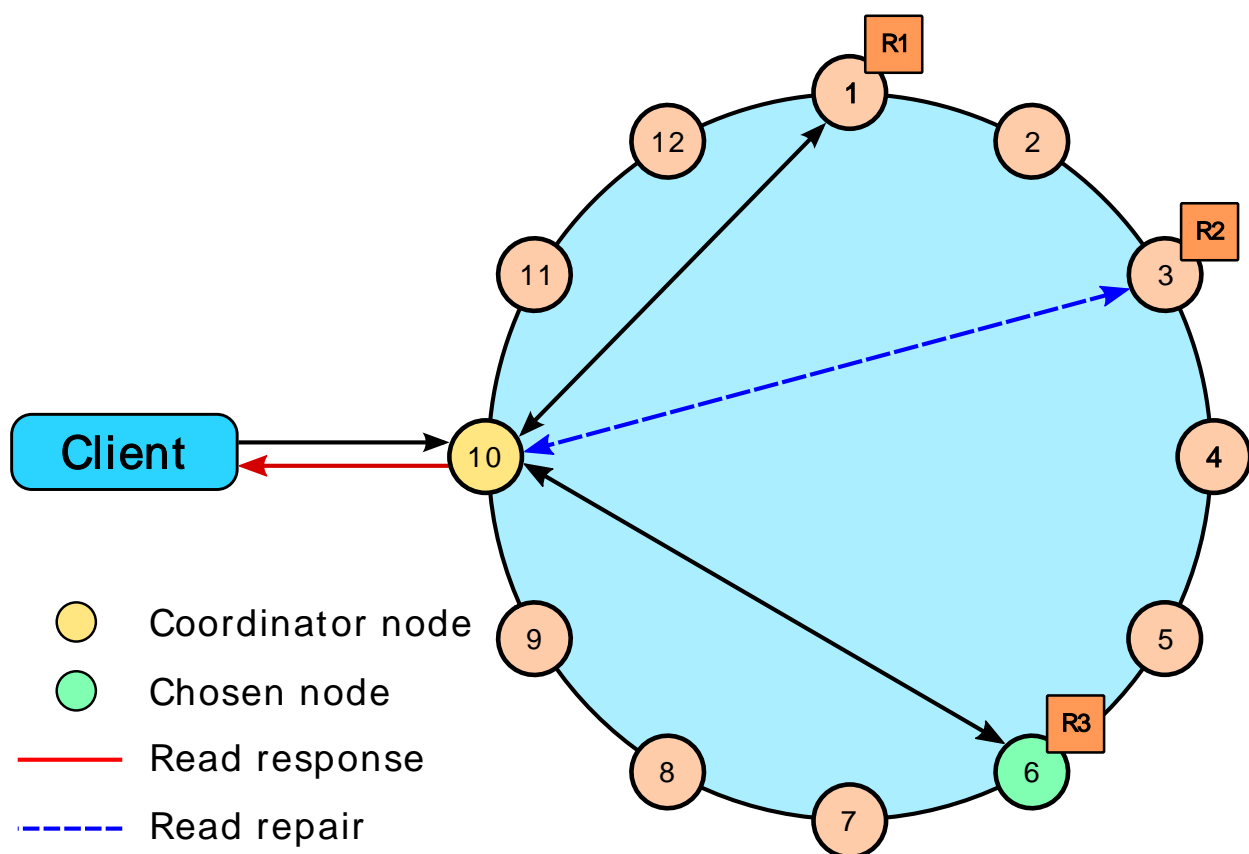
- LOCAL_ONE in two data centers

Rapid read protection diagram shows how the speculative retry table property affects consistency.

A single data center cluster with a consistency level of QUORUM

In a single data center cluster with a replication factor of 3, and a read consistency level of `QUORUM`, 2 of the 3 replicas for the given row must respond to fulfill the read request. If the contacted replicas have different versions of the row, the replica with the most recent version will return the requested data. In the background, the third replica is checked for consistency with the first two, and if needed, a read repair is initiated for the out-of-date replicas.

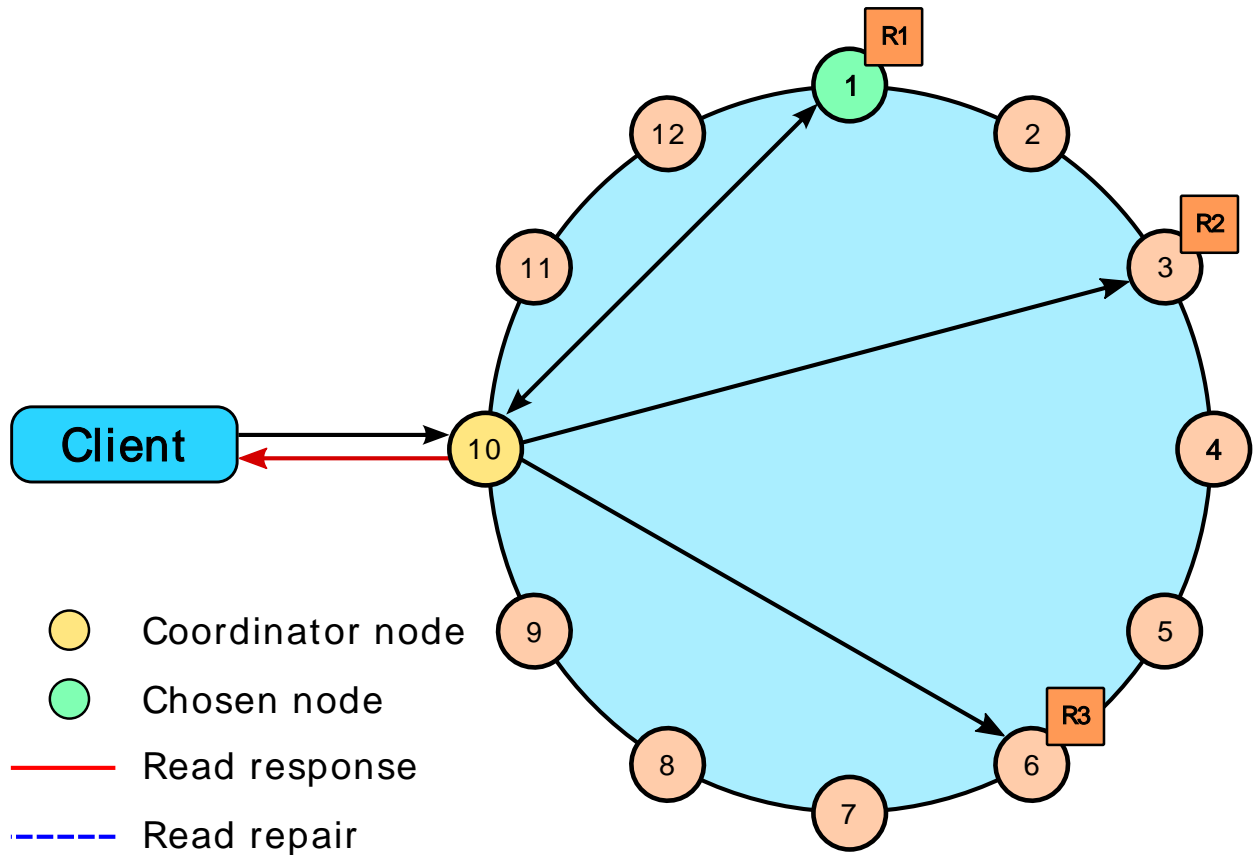
Figure: Single data center cluster with 3 replica nodes and consistency set to QUORUM



A single data center cluster with a consistency level of ONE

In a single data center cluster with a replication factor of 3, and a read consistency level of `ONE`, the closest replica for the given row is contacted to fulfill the read request. In the background a read repair is potentially initiated, based on the `read_repair_chance` setting of the table, for the other replicas.

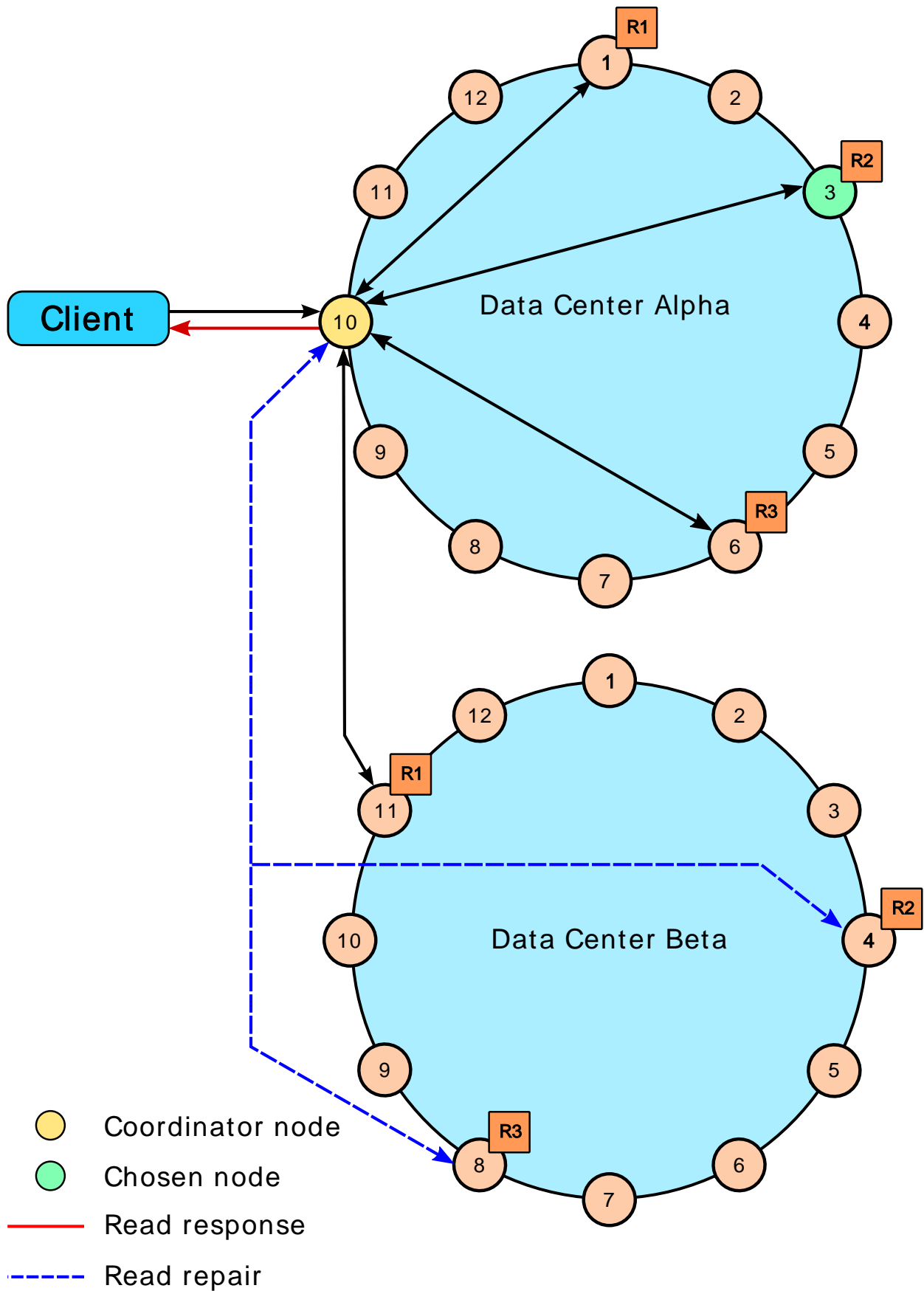
Figure: Single data center cluster with 3 replica nodes and consistency set to ONE



A two data center cluster with a consistency level of QUORUM

In a two data center cluster with a replication factor of 3, and a read consistency of `QUORUM`, 4 replicas for the given row must respond to fulfill the read request. The 4 replicas can be from any data center. In the background, the remaining replicas are checked for consistency with the first four, and if needed, a read repair is initiated for the out-of-date replicas.

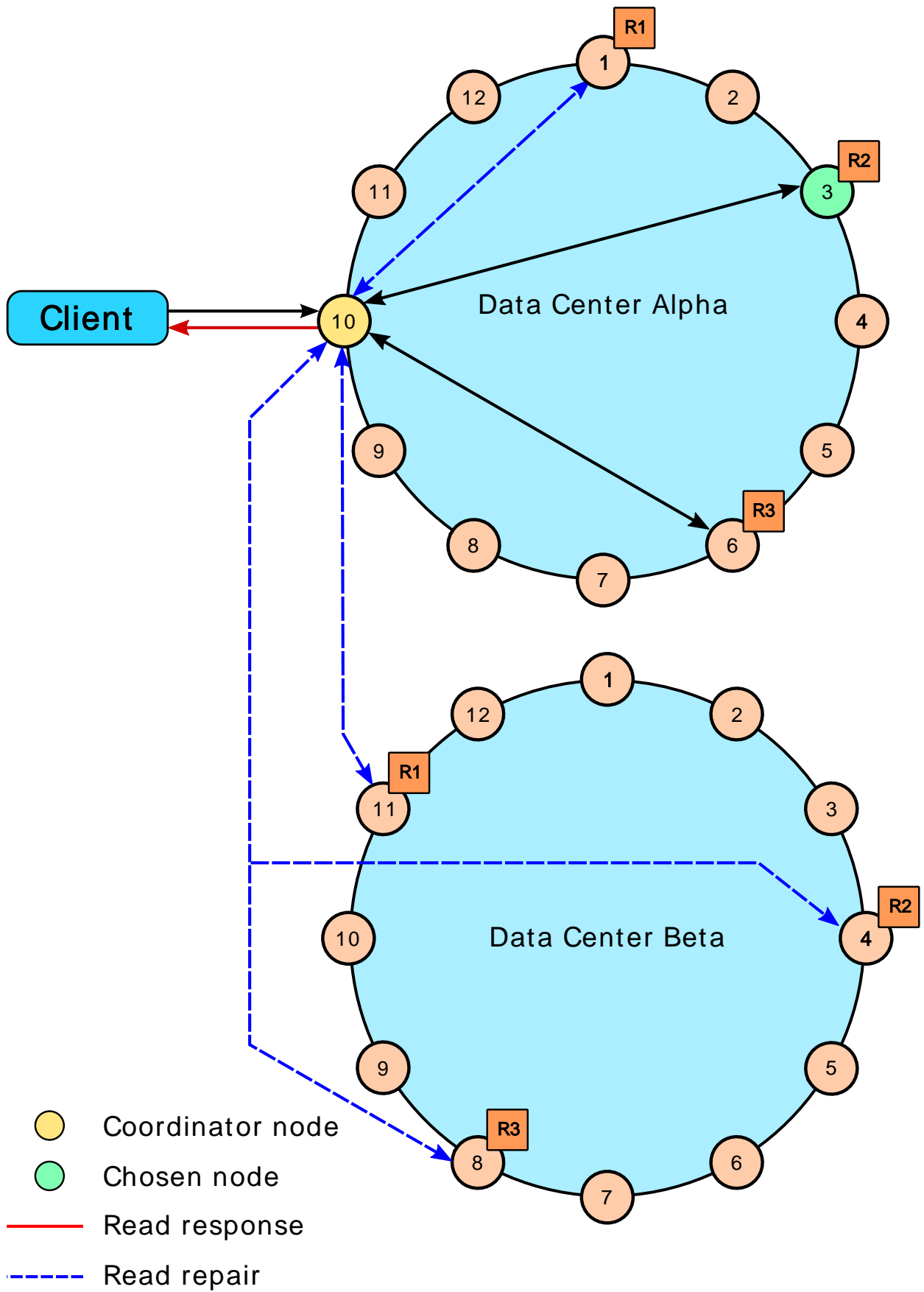
Figure: Multiple data center cluster with 3 replica nodes and consistency level set to QUORUM



A two data center cluster with a consistency level of LOCAL_QUORUM

In a multiple data center cluster with a replication factor of 3, and a read consistency of `LOCAL_QUORUM`, 2 replicas in the same data center as the coordinator node for the given row must respond to fulfill the read request. In the background, the remaining replicas are checked for consistency with the first 2, and if needed, a read repair is initiated for the out-of-date replicas.

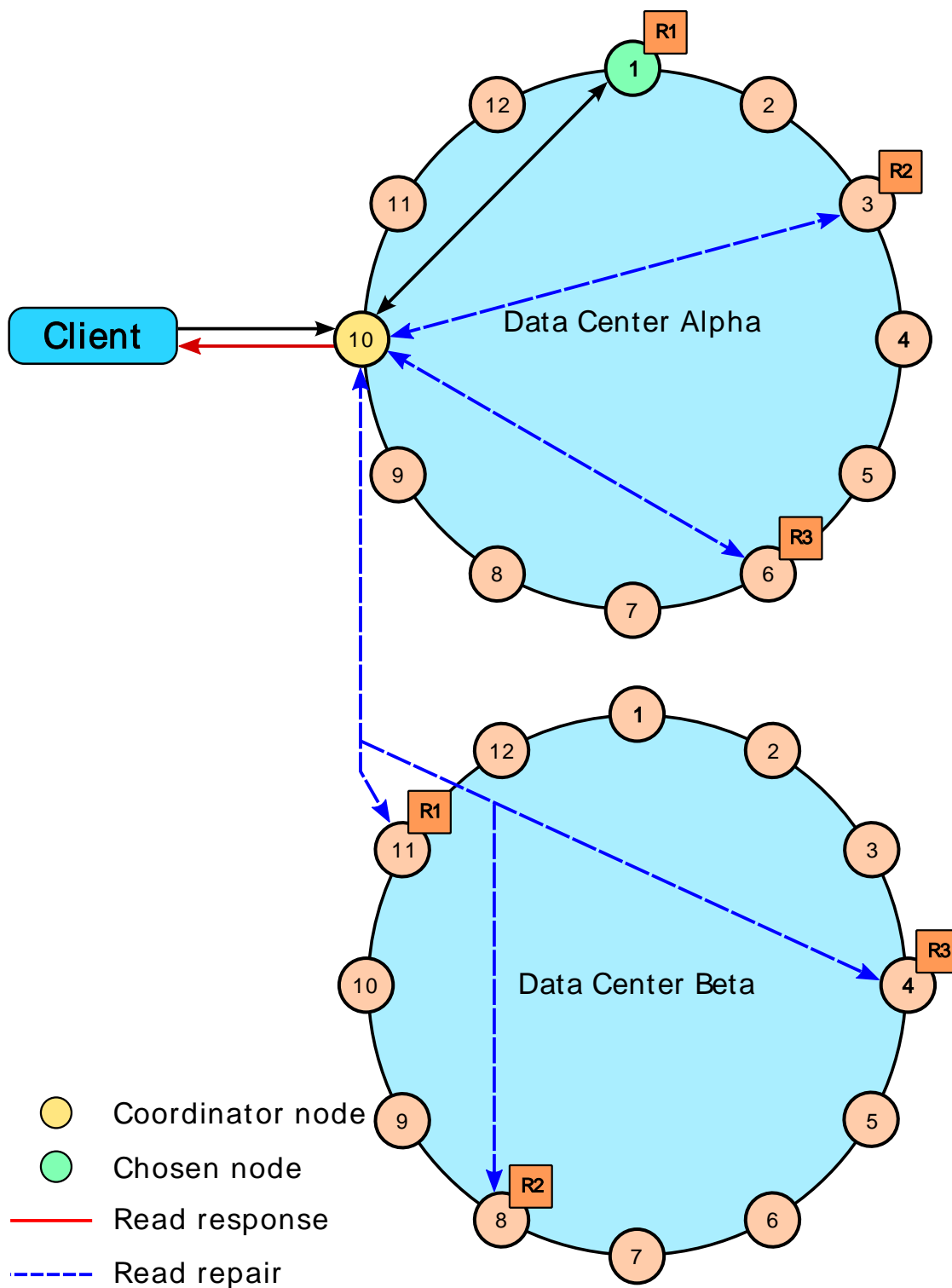
Figure: Multiple data center cluster with 3 replica nodes and consistency set to LOCAL_QUORUM



A two data center cluster with a consistency level of ONE

In a multiple data center cluster with a replication factor of 3, and a read consistency of `ONE`, the closest replica for the given row, regardless of data center, is contacted to fulfill the read request. In the background a read repair is potentially initiated, based on the `read_repair_chance` setting of the table, for the other replicas.

Figure: Multiple data center cluster with 3 replica nodes and consistency set to ONE

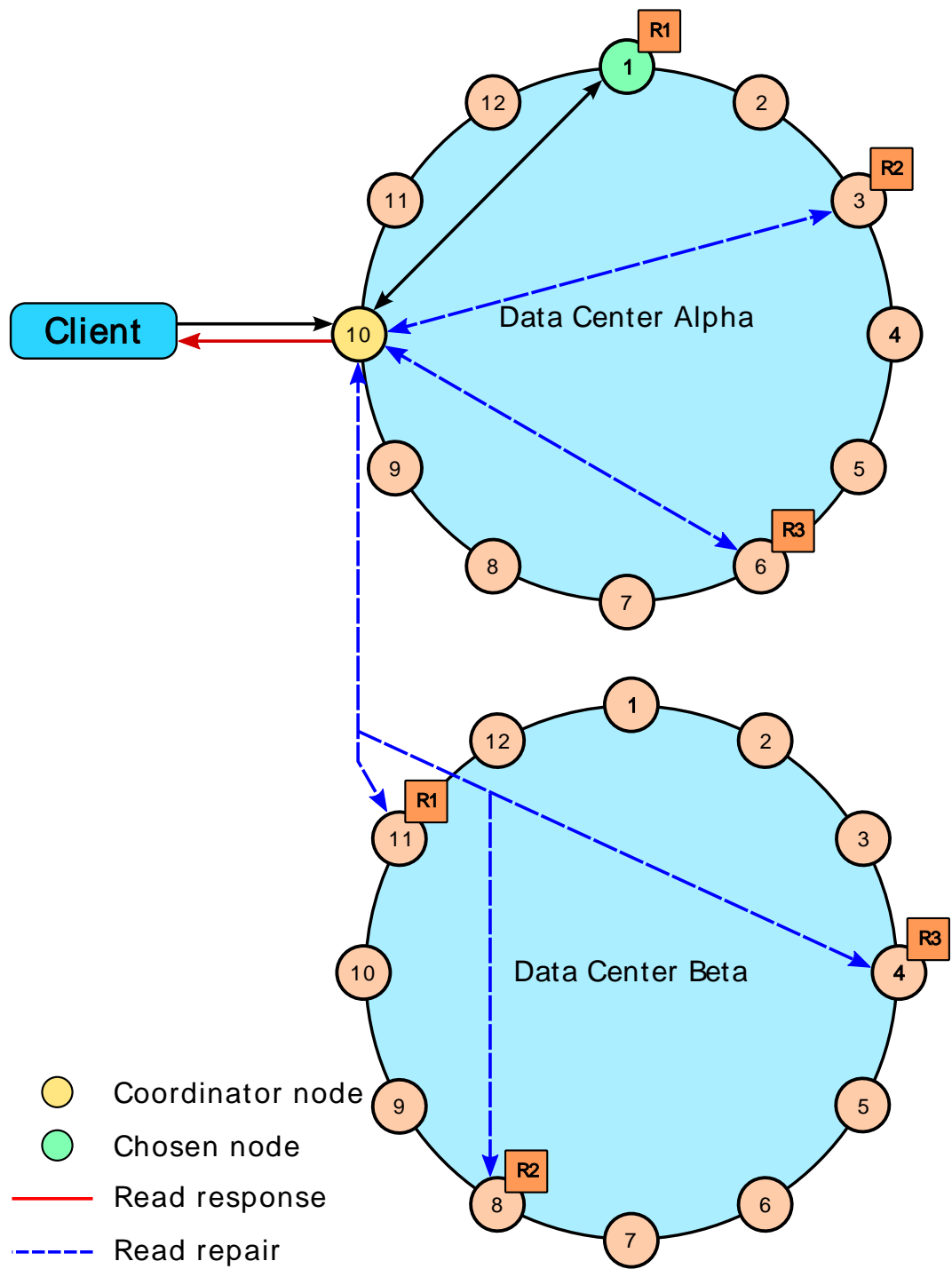


A two data center cluster with a consistency level of `LOCAL_ONE`

In a multiple data center cluster with a replication factor of 3, and a read consistency of `LOCAL_ONE`, the closest replica for the given row in the same data center as the coordinator node is contacted to fulfill the

read request. In the background a read repair is potentially initiated, based on the `read_repair_chance` setting of the table, for the other replicas.

Figure: Multiple data center cluster with 3 replica nodes and consistency set to `LOCAL_ONE`



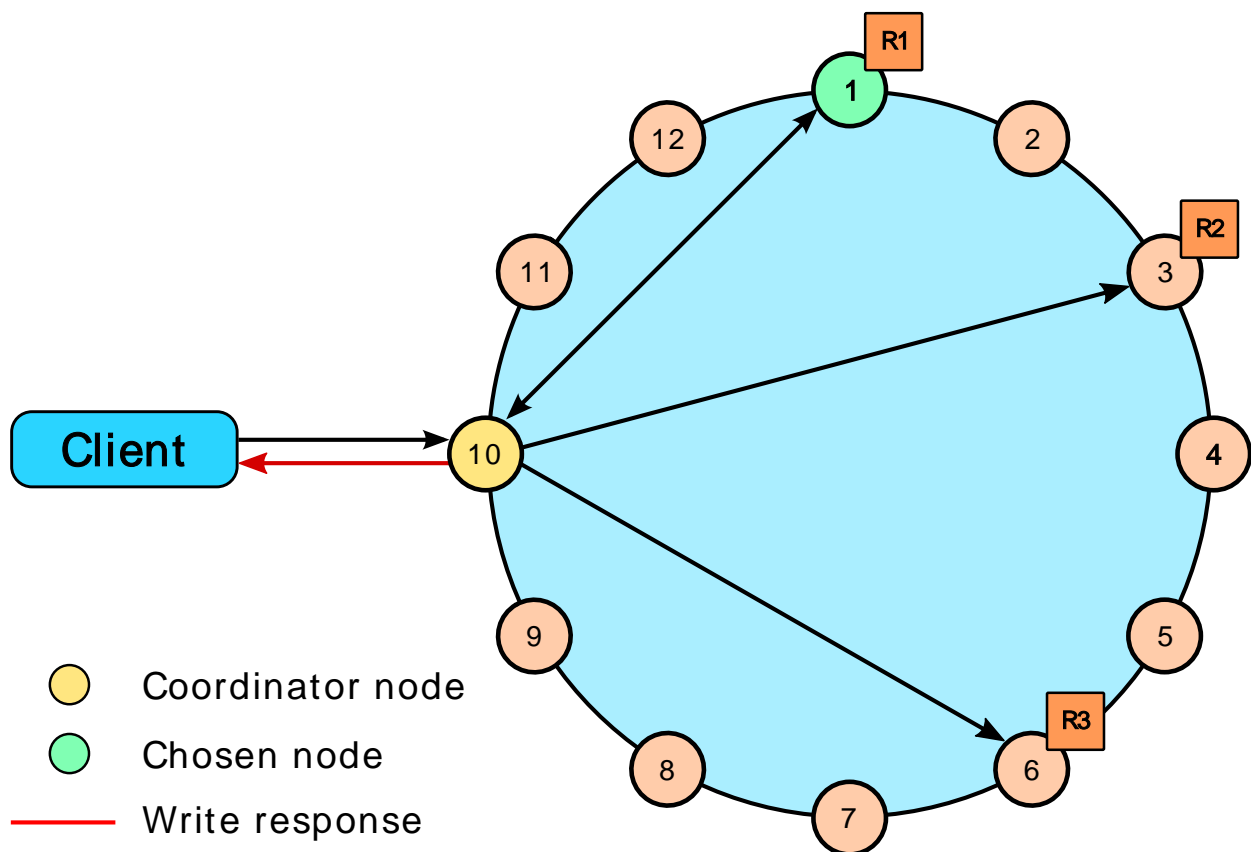
How are write requests accomplished?

The coordinator sends a write request to *all* replicas that own the row being written. As long as all replica nodes are up and available, they will get the write regardless of the [consistency level](#) specified by the client. The write consistency level determines how many replica nodes must respond with a success acknowledgment in order for the write to be considered successful. Success means that the data was written to the commit log and the memtable as described in [how data is written](#).

For example, in a single data center 10 node cluster with a replication factor of 3, an incoming write will go to all 3 nodes that own the requested row. If the write consistency level specified by the client is ONE, the first node to complete the write responds back to the coordinator, which then proxies the success message back to the client. A consistency level of ONE means that it is possible that 2 of the 3 replicas could miss the write if they happened to be down at the time the request was made. If a replica misses a write, Cassandra will make the row consistent later using one of its [built-in repair mechanisms](#): hinted handoff, read repair, or anti-entropy node repair.

That node forwards the write to all replicas of that row. It responds back to the client once it receives a write acknowledgment from the number of nodes specified by the consistency level.

Figure: Single data center cluster with 3 replica nodes and consistency set to ONE



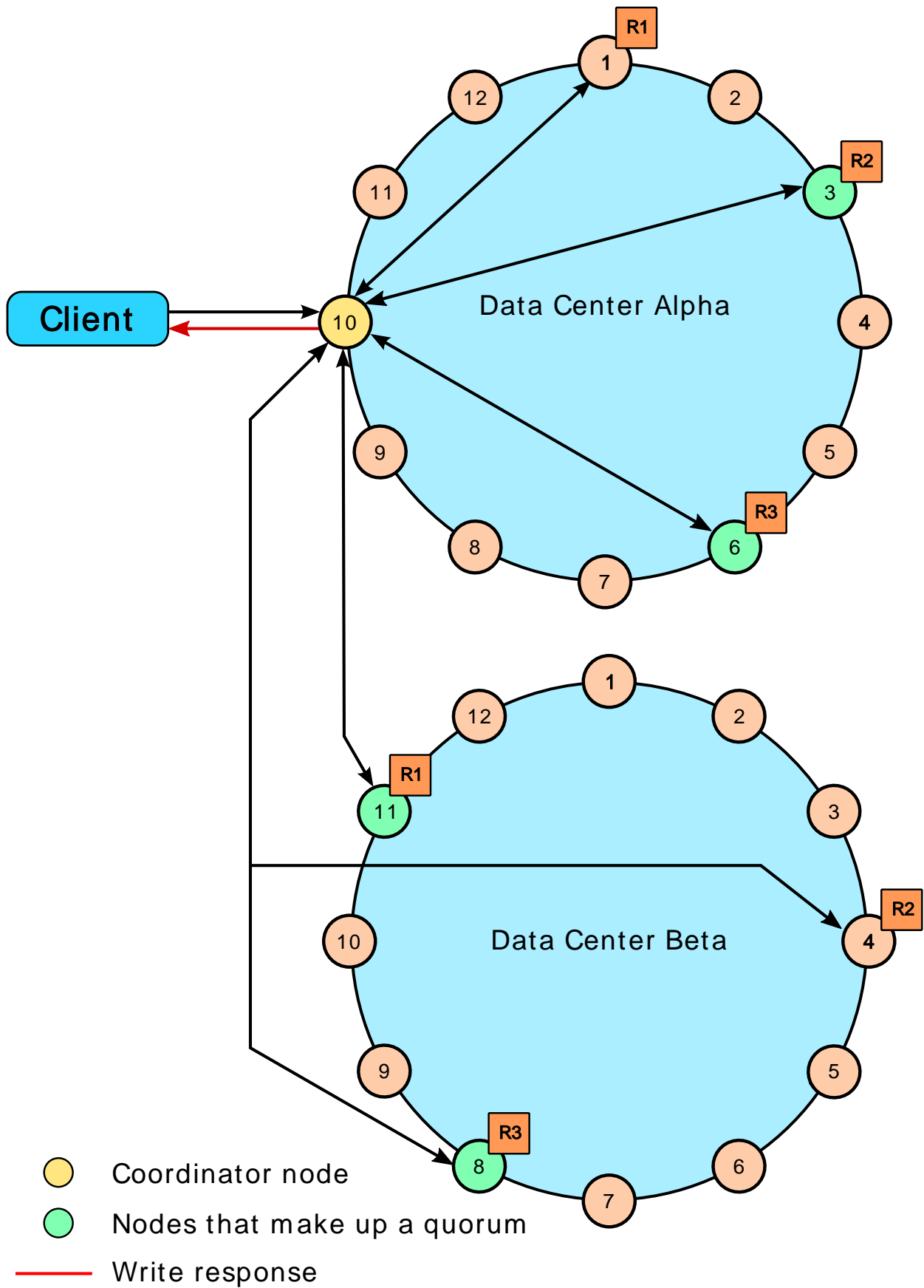
Multiple data center write requests

In multiple data center deployments, Cassandra optimizes write performance by choosing one coordinator node. The coordinator node contacted by the client application forwards the write request to each replica node in each all the data centers.

Database internals

If using a [consistency level](#) of `LOCAL_ONE` or `LOCAL_QUORUM`, only the nodes in the same data center as the coordinator node must respond to the client request in order for the request to succeed. This way, geographical latency does not impact client request response times.

Figure: Multiple data center cluster with 3 replica nodes and consistency set to QUORUM



Planning a cluster deployment

When planning a Cassandra cluster deployment, you should have a good idea of the initial volume of data you plan to store and a good estimate of your typical application workload. The following topics provide information for planning your cluster:

Selecting hardware for enterprise implementations

Choosing appropriate hardware depends on selecting the right balance of the following resources: memory, CPU, disks, number of nodes, and network. [Anti-patterns in Cassandra](#) on page 60 also contains important information about hardware, particularly SAN storage, NAS devices, and NFS.

CAUTION: Do not use a machine suited for development for load testing or production. Failure may result.

Memory

The more memory a Cassandra node has, the better read performance. More RAM also allows memory tables (memtables) to hold more recently written data. Larger memtables lead to a fewer number of SSTables being flushed to disk and fewer files to scan during a read. The ideal amount of RAM depends on the anticipated size of your hot data.

For both dedicated hardware and virtual environments:

- Production: 16GB to 64GB; the minimum is 8GB.
- Development in non-loading testing environments: no less than 4GB.
- For setting Java heap space, see [Tuning Java resources](#).

CPU

Insert-heavy workloads are CPU-bound in Cassandra before becoming memory-bound. (All writes go to the commit log, but Cassandra is so efficient in writing that the CPU is the limiting factor.) Cassandra is highly concurrent and uses as many CPU cores as available:

- Production environments:
 - For dedicated hardware, 8-core CPU processors are the current price-performance sweet spot.
 - For virtual environments, 4 to 8-core CPU processors.
- Development in non-loading testing environments:
 - For dedicated hardware, 2-core CPU processors.
 - For virtual environments, 2-core CPU processors.

Spinning disks versus Solid State Drives

SSDs are recommended for Cassandra. The NAND Flash chips that power SSDs provide extremely low-latency response times for random reads while supplying ample sequential write performance for compaction operations. In recent years, drive manufacturers have improved overall endurance, usually in conjunction with spare (unexposed) capacity. Additionally, because PBW/DWPD ratings are probabilistic estimates based on worst case scenarios, such as random write workloads, and Cassandra does only large sequential writes, drives significantly exceed their endurance ratings. However, it is important to plan for drive failures and have spares available. A large variety of SSDs are available on the market from server vendors and third-party drive manufacturers.

For purchasing SSDs, the best recommendation is to make SSD endurance decisions not based on workload, but on how difficult it is to change drives when they fail. Remember, your data is protected because Cassandra replicates data across the cluster. Buying strategies include:

- If drives are quickly available, buy the cheapest drives that provide the performance you want.
- If it is more challenging to swap the drives, consider higher endurance models, possibly starting in the mid range, and then choose replacements of higher or lower endurance based on the failure rates of the initial model chosen.
- Always buy cheap SSDs and keep several spares online and unused in the servers until the initial drives fail. This way you can replace the drives without touching the server.

DataStax customers that need help in determining the most cost-effective option for a given deployment and workload, should contact their Solutions Engineer or Architect.

Disk space

Disk space depends on usage, so it's important to understand the mechanism. Cassandra writes data to disk when appending data to the `for durability` and when flushing to `data files` for persistent storage. The commit log has a different access pattern (read/writes ratio) than the pattern for accessing data from SSTables. This is more important for spinning disks than for SSDs (solid state drives).

SSTables are periodically compacted. Compaction improves performance by merging and rewriting data and discarding old data. However, depending on the type of [Configuring compaction](#) on page 155 and size of the compactions, during compaction disk utilization and data directory volume temporarily increases. For this reason, you should leave an adequate amount of free disk space available on a node. For large compactions, leave an adequate amount of free disk space available on a node: 50% (worst case) for `SizeTieredCompactionStrategy` and `DateTieredCompactionStrategy`, and 10% for `LeveledCompactionStrategy`. For more information about compaction, see:

- [Compaction](#)
- [The Apache Cassandra storage engine](#)
- [Leveled Compaction in Apache Cassandra](#)
- [When to Use Leveled Compaction](#)
- [DateTieredCompactionStrategy: Compaction for Time Series Data](#) and [Getting Started with Time Series Data Modeling](#)

For information on calculating disk size, see [Calculating usable disk capacity](#).

Recommendations:

Capacity per node

Most workloads work best with a capacity under 500GB to 1TB per node depending on I/O. Maximum recommended capacity for Cassandra 1.2 and later is 3 to 5TB per node for uncompressed data. For Cassandra 1.1, it is 500 to 800GB per node. Be sure to account for [replication](#).

Capacity and I/O

When choosing disks, consider both capacity (how much data you plan to store) and I/O (the write/read throughput rate). Some workloads are best served by using less expensive SATA disks and scaling disk capacity and I/O by adding more nodes (with more RAM).

Number of disks - SATA

Ideally Cassandra needs at least two disks, one for the commit log and the other for the data directories. At a minimum the commit log should be on its own partition.

Commit log disk - SATA

The disk need not be large, but it should be fast enough to receive all of your writes as appends (sequential I/O).

Commit log disk - SSD

Unlike spinning disks, it's all right to store both commit logs and SSTables are on the same mount point.

Data disks

Use one or more disks and make sure they are large enough for the data volume and fast enough to both satisfy reads that are not cached in memory and to keep up with compaction.

RAID on data disks

It is generally not necessary to use RAID for the following reasons:

- Data is replicated across the cluster based on the replication factor you've chosen.
- Starting in version 1.2, Cassandra includes a JBOD (Just a bunch of disks) feature to take care of disk management. Because Cassandra properly reacts to a disk failure either by stopping the affected node or by blacklisting the failed drive, you can deploy Cassandra nodes with large disk arrays without the overhead of RAID 10. You can configure Cassandra to stop the affected node or blacklist the drive according to your availability/consistency requirements. Also see [Recovering from a single disk failure using JBOD](#) on page 136.

RAID on the commit log disk

Generally RAID is not needed for the commit log disk. Replication adequately prevents data loss. If you need extra redundancy, use RAID 1.

Extended file systems

DataStax recommends deploying Cassandra on XFS or ext4. On ext2 or ext3, the maximum file size is 2TB even using a 64-bit kernel. On ext4 it is 16TB.

Because Cassandra can use almost half your disk space for a single file when using [SizeTieredCompactionStrategy](#), use XFS when using large disks, particularly if using a 32-bit kernel. XFS file size limits are 16TB max on a 32-bit kernel, and essentially unlimited on 64-bit.

Number of nodes

Prior to version 1.2, the recommended size of disk space per node was 300 to 500GB. Improvement to Cassandra 1.2 and later, such as JBOD support, virtual nodes (vnodes), off-heap Bloom filters, and parallel leveled compaction (SSD nodes only), allow you to use few machines with multiple terabytes of disk space.

Network

Since Cassandra is a distributed data store, it puts load on the network to handle read/write requests and replication of data across nodes. Be sure that your network can handle traffic between nodes without bottlenecks. You should bind your interfaces to separate Network Interface Cards (NIC). You can use public or private depending on your requirements.

- Recommended bandwidth is 1000 Mbit/s (gigabit) or greater.
- Thrift/native protocols use the [rpc_address](#).
- Cassandra's internal storage protocol uses the [listen_address](#).

Cassandra efficiently routes requests to replicas that are geographically closest to the coordinator node and chooses a replica in the same rack if possible; it always chooses replicas located in the same data center over replicas in a remote data center.

Firewall

If using a firewall, make sure that nodes within a cluster can reach each other. See [Configuring firewall port access](#) on page 102.

Planning an Amazon EC2 cluster

Before planning an Amazon EC2 cluster, please see the [Amazon Web Services EC2 Management Console](#).

DataStax AMI deployments

The [DataStax AMI](#) is available only for Cassandra 2.1 and earlier. It is intended only for a single region and availability zone. For an EC2 cluster that spans multiple regions and availability zones, see [EC2 clusters spanning multiple regions and availability zones](#).

Use AMIs from trusted sources

Use only AMIs from a trusted source. Random AMIs pose a security risk and may perform slower than expected due to the way the EC2 install is configured. The following are examples of trusted AMIs:

- [Ubuntu Amazon EC2 AMI Locator](#)
- [Debian AmazonEC2Image](#)
- [CentOS-6 images on Amazon's EC2 Cloud](#)

EC2 clusters spanning multiple regions and availability zones

When creating an EC2 cluster that spans multiple regions and availability zones, use any of the [supported platforms](#) and [install Cassandra](#) on each node. It is best practice to use the same platform on all nodes. If your cluster was instantiated using the DataStax AMI, use Ubuntu for the additional nodes. Configure the cluster as a [multiple data center cluster](#) using the [Ec2MultiRegionSnitch](#) on page 23.

Production Cassandra clusters on EC2

For production Cassandra clusters on EC2, use these guidelines for choosing the instance types:

- Development and light production: **m3.large**
- Moderate production: **m3.xlarge**
- SSD production with light data: **c3.2xlarge**
- Largest heavy production: **m3.2xlarge** (PV) or **i2.2xlarge** (HVM)
- Micro, small, and medium types are not supported.

Note: The main difference between **m1** and **m3** instance types for use with Cassandra is that **m3** instance types have faster, smaller SSD drives and **m1** instance types have slower, larger rotational drives. Use **m1** instance types when you have higher tolerance for latency SLAs and you require smaller cluster sizes, or both. For more aggressive workloads use **m3** instance types with appropriately sized clusters.

EBS volumes recommended for production

SSD-backed general purpose volumes (GP2) or provisioned IOPS volumes (PIOPS) are suitable for production workloads. These volume types are designed to deliver consistent, low latency performance:

GP2	PIOPS
<ul style="list-style-type: none"> • The best choice for most workloads and have the advantage of guaranteeing 10,000 IOPS when volumes larger than 3.5TB are attached to instances. • Designed to deliver single-digit millisecond latencies. • Designed to deliver the provisioned performance 99.0% of the time. 	<ul style="list-style-type: none"> • Designed to deliver single-digit millisecond latencies. • Designed to deliver the provisioned performance 99.9% of the time.

EBS magnetic volumes not recommended

EBS magnetic volumes are not recommended for Cassandra data storage volumes for the following reasons:

Planning a cluster deployment

- EBS magnetic volumes contend directly for network throughput with standard packets. This contention means that EBS throughput is likely to fail if you saturate a network link.
- EBS magnetic volumes have unreliable performance. I/O performance can be exceptionally slow, causing the system to back load reads and writes until the entire cluster becomes unresponsive.
- Adding capacity by increasing the number of EBS volumes per host does not scale. You can easily surpass the ability of the system to keep effective buffer caches and concurrently serve requests for all of the data it is responsible for managing.

Note: Use only ephemeral **instance-store** or the recommended EBS volume types for Cassandra data storage.

For more information and graphs related to ephemeral versus EBS performance, see the blog article [Systematic Look at EC2 I/O](#).

Disk Performance Optimization

To ensure high disk performance to mounted drives, it is recommended that you pre-warm your drives by writing once to every drive location before production use. Depending on EC2 conditions, you can get moderate to enormous increases in throughput. See [Optimizing Disk Performance](#) in the *Amazon Elastic Compute Cloud Documentation*.

Storage recommendations for Cassandra 1.2 and later

Cassandra 1.2 and later supports JBOD (just a bunch of disks). JBOD excels at tolerating partial failures in a disk array. Configure using the [disk_failure_policy](#) in the `cassandra.yaml` file. Additional information is available in the [Handling Disk Failures In Cassandra 1.2](#) blog and [Recovering from a single disk failure using JBOD](#) on page 136.

Note: Cassandra JBOD support allows you to use standard disks. However, RAID0 may provide better throughput because it splits every block to be on another device. This means that writes are written in parallel fashion instead of written serially on disk.

The location of the [cassandra.yaml](#) file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra.yaml</code>
Tarball installations	<code>install_location/resources/cassandra/conf/cassandra.yaml</code>
Windows installations	

Storage recommendations for Cassandra 1.1 and earlier

RAID 0 the ephemeral disks. Then put both the data directory and the commit log on that volume. This has proved to be better in practice than putting the commit log on the root volume, which is also a shared resource. For more data redundancy, consider deploying your Cassandra cluster across multiple availability zones or using EBS volumes to store your Cassandra backup files.

Calculating partition size

For efficient operation, partitions must be sized within certain limits. Two measures of partition size are the number of values in a partition and the partition size on disk. The maximum number of columns per row is two billion. Sizing the disk space is more complex, and involves the number of rows and the number of columns, primary key columns and static columns in each table. Each application will have different efficiency parameters, but a good rule of thumb is to keep the maximum number of values below 100,000 items and the disk size under 100MB.

Calculating usable disk capacity

To calculate how much data your Cassandra nodes can hold, calculate the usable disk capacity per node and then multiply that by the number of nodes in your cluster. Remember that in a production cluster, you will typically have your commit log and data directories on different disks.

Procedure

1. Start with the raw capacity of the physical disks:

```
raw_capacity = disk_size * number_of_data_disks
```

2. Calculate the formatted disk space as follows:

```
formatted_disk_space = (raw_capacity * 0.9)
```

During normal operations, Cassandra routinely requires disk capacity for compaction and repair operations. For optimal performance and cluster health, DataStax recommends not filling your disks to capacity, but running at 50% to 80% capacity depending on the [compaction strategy](#) and size of the compactions.

3. Calculate the usable disk space accounting for file system formatting overhead (roughly 10 percent):

```
usable_disk_space = formatted_disk_space * (0.5 to 0.8)
```

Calculating user data size

The size of your raw data may be larger or smaller once it is loaded into Cassandra due to storage overhead. How much depends on how well it compresses and the characteristics of your data and tables. The following calculations account for data persisted to disk, not for data stored in memory.

Procedure

- Determine column overhead:

```
regular_total_column_size = column_name_size + column_value_size + 15
```

```
counter - expiring_total_column_size = column_name_size +  
column_value_size + 23
```

Every column in Cassandra incurs 15 bytes of overhead. Since each row in a table can have different column names as well as differing numbers of columns, metadata is stored for each column. For counter columns and expiring columns, you should add an additional 8 bytes (23 bytes total).

- Account for row overhead.

Every row in Cassandra incurs 23 bytes of overhead.

- Estimate primary key index size:

```
primary_key_index = number_of_rows * ( 32 + average_key_size )
```

Every table also maintains a partition index. This estimation is in bytes.

Planning a cluster deployment

- Determine replication overhead:

```
replication_overhead = total_data_size * ( replication_factor - 1 )
```

The replication factor plays a role in how much disk capacity is used. For a replication factor of 1, there is no overhead for replicas (as only one copy of data is stored in the cluster). If replication factor is greater than 1, then your total data storage requirement will include replication overhead.

Anti-patterns in Cassandra

Implementation or design patterns that are ineffective and/or counterproductive in Cassandra production installations. Correct patterns are suggested in most cases.

Storage area network

SAN storage is **not** recommended for on-premises deployments.

Note: Storage in clouds works very differently. Customers should contact DataStax for questions.

Although used frequently in Enterprise IT environments, SAN storage has proven to be a difficult and expensive architecture to use with distributed databases for a variety of reasons, including:

- SAN ROI (return on investment) does not scale along with that of Cassandra, in terms of capital expenses and engineering resources.
- In distributed architectures, SAN generally introduces a bottleneck and single point of failure because Cassandra's IO frequently surpasses the array controller's ability to keep up.
- External storage, even when used with a high-speed network and SSD, adds latency for all operations.
- Heap pressure is increased because pending I/O operations take longer.
- When the SAN transport shares operations with internal and external Cassandra traffic, it can saturate the network and lead to network availability problems.

Taken together these factors can create problems that are difficult to resolve in production. In particular, new users deploying Cassandra with SAN must first develop adequate methods and allocate sufficient engineering resources to identify these issues before they become a problem in production. For example, methods are needed for all key scaling factors, such as operational rates and SAN fiber saturation.

[Impact of Shared Storage on Cassandra](#) details metrics on how severely performance can be affected.

Network attached storage

Storing SSTables on a network attached storage (NAS) device is **not** recommended. Using a NAS device often results in network related bottlenecks resulting from high levels of I/O wait time on both reads and writes. The causes of these bottlenecks include:

- Router latency.
- The Network Interface Card (NIC) in the node.
- The NIC in the NAS device.

If you are required to use NAS for your environment, please contact a technical resource from DataStax for assistance.

Shared network file systems

Shared network file systems (NFS) has exhibited inconsistent behavior with its abilities to delete and move files. This configuration is not supported in Cassandra and it is not recommend to use.

Excessive heap space size

DataStax recommends using the default heap space size for most use cases. Exceeding this size can impair the Java virtual machine's (JVM) ability to perform fluid garbage collections (GC). The following table shows a comparison of heap space performances reported by a Cassandra user:

Heap	CPU utilization	Queries per second	Latency
40 GB	50%	750	1 second
8 GB	5%	8500 (not maxed out)	10 ms

For information on heap sizing, see [Tuning Java resources](#) on page 150.

Cassandra's rack feature

This information applies only to single-token architecture, not to [virtual nodes](#).

Defining one rack for the entire cluster is the simplest and most common implementation. Multiple racks should be avoided for the following reasons:

- Most users tend to ignore or forget rack requirements that racks should be organized in an alternating order. This order allows the data to get distributed safely and appropriately.
- Many users are not using the rack information effectively. For example, setting up with as many racks as nodes (or similar non-beneficial scenarios).
- Expanding a cluster when using racks can be tedious. The procedure typically involves several node moves and must ensure that racks are distributing data correctly and evenly. When clusters need immediate expansion, racks should be the last concern.

To use racks correctly:

- Use the same number of nodes in each rack.
- Use one rack and place the nodes in different racks in an alternating pattern. This allows you to still get the benefits of Cassandra's rack feature, and allows for quick and fully functional cluster expansions. Once the cluster is stable, you can swap nodes and make the appropriate moves to ensure that nodes are placed in the ring in an alternating fashion with respect to the racks.

Also see [About Replication in Cassandra](#) in the Cassandra 1.1 documentation.

SELECT ... IN or index lookups

SELECT ... IN and index lookups (formerly secondary indexes) should be avoided except for specific scenarios. See *When not to use IN* in [SELECT](#) and *When not to use an index* in [Indexing](#) in *CQL for Cassandra 2.2*.

Using the Byte Ordered Partitioner

The Byte Ordered Partitioner (BOP) is not recommended.

Use [virtual nodes](#) (vnodes) and use either the [Murmur3Partitioner](#) (default) or the [RandomPartitioner](#). Vnodes allow each node to own a large number of small ranges distributed throughout the cluster. Using vnodes saves you the effort of generating tokens and assigning tokens to your nodes. If not using vnodes, these partitioners are recommended because all writes occur on the hash of the key and are therefore spread out throughout the ring amongst tokens range. These partitioners ensure that your cluster evenly distributes data by placing the key at the correct token using the key's hash value.

Reading before writing

Reads take time for every request, as they typically have multiple disk hits for uncached reads. In work flows requiring reads before writes, this small amount of latency can affect overall throughput. All write I/

O in Cassandra is sequential so there is very little performance difference regardless of data size or key distribution.

Load balancers

Cassandra was designed to avoid the need for load balancers. Putting load balancers between Cassandra and Cassandra clients is harmful to performance, cost, availability, debugging, testing, and scaling. All high-level clients, such as the [Java](#) and [Python](#) drivers for Cassandra, implement load balancing directly.

Insufficient testing

Be sure to test at scale and production loads. This the best way to ensure your system will function properly when your application goes live. The information you gather from testing is the best indicator of what throughput per node is needed for future expansion calculations.

To properly test, set up a small cluster with production loads. There will be a maximum throughput associated with each node count before the cluster can no longer increase performance. Take the maximum throughput at this cluster size and apply it linearly to a cluster size of a different size. Next extrapolate (graph) your results to predict the correct cluster sizes for required throughputs for your production cluster. This allows you to predict the correct cluster sizes for required throughputs in the future. The [Netflix case study](#) shows an excellent example for testing.

Too many keyspaces or tables

Each Cassandra keyspace has a certain amount of overhead space that uses JVM memory. Each table uses approximately 1MB of memory. For example, 3,500 tables would use about 3.5GB of JVM memory. Using too many tables, or by extension, too many keyspaces will bloat the memory requirements. A good rule of thumb is to keep the number of tables within a cluster to 1,000 at most, and aim for 500 or less.

Lack of familiarity with Linux

Linux has a great collection of tools. Become familiar with the Linux built-in tools. It will help you greatly and ease operation and management costs in normal, routine functions. The essential list of tools and techniques to learn are:

- Parallel SSH and Cluster SSH: The pssh and cssh tools allow SSH access to multiple nodes. This is useful for inspections and cluster wide changes.
- Passwordless SSH: SSH authentication is carried out by using public and private keys. This allows SSH connections to easily hop from node to node without password access. In cases where more security is required, you can implement a bastion host and/or VPN.
- Useful common command-line tools include:
 - dstat: Shows all system resources instantly. For example, you can compare disk usage in combination with interrupts from your IDE controller, or compare the network bandwidth numbers directly with the disk throughput (in the same interval).
 - top: Provides an ongoing look at CPU processor activity in real time.
 - System performance tools: Tools such as iostat, mpstat, iftop, sar, lsof, netstat, htop, vmstat, and similar can collect and report a variety of metrics about the operation of the system.
 - vmstat: Reports information about processes, memory, paging, block I/O, traps, and CPU activity.
 - iftop: Shows a list of network connections. Connections are ordered by bandwidth usage, with the pair of hosts responsible for the most traffic at the top of list. This tool makes it easier to identify the hosts causing network congestion.

Running without the recommended settings

Be sure to use the [recommended settings](#) in the Cassandra documentation.

Also be sure to consult the [Planning a Cassandra cluster deployment](#) documentation, which discusses hardware and other recommendations before making your final hardware purchases.

Installing DataStax Distribution of Apache Cassandra 3.x

Installing the DataStax Distribution of Apache Cassandra 3.x on RHEL-based systems

Attention: OpsCenter is not supported or installed with Cassandra 2.2 and later.

Use these steps to install Cassandra using Yum repositories on RHEL, CentOS, and Oracle Linux.

Note: To install on SUSE, use the [Cassandra binary tarball distribution](#).

Prerequisites

- Ensure that your platform is [supported](#).
- Yum Package Management application installed.
- Root or sudo access to the install machine.
- Latest version of [Oracle Java Platform, Standard Edition 8 \(JDK\)](#) or [OpenJDK 8](#).

Note: The JDK is recommended for development and production systems. It provides tools that are not in the JRE, such as `jstack`, `jmap`, `jps`, and `jstat`, that are useful for troubleshooting.

- Python 2.7.

The packaged releases create a `cassandra` user. When starting Cassandra as a service, the service runs as this user. The following utilities are included in a separate package: `sstable2json`, `sstablelevelreset`, `sstablemetadata`, `json2sstable`, `sstable repairedset`, `sstablesplit`, and `token-generator`.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

```
$ java -version
```

It is recommended to use the latest version of Oracle Java 8 or OpenJDK 8 on all nodes.

2. Add the DataStax Distribution of Apache Cassandra 3.x repository to the `/etc/yum.repos.d/` `datastax.repo`:

```
[datastax-ddc]
name = DataStax Repo for Apache Cassandra
baseurl = http://rpm.datastax.com/datastax-ddc/3.version_number
enabled = 1
gpgcheck = 0
```

Note: Be sure to specify the version number. For example: 3.2.

3. Install the latest packages:

```
$ sudo yum install datastax-ddc
```


Installing DataStax Distribution of Apache Cassandra 3.x

This command automatically installs the Cassandra utilities such as `sstablelevelreset`, `sstablemetadata`, `sstableofflinerelevel`, `sstable repairedset`, `sstablesplit`, `token-generator`. Each utility provides usage/help information; type `help` after entering the command.

What to do next

- [Configure DataStax Community](#)
- [Initializing a multiple node cluster \(single data center\)](#) on page 114
- [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116
- [Recommended production settings](#)
- [Key components for configuring Cassandra](#)
- [Starting Cassandra as a service](#) on page 119
- [Package installation directories](#) on page 75

Installing DataStax Distribution of Apache Cassandra 3.x on Debian-based systems

Attention: OpsCenter is not supported or installed with Cassandra 2.2 and later.

Use these steps to install Cassandra using APT repositories on Debian and Ubuntu Linux.

Prerequisites

- Ensure that your platform is [supported](#).
- Advanced Package Tool is installed.
- Root or sudo access to the install machine.
- Latest version of [Oracle Java Platform, Standard Edition 8 \(JDK\)](#) or [OpenJDK 8](#).

Note: The JDK is recommended for development and production systems. It provides tools that are not in the JRE, such as `jstack`, `jmap`, `jps`, and `jstat`, that are useful for troubleshooting.

- Python 2.7.

The packaged releases create a `cassandra` user. When starting Cassandra as a service, the service runs as this user. The following utilities are included in a separate package: `sstable2json`, `sstablelevelreset`, `sstablemetadata`, `json2sstable`, `sstable repairedset`, `sstablesplit`, and `token-generator`.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

```
$ java -version
```

It is recommended to use the latest version of Oracle Java 8 or OpenJDK 8 on all nodes.

2. Add the DataStax Distribution of Apache Cassandra 3.x repository to the `/etc/apt/sources.list.d/cassandra.sources.list`

```
$ echo "deb http://debian.datastax.com/datastax-ddc 3.version_number main" |  
sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

Note: Be sure to specify the version number. For example: 3.2.

3. Optional: On Debian systems, to allow installation of the Oracle JVM instead of the OpenJDK JVM:

- a) In `/etc/apt/sources.list`, find the line that describes your source repository for Debian and add `contrib non-free` to the end of the line. For example:

```
deb http://some.debian.mirror/debian/ $distro main contrib non-free
```

- b) Save and close the file when you are done.

4. Add the DataStax repository key to your aptitude trusted keys.

```
$ curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
```

5. Install the latest package:

```
$ sudo apt-get update
$ sudo apt-get install datastax-ddc
```

This command automatically installs the Cassandra utilities such as `sstablelevelreset`, `sstablemetadata`, `sstableofflinerelevel`, `sstable repairedset`, `sstablesplit`, `token-generator`. Each utility provides usage/help information; type `help` after entering the command.

6. Because the Debian packages start the Cassandra service automatically, you must stop the server and clear the data:

Doing this removes the default `cluster_name` (Test Cluster) from the system table. All nodes must use the same cluster name.

```
$ sudo service cassandra stop
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

What to do next

- [Configure DataStax Community](#)
- [Initializing a multiple node cluster \(single data center\)](#) on page 114
- [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116
- [Recommended production settings](#)
- [Key components for configuring Cassandra](#)
- [Starting Cassandra as a service](#) on page 119
- [Package installation directories](#) on page 75

Related tasks

[Starting Cassandra as a service](#) on page 119

Related reference

[Package installation directories](#) on page 75

Installing DataStax Distribution of Apache Cassandra 3.x on any Linux-based platform

Attention: OpsCenter is not supported or installed with Cassandra 2.2 and later.

Use these steps to install Cassandra on all Linux-based platforms using a binary tarball.

Use this install for Mac OS X and platforms without package support, or if you do not have or want a root installation.

Prerequisites

- Ensure that your platform is [supported](#).
- Latest version of [Oracle Java Platform, Standard Edition 8 \(JDK\)](#) or [OpenJDK 8](#).

Note: The JDK is recommended for development and production systems. It provides tools that are not in the JRE, such as `jstack`, `jmap`, `jps`, and `jstat`, that are useful for troubleshooting.

- Python 2.7.

The binary tarball runs as a stand-alone process.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

```
$ java -version
```

It is recommended to use the latest version of Oracle Java 8 or OpenJDK 8 on all nodes.

2. Download the DataStax Distribution of Apache Cassandra 3.x:

```
$ curl -L http://downloads.datastax.com/datastax-ddc/datastax-ddc-version_number-bin.tar.gz | tar xz
```

You can also download from [Planet Cassandra](#). To untar:

```
$ tar -xvzf datastax-ddc-version_number-bin.tar.gz
```

Note: Automatically installs the Cassandra utilities such as `sstablelevelreset`, `sstablemetadata`, `sstableofflinerelevel`, `sstable repairedset`, `sstablesplit`, `token-generator`. Each utility provides usage/help information; type `help` after entering the command.

3. To configure, go to the `install/conf` directory:

```
$ cd datastax-ddc-version_number/conf
```

What to do next

- [Key components for configuring Cassandra](#)
- [Tarball installation directories](#) on page 74
- [Starting Cassandra as a stand-alone process](#) on page 119

Installing earlier releases of DataStax Distribution of Apache Cassandra 3.x

How to install the same version as other nodes in your cluster.

Follow the install instructions in the [relevant documentation](#) and specify the specific version in the install command.

Installing the packages for earlier releases on RHEL-based platforms

Cassandra 3.0.1 example:

```
$ sudo yum install dsc30-3.0.1.1 cassandra30-3.0.1.1
```

```
$ sudo yum install dsc30-3.0.1.1 cassandra30-3.0.1.1 cassandra30-tools-3.0.1.1  
## Optional
```

Cassandra 2.1.2 example:

```
$ sudo yum install dsc21-2.1.2.1 cassandra21-2.1.2-1  
$ sudo yum install dsc21-2.1.2.1 cassandra21-2.1.2-1 cassandra21-tools-2.1.2-1  
## Optional
```

Installing the packages for earlier releases on Debian-based platforms

Cassandra 3.0.1 example:

```
$ sudo apt-get install dsc30=3.0.1-1 cassandra=3.0.1  
$ sudo apt-get install dsc30=3.0.1-1 cassandra=3.0.1 cassandra-tools=3.0.1 ##  
Optional
```

Cassandra 2.1.2 example:

```
$ sudo apt-get install dsc21=2.1.2-1 cassandra=2.1.2  
$ sudo apt-get install dsc21=2.1.2-1 cassandra=2.1.2 cassandra-tools=2.1.2 ##  
Optional
```

Installing earlier releases using the binary tarball

1. Download the tarball using the URL for the prior version.

Cassandra 3.0.1 example:

```
http://downloads.datastax.com/community/dsc-cassandra-3.0.1-bin.tar.gz
```

Cassandra 2.1.2 example:

```
http://downloads.datastax.com/community/dsc-cassandra-2.1.2-bin.tar.gz
```

2. Unpack the distribution. For example:

```
$ tar -xzf dsc-cassandra-2.1.2-bin.tar.gz
```

The files are extracted into the dsc-cassandra-2.1.2 directory.

Installing earlier releases on Windows

Download the previous version you want from Planet Cassandra and follow the install instructions for that version.

Uninstalling the DataStax Distribution of Apache Cassandra 3.x

Select the uninstall method for your type of installation:

- [Debian- and RHEL-based packages](#)
- [Binary tarball](#)

Uninstalling Debian- and RHEL-based packages

Use this method when you have installed Apache Cassandra 3.0 using [APT](#) or [Yum](#).

1. Stop the Cassandra and DataStax Agent services:

Installing DataStax Distribution of Apache Cassandra 3.x

```
$ sudo service cassandra stop
```

2. Make sure all services are stopped:

```
$ ps aux | grep cassandra
$ ps aux | grep datastax-agent ## If the DataStax Agent was installed.
```

3. If services are still running, use the PID to kill the service:

```
$ sudo kill cassandra_pid
$ sudo kill datastax_agent_pid ## If the DataStax Agent was installed.
```

4. Remove the library and log directories:

```
$ sudo rm -r /var/lib/cassandra
$ sudo rm -r /var/log/cassandra
```

5. Remove the installation directories:

RHEL-based packages:

```
$ sudo yum remove "cassandra-*" "datastax-*" ## datastax-* required if the
DataStax Agent was installed.
```

Debian-based packages:

```
$ sudo apt-get purge "cassandra-*" "datastax-*" ## datastax-* required if
the DataStax Agent was installed.
```

Uninstalling the binary tarball

Use this method when you have installed Apache Cassandra 3.0 using the [binary tarball](#).

1. Stop the node:

```
$ ps aux | grep cassandra
$ sudo kill cassandra_pid
```

2. Stop the DataStax Agent if installed:

```
$ ps aux | grep datastax-agent
$ sudo kill datastax_agent_pid
```

3. Remove the installation directory.

Installing on cloud providers

You can install Cassandra on cloud providers that supply any of the [supported platforms](#).

You can install Cassandra 2.1 and earlier versions on Amazon EC2 using the DataStax AMI (Amazon Machine Image) as described in the [AMI documentation](#) for Cassandra 2.1.

To install Cassandra 2.2 and later on Amazon EC2, use a [trusted AMI](#) for your platform and the appropriate [install method](#) for that platform.

Installing the JDK

Installing Oracle JDK on RHEL-based Systems

Configure your operating system to use the latest version of [Oracle Java Platform, Standard Edition 8](#).

Procedure

1. Check which version of the JDK your system is using:

```
$ java -version
```

If Oracle Java is used, the results should look like:

```
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
```

2. If necessary, go to [Oracle Java SE Downloads](#), accept the license agreement, and download the installer for your distribution.

Note: If installing the Oracle JDK in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. From the directory where you downloaded the package, run the install:

```
$ sudo rpm -ivh jdk-8uversion-linux-x64.rpm
```

The RPM installs the JDK into the `/usr/java/` directory.

4. Set your system to use the Oracle JDK:

```
$ sudo alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_version/
bin/java 200000
```

5. Use the `alternatives` command to switch to the Oracle JDK.

```
$ sudo alternatives --config jav
```

Note: If you have trouble, you may need to set `JAVA_HOME` and `PATH` in your profile, such as `.bash_profile`.

The following examples assume that the JDK is in `/usr/java` and which `java` shows `/usr/bin/java`:

- Shell or bash:

```
$ export JAVA_HOME=/usr/java/latest
$ export PATH=$JAVA_HOME/bin:$PATH
```

- C shell (csh):

```
$ setenv JAVA_HOME "/usr/java/latest"
$ setenv PATH $JAVA_HOME/bin:$PATH
```

6. Make sure your system is using the correct JDK:

```
$ java -version
```

```
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
```

Installing Oracle JDK on Debian or Ubuntu Systems

Configure your operating system to use the latest version of [Oracle Java Platform, Standard Edition 8](#).

The Oracle Java Platform, Standard Edition (JDK) has been removed from the official software repositories of Ubuntu and only provides a binary (.bin) version. You can get the JDK from the [Java SE Downloads](#).

Procedure

1. Check which version of the JDK your system is using:

```
$ java -version
```

If Oracle Java is used, the results should look like:

```
java version "1.8.0_65"  
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)  
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
```

2. If necessary, go to [Oracle Java SE Downloads](#), accept the license agreement, and download the installer for your distribution.

Note: If installing the Oracle JDK in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. Make a directory for the JDK:

```
$ sudo mkdir -p /usr/lib/jvm
```

4. Unpack the tarball and install the JDK:

```
$ sudo tar zxvf jdk-8u65-linux-x64.tar.gz -C /usr/lib/jvm
```

The JDK files are installed into a directory called `/usr/lib/jvm/jdk-8u_version`.

5. Tell the system that there's a new Java version available:

```
$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/  
jdk1.8.0_version/bin/java" 1
```

If updating from a previous version that was removed manually, you may need to execute the above command twice, because you'll get an error message the first time.

6. Set the new JDK as the default using the following command:

```
$ sudo update-alternatives --config java
```

7. Make sure your system is using the correct JDK:

```
$ java -version
```

```
java version "1.8.0_65"  
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)  
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
```

Installing OpenJDK on RHEL-based Systems

Configure your operating system to use the [OpenJDK 8](#).

Procedure

In a terminal:

1. Install the OpenJDK 8:

```
$ su -c "yum install java-1.8.0-openjdk"
```

2. If you have more than one Java version installed on your system use the following command to switch versions:

```
$ sudo alternatives --config java
```

3. Make sure your system is using the correct JDK:

```
$ java -version
```

```
openjdk version "1.8.0_71"
OpenJDK Runtime Environment (build 1.8.0_71-b15)
OpenJDK 64-Bit Server VM (build 25.71-b15, mixed mode)
```

Installing OpenJDK on Debian-based Systems

Configure your operating system to use the [OpenJDK 8](#).

Procedure

In a terminal:

1. Install the OpenJDK 8 from a [PPA repository](#):

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
```

2. Update the system package cache and install:

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

3. If you have more than one Java version installed on your system use the following command to switch versions:

```
$ sudo update-alternatives --config java
```

4. Make sure your system is using the correct JDK:

```
$ java -version
```

```
openjdk version "1.8.0_72-internal"
OpenJDK Runtime Environment (build 1.8.0_72-internal-b05)
OpenJDK 64-Bit Server VM (build 25.72-b05, mixed mode)
```

Recommended production settings for Linux

Recommendations for production environments; adjust them accordingly for your implementation.

Java Virtual Machine

The latest 64-bit version of [Oracle Java Platform, Standard Edition 8 \(JDK\)](#) or [OpenJDK 7](#).

Synchronize clocks

The clocks on all nodes should be synchronized. You can use NTP (Network Time Protocol) or other methods.

This is required because columns are only overwritten if the timestamp in the new version of the column is more recent than the existing column.

Optimizing SSDs

For the majority of Linux distributions, SSDs are not configured optimally by default. The following steps ensures best practice settings for SSDs:

1. Ensure that the SysFS rotational flag is set to false (zero).

This overrides any detection by the operating system to ensure the drive is considered an SSD.

2. Repeat for any block devices created from SSD storage, such as mdarrays.

3. Set the IO scheduler to either deadline or noop:

- The noop scheduler is the right choice when the target block device is an array of SSDs behind a high-end IO controller that performs IO optimization.
- The deadline scheduler optimizes requests to minimize IO latency. If in doubt, use the deadline scheduler.

4. Set the read-ahead value for the block device to 8KB.

This setting tells the operating system not to read extra bytes, which can increase IO time and pollute the cache with bytes that weren't requested by the user.

For example, if the SSD is `/dev/sda`, in `/etc/rc.local`:

```
echo deadline > /sys/block/sda/queue/scheduler
#OR...
#echo noop > /sys/block/sda/queue/scheduler
echo 0 > /sys/class/block/sda/queue/rotational
echo 8 > /sys/class/block/sda/queue/read_ahead_kb
```

Disable zone_reclaim_mode on NUMA systems

The Linux kernel can be inconsistent in enabling/disabling `zone_reclaim_mode`. This can result in odd performance problems.

To ensure that `zone_reclaim_mode` is disabled:

```
$ echo 0 > /proc/sys/vm/zone_reclaim_mode
```

For more information, see [Peculiar Linux kernel performance problem on NUMA systems](#) on page 255.

User resource limits

You can view the current limits using the `ulimit -a` command. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Packaged installs: Ensure that the following settings are included in the `/etc/security/limits.d/cassandra.conf` file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

Tarball installs: Ensure that the following settings are included in the `/etc/security/limits.conf` file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

If you run Cassandra as root, some Linux distributions such as Ubuntu, require setting the limits for root explicitly instead of using `<cassandra_user>`:

```
root - memlock unlimited
root - nofile 100000
root - nproc 32768
root - as unlimited
```

For CentOS, RHEL, OEL systems, also set the nproc limits in `/etc/security/limits.d/90-nproc.conf`:

```
<cassandra_user> - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 131072
```

To make the changes take effect, reboot the server or run the following command:

```
$ sudo sysctl -p
```

To confirm the limits are applied to the Cassandra process, run the following command where *pid* is the process ID of the currently running Cassandra process:

```
$ cat /proc/<pid>/limits
```

For more information, see [Insufficient user resource limits errors](#).

Disable swap

You must disable swap entirely. Failure to do so can severely lower performance. Because Cassandra has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks.

If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

```
$ sudo swapoff --all
```

To make this change permanent, remove all swap file entries from `/etc/fstab`.

For more information, see [Nodes seem to freeze after some period of time](#).

Optimum blockdev --setra settings for RAID on spinning disks

Typically, a readahead of 128 is recommended.

Check to ensure setra is not set to 65536:

```
$ sudo blockdev --report /dev/<device>
```

To set setra:

```
$ sudo blockdev --setra 128 /dev/<device>
```

Install locations

Tarball installation directories

The configuration files are located in the following directories:

Configuration and sample files	Locations	Description
<code>cassandra.yaml</code>	<code>install_location/conf</code>	Main configuration file.
<code>cassandra-env.sh</code>	<code>install_location/conf</code>	Linux settings for Java, some JVM, and JMX.
<code>jvm.options</code>	<code>install_location/conf</code>	Static JVM settings for heap, garbage collection, and Cassandra startup parameters.
<code>cassandra.in.sh</code>	<code>install_location/bin</code>	Sets environment variables.
<code>cassandra-rackdc.properties</code>	<code>install_location/conf</code>	Defines the default data center and rack used by the GossipingPropertyFileSnitch, Ec2Snitch, Ec2MultiRegionSnitch, and GoogleCloudSnitch.
<code>cassandra-topology.properties</code>	<code>install_location/conf</code>	Defines the default data center and rack used by the PropertyFileSnitch.
<code>commit_archiving.properties</code>	<code>install_location/conf</code>	Configures commitlog archiving.
<code>cqlshrc.sample</code>	<code>install_location/conf</code>	Example file for using cqlsh with SSL encryption.
<code>metrics-reporter-config-sample.yaml</code>	<code>install_location/conf</code>	Example file for configuring metrics in Cassandra.
<code>logback.xmlcat</code>	<code>install_location/conf</code>	Configuration file for logback.
<code>triggers</code>	<code>install_location/conf</code>	The default location for the trigger JARs.

The binary tarball releases install into the installation directory.

Directories	Description
<code>bin</code>	Utilities and start scripts.
<code>conf</code>	Configuration files and environment settings.
<code>data</code>	Directory containing the files for commitlog, data, and saved_caches (unless set in <code>cassandra.yaml</code> .)

Directories	Description
interface	Thrift legacy API.
javadoc	Cassandra Java API documentation.
lib	JAR and license files.
tools	Cassandra tools and sample <code>cassandra.yaml</code> files for stress testing.

For DataStax Enterprise installs, see the documentation for your DataStax Enterprise version.

Package installation directories

The configuration files are located in the following directories:

Configuration Files	Locations	Description
<code>cassandra.yaml</code>	<code>/etc/cassandra</code>	Main configuration file.
<code>cassandra-env.sh</code>	<code>/etc/cassandra</code>	Linux settings for Java, some JVM, and JMX.
<code>jvm.options</code>	<code>/etc/cassandra</code>	Static JVM settings for heap, garbage collection, and Cassandra startup parameters.
<code>cassandra.in.sh</code>	<code>/usr/share/cassandra</code>	Sets environment variables.
<code>cassandra-rackdc.properties</code>	<code>/etc/cassandra</code>	Defines the default data center and rack used by the GossipingPropertyFileSnitch, Ec2Snitch, Ec2MultiRegionSnitch, and GoogleCloudSnitch.
<code>cassandra-topology.properties</code>	<code>/etc/cassandra</code>	Defines the default data center and rack used by the PropertyFileSnitch.
<code>commit_archiving.properties</code>	<code>/etc/cassandra</code>	Configures commitlog archiving.
<code>cqlshrc.sample</code>	<code>/etc/cassandra</code>	Example file for using <code>cqlsh</code> with SSL encryption.
<code>logback.xml</code>	<code>/etc/cassandra</code>	Configuration file for logback.
<code>triggers</code>	<code>/etc/cassandra</code>	The default location for the trigger JARs.

The packaged releases install into these directories:

Directories	Description
<code>/etc/default</code>	
<code>/etc/init.d/cassandra</code>	Service startup script.
<code>/etc/security/limits.d</code>	Cassandra user limits.
<code>/etc/cassandra</code>	Configuration files.
<code>/usr/bin</code>	Binary files.
<code>/usr/sbin</code>	

Directories	Description
<code>/usr/share/doc/cassandra/examples</code>	Sample <code>yaml</code> files for stress testing.
<code>/usr/share/cassandra</code>	JAR files and environment settings (<code>cassandra.in.sh</code>).
<code>/usr/share/cassandra/lib</code>	JAR files.
<code>/var/lib/cassandra</code>	Data, commitlog, and saved_caches directories.
<code>/var/log/cassandra</code>	Log directory.
<code>/var/run/cassandra</code>	Runtime files.

For DataStax Enterprise installs, see the documentation for your DataStax Enterprise version.

Configuration

The `cassandra.yaml` configuration file

The `cassandra.yaml` file is the main configuration file for Cassandra.

Important: After changing properties in the `cassandra.yaml` file, you must restart the node for the changes to take effect. It is located in the following directories:

- Cassandra package installations: `/etc/cassandra`
- Cassandra tarball installations: `install_location/conf`
- DataStax Enterprise package installations: `/etc/dse/cassandra`
- DataStax Enterprise tarball installations: `install_location/resources/cassandra/conf`

The configuration properties are grouped into the following sections:

- **Quick start**
The minimal properties needed for configuring a cluster.
- **Commonly used**
Properties most frequently used when configuring Cassandra.
- **Performance tuning**
Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.
- **Advanced**
Properties for advanced users or properties that are less commonly used.
- **Security**
Server and client security settings.

Note: Values with ^{note} indicate default values that are defined internally, missing, commented out, or implementation depends on other properties in the `cassandra.yaml` file. Additionally, some commented out values may not match the actual default value; these values are recommended when changing from the default.

Quick start properties

The minimal properties needed for configuring a cluster.

Related information: [Initializing a multiple node cluster \(single data center\)](#) on page 114 and [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116.

cluster_name

(Default: Test Cluster) The name of the cluster. This setting prevents nodes in one logical cluster from joining another. All nodes in a cluster must have the same value.

listen_address

(Default: localhost) The IP address or hostname that Cassandra binds to for connecting to other Cassandra nodes. Set this parameter or [listen_interface](#), not both. You must change the default setting for multiple nodes to communicate:

- Generally set to empty. If the node is properly configured (host name, name resolution, and so on), Cassandra uses `InetAddress.getLocalHost()` to get the local address from the system.
- For a single node cluster, you can use the default setting (localhost).
- If Cassandra can't find the correct address, you must specify the IP address or host name.
- Never specify 0.0.0.0; it is always wrong.

listen_interface

(Default: eth0)^{note} The interface that Cassandra binds to for connecting to other Cassandra nodes. Interfaces must correspond to a single address, IP aliasing is not supported. See [listen_address](#).

listen_interface_prefer_ipv6

(Default: false) By default, if an interface has an ipv4 and an ipv6 address, the first ipv4 address will be used. If set to true, the first ipv6 address will be used.

Default directories

If you have changed any of the default directories during installation, make sure you have root access and set these properties:

commitlog_directory

The directory where the commit log is stored. Default locations:

For optimal write performance, place the commit log be on a separate disk partition, or (ideally) a separate physical device from the data file directories. Because the commit log is append only, an HDD for is acceptable for this purpose.

data_file_directories

The directory location where table data (SSTables) is stored. Cassandra distributes data evenly across the location, subject to the granularity of the configured compaction strategy. Default locations:

- Package installations: `/var/lib/cassandra/data`
- Tarball installations: `install_location/data/data`
- Windows installations:
- Windows installations:

As a production best practice, use [RAID 0 and SSDs](#).

saved_caches_directory

The directory location where table key and row caches are stored. Default location:

- Package installations: `/var/lib/cassandra/saved_caches`

- Tarball installations: `install_location/data/saved_caches`
- Windows installations:

Commonly used properties

Properties most frequently used when configuring Cassandra.

Before starting a node for the first time, you should carefully evaluate your requirements.

Common initialization properties

Note: Be sure to set the properties in the [Quick start section](#) as well.

commit_failure_policy

(Default: stop) Policy for commit disk failures:

- die
Shut down gossip and Thrift and kill the JVM, so the node can be replaced.
- stop
Shut down gossip and Thrift, leaving the node effectively dead, but can be inspected using JMX.
- stop_commit
Shut down the commit log, letting writes collect but continuing to service reads (as in pre-2.0.5 Cassandra).
- ignore
Ignore fatal errors and let the batches fail.

disk_optimization_strategy

(Default: ssd) The strategy for optimizing disk reads can be set to either ssd or spinning.

disk_failure_policy

(Default: stop) Sets how Cassandra responds to disk failure. Recommend settings are stop or best_effort.

- die
Shut down gossip and Thrift and kill the JVM for any file system errors or single SSTable errors, so the node can be replaced.
- stop_paranoid
Shut down gossip and Thrift even for single SSTable errors.
- stop
Shut down gossip and Thrift, leaving the node effectively dead, but available for inspection using JMX.
- best_effort
Stop using the failed disk and respond to requests based on the remaining available SSTables. This means you will see obsolete data at consistency level of ONE.
- ignore
Ignores fatal errors and lets the requests fail; all file system errors are logged but otherwise ignored. Cassandra acts as in versions prior to 1.2.

Related information: [Handling Disk Failures In Cassandra 1.2](#) blog and [Recovering from a single disk failure using JBOD](#) on page 136.

endpoint_snitch

(Default: `org.apache.cassandra.locator.SimpleSnitch`) Set to a class that implements the `IEndpointSnitch`. Cassandra uses snitches for locating nodes and routing requests.

- SimpleSnitch

Use for single-data center deployments or single-zone in public clouds. Does not recognize data center or rack information. It treats strategy order as proximity, which can improve cache locality when disabling read repair.

- **GossipingPropertyFileSnitch**

Recommended for production. The rack and data center for the local node are defined in the `cassandra-rackdc.properties` file and propagated to other nodes via gossip. To allow migration from the `PropertyFileSnitch`, it uses the `cassandra-topology.properties` file if it is present.

The location of the `cassandra-rackdc.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-rackdc.properties</code>
Tarball installations	<code>install_location/conf/cassandra-rackdc.properties</code>
Windows installations	

The location of the `cassandra-topology.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-topology.properties</code>
Tarball installations	<code>install_location/conf/cassandra-topology.properties</code>
Windows installations	

- **PropertyFileSnitch**

Determines proximity by rack and data center, which are explicitly configured in the `cassandra-topology.properties` file.

- **Ec2Snitch**

For EC2 deployments in a single region. Loads region and availability zone information from the EC2 API. The region is treated as the data center and the availability zone as the rack. Uses only private IPs. Subsequently it does not work across multiple regions.

- **Ec2MultiRegionSnitch**

Uses public IPs as the [broadcast_address](#) to allow cross-region connectivity. This means you must also set [seed](#) addresses to the public IP and open the [storage_port](#) or [ssl_storage_port](#) on the public IP firewall. For intra-region traffic, Cassandra switches to the private IP after establishing a connection.

- **RackInferringSnitch:**

Proximity is determined by rack and data center, which are assumed to correspond to the 3rd and 2nd octet of each node's IP address, respectively. This snitch is best used as an example for writing a custom snitch class (unless this happens to match your deployment conventions).

Related information: [Snitches](#) on page 20

rpc_address

(Default: localhost) The listen address for client connections (Thrift RPC service and native transport). Valid values are:

- **unset:**

Resolves the address using the hostname configuration of the node. If left unset, the hostname must resolve to the IP address of this node using `/etc/hostname`, `/etc/hosts`, or DNS.

- **0.0.0.0:**

Configuration

Listens on all configured interfaces, but you must set the [broadcast_rpc_address](#) to a value other than 0.0.0.0.

- IP address
- hostname

Related information: [Network](#)

rpc_interface

(Default: eth1) ^{note} The listen address for client connections. Interfaces must correspond to a single address, IP aliasing is not supported. See [rpc_address](#).

rpc_interface_prefer_ipv6

(Default: false) By default, if an interface has an ipv4 and an ipv6 address, the first ipv4 address will be used. If set to true, the first ipv6 address will be used.

seed_provider

The addresses of hosts deemed contact points. Cassandra nodes use the -seeds list to find each other and learn the topology of the ring.

- `class_name` (Default: `org.apache.cassandra.locator.SimpleSeedProvider`)

The class within Cassandra that handles the seed logic. It can be customized, but this is typically not required.

- `-seeds` (Default: 127.0.0.1)

A comma-delimited list of IP addresses used by [gossip](#) for bootstrapping new nodes joining a cluster. When running multiple nodes, you must change the list from the default value. In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is **not** recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).

Related information: [Initializing a multiple node cluster \(single data center\)](#) on page 114 and [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116.

enable_user_defined_functions

(Default: false) In Cassandra 3.0 and later, user defined functions (UDFs) are disabled by default. UDFs are sandboxed to prevent execution of evil code. If you wish to use UDFs, change to true.

enable_scripted_user_defined_functions

(Default: false) Java UDFs are always enabled, if `enable_user_defined_functions` is true. Enable this option to use UDFs with language `javascript` or any custom JSR-223 provider. This option has no effect, if `enable_user_defined_functions` is false.

Common compaction settings

compaction_throughput_mb_per_sec

(Default: 16) Throttles compaction to the specified total throughput across the node. The faster you insert data, the faster you need to compact in order to keep the SSTable count down. The recommended value is 16 to 32 times the rate of write throughput (in MB/second). Setting the value to 0 disables compaction throttling.

Related information: [Configuring compaction](#) on page 155

compaction_large_partition_warning_threshold_mb

(Default: 100) Logs a warning when compaction partitions larger than the set value.

Common memtable settings

memtable_total_space_in_mb

(Default: 1/4 of heap)^{note} Specifies the total memory used for all memtables on a node. This replaces the per-table storage settings `memtable_operations_in_millions` and `memtable_throughput_in_mb`.

Related information: [Tuning the Java heap](#)

Common disk settings

concurrent_reads

(Default: 32)^{note} For workloads with more data than can fit in memory, the bottleneck is reads fetching data from disk. Setting to $(16 \times \text{number_of_drives})$ allows operations to queue low enough in the stack so that the OS and drives can reorder them. The default setting applies to both logical volume managed (LVM) and RAID drives.

concurrent_writes

(Default: 32)^{note} Writes in Cassandra are rarely I/O bound, so the ideal number of concurrent writes depends on the number of CPU cores in your system. The recommended value is $8 \times \text{number_of_cpu_cores}$.

concurrent_counter_writes

(Default: 32)^{note} Counter writes read the current values before incrementing and writing them back. The recommended value is $(16 \times \text{number_of_drives})$.

concurrent_batchlog_writes

(Default: 32) Limit the concurrent batchlog writes, similar to `concurrent_writes`.

concurrent_materialized_view_writes

(Default: 32) Limit the concurrent materialize view writes to the lesser of concurrent reads or concurrent writes, because there is a read involved in materialized view writes.

Common automatic backup settings

incremental_backups

(Default: false) Backs up data updated since the last snapshot was taken. When enabled, Cassandra creates a hard link to each SSTable flushed or streamed locally in a `backups/` subdirectory of the keyspace data. Removing these links is the operator's responsibility.

Related information: [Enabling incremental backups](#) on page 133

snapshot_before_compaction

(Default: false) Enable or disable taking a snapshot before each compaction. This option is useful to back up data when there is a data format change. Be careful using this option because Cassandra does not clean up older snapshots automatically.

Related information: [Configuring compaction](#) on page 155

Common fault detection setting

phi_convict_threshold

(Default: 8)^{note} Adjusts the sensitivity of the failure detector on an exponential scale. Generally this setting never needs adjusting.

Related information: [Failure detection and recovery](#) on page 14

Performance tuning properties

Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

Commit log settings

commitlog_sync

(Default: periodic) The method that Cassandra uses to acknowledge writes in milliseconds:

- periodic: (Default: 10000 milliseconds [10 seconds])

Configuration

Used with `commitlog_sync_period_in_ms` to control how often the commit log is synchronized to disk. Periodic syncs are acknowledged immediately.

- `batch`: (Default: disabled)^{note}

Used with `commitlog_sync_batch_window_in_ms` (Default: 2 ms) to control how long Cassandra waits for other writes before performing a sync. When using this method, writes are not acknowledged until fsynced to disk.

commitlog_segment_size_in_mb

(Default: 32MB) Sets the size of the individual commitlog file segments. A commitlog segment may be archived, deleted, or recycled after all its data has been flushed to SSTables. This amount of data can potentially include commitlog segments from every table in the system. The default size is usually suitable for most commitlog archiving, but if you want a finer granularity, 8 or 16 MB is reasonable.

This property determines the maximum mutation size, defined as half the segment size. If a mutation's size exceeds the maximum mutation size, the mutation is rejected. Before increasing the commitlog segment size of the commitlog segments, investigate why the mutations are larger than expected. Look for underlying issues with access patterns and data model, because increasing the commitlog segment size is a limited fix.

Related information: [Commit log archive configuration](#) on page 110

commitlog_compression

(Default: not enabled) Sets the compressor to use if commit log is compressed. Options are: LZ4, Snappy or Deflate. The commit log is written uncompressed if a compressor option is not set.

commitlog_total_space_in_mb

(Default: 32MB for 32-bit JVMs, 8192MB for 64-bit JVMs)^{note} Total space used for commit logs. If the used space goes above this value, Cassandra rounds up to the next nearest segment multiple and flushes memtables to disk for the oldest commitlog segments, removing those log segments. This reduces the amount of data to replay on start-up, and prevents infrequently-updated tables from indefinitely keeping commitlog segments. A small total commitlog space tends to cause more flush activity on less-active tables.

Related information: [Configuring memtable throughput](#) on page 154

Compaction settings

Related information: [Configuring compaction](#) on page 155

concurrent_compactors

(Default: Smaller of number of disks or number of cores, with a minimum of 2 and a maximum of 8 per CPU core)^{note} Sets the number of concurrent compaction processes allowed to run simultaneously on a node, not including validation compactions for anti-entropy repair. Simultaneous compactions help preserve read performance in a mixed read-write workload by mitigating the tendency of small SSTables to accumulate during a single long-running compaction. If your data directories are backed by SSD, increase this value to the number of cores. If compaction running too slowly or too fast, adjust [compaction_throughput_mb_per_sec](#) first.

sstable_preemptive_open_interval_in_mb

(Default: 50MB) When compacting, the replacement opens SSTables before they are completely written and uses in place of the prior SSTables for any range previously written. This setting helps to smoothly transfer reads between the SSTables by reducing page cache churn and keeps hot rows hot.

Memtable settings

memtable_allocation_type

(Default: `heap_buffers`) Specify the way Cassandra allocates and manages memtable memory. See [Off-heap memtables in Cassandra 2.1](#). Options are:

- `heap_buffers`
On heap NIO (non-blocking I/O) buffers.
- `offheap_buffers`

Off heap (direct) NIO buffers.

- `offheap_objects`

Native memory, eliminating NIO buffer heap overhead.

memtable_cleanup_threshold

(Default: 0.11 $1/(\text{memtable_flush_writers} + 1)$)^{note} Ratio of occupied non-flushing memtable size to total permitted size for triggering a flush of the largest memtable. Larger values mean larger flushes and less compaction, but also less concurrent flush activity, which can make it difficult to keep your disks saturated under heavy write load.

file_cache_size_in_mb

(Default: Smaller of 1/4 heap or 512) Total memory to use for SSTable-reading buffers.

buffer_pool_use_heap_if_exhausted

(Default: true, but commented out) Indicates whether to allocate on or off heap when the sstable buffer pool is exhausted (when the buffer pool has exceeded the maximum memory `file_cache_size_in_mb`), beyond which it will not cache buffers but allocate on request.

memtable_flush_writers

(Default: Smaller of number of disks or number of cores with a minimum of 2 and a maximum of 8)^{note} Sets the number of memtable flush writer threads. These threads are blocked by disk I/O, and each one holds a memtable in memory while blocked. If your data directories are backed by SSD, increase this setting to the number of cores.

memtable_heap_space_in_mb

(Default: 1/4 heap)^{note} Total permitted memory to use for memtables. Triggers a flush based on `memtable_cleanup_threshold`. Cassandra stops accepting writes when the limit is exceeded until a flush completes. If unset, sets to default.

Related information: [Flushing data from the memtable](#)

memtable_offheap_space_in_mb

(Default: 1/4 heap)^{note} See `memtable_heap_space_in_mb`.

Related information: [Flushing data from the memtable](#)

Cache and index settings

column_index_size_in_kb

(Default: 64) Granularity of the index of rows within a partition. For huge rows, decrease this setting to improve seek time. If you use key cache, be careful not to make this setting too large because key cache will be overwhelmed. If you're unsure of the size of the rows, it's best to use the default setting.

index_summary_capacity_in_mb

(Default: 5% of the heap size [empty])^{note} Fixed memory pool size in MB for SSTable index summaries. If the memory usage of all index summaries exceeds this limit, any SSTables with low read rates shrink their index summaries to meet this limit. This is a best-effort process. In extreme conditions, Cassandra may need to use more than this amount of memory.

index_summary_resize_interval_in_minutes

(Default: 60 minutes) How frequently index summaries should be re-sampled. This is done periodically to redistribute memory from the fixed-size pool to SSTables proportional their recent read rates. To disable, set to -1. This leaves existing index summaries at their current sampling level.

Disks settings

stream_throughput_outbound_megabits_per_sec

(Default: 200 seconds)^{note} Throttles all outbound streaming file transfers on a node to the specified throughput. Cassandra does mostly sequential I/O when streaming data during bootstrap or repair, which can lead to saturating the network connection and degrading client (RPC) performance.

inter_dc_stream_throughput_outbound_megabits_per_sec

(Default: unset)^{note} Throttles all streaming file transfer between the data centers. This setting allows throttles streaming throughput between data centers in addition to throttling all network stream traffic as configured with [stream_throughput_outbound_megabits_per_sec](#).

trickle_fsync

(Default: false) When doing sequential writing, enabling this option tells fsync to force the operating system to flush the dirty buffers at a set interval `trickle_fsync_interval_in_kb`. Enable this parameter to avoid sudden dirty buffer flushing from impacting read latencies. Recommended to use on SSDs, but not on HDDs.

trickle_fsync_interval_in_kb

(Default: 10240). Sets the size of the fsync in kilobytes.

windows_timer_interval

(Default: 1) The default Windows kernel timer and scheduling resolution is 15.6ms for power conservation. Lowering this value on Windows can provide much tighter latency and better throughput. However, some virtualized environments may see a negative performance impact from changing this setting below the system default. The sysinternals 'clockres' tool can confirm your system's default setting.

Advanced properties

Properties for advanced users or properties that are less commonly used.

Advanced initialization properties

auto_bootstrap

(Default: true) This setting has been removed from default configuration. It makes new (non-seed) nodes automatically migrate the right data to themselves. When initializing a fresh cluster *without* data, add `auto_bootstrap: false`.

Related information: [Initializing a multiple node cluster \(single data center\)](#) on page 114 and [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116.

batch_size_warn_threshold_in_kb

(Default: 5KB per batch) Log WARN on any batch size exceeding this value in kilobytes. Caution should be taken on increasing the size of this threshold as it can lead to node instability.

batch_size_fail_threshold_in_kb

(Default: 50KB per batch) Fail any batch exceeding this setting. The default value is 10X the value of `batch_size_warn_threshold_in_kb`.

broadcast_address

(Default: `listen_address`)^{note} The IP address a node tells other nodes in the cluster to contact it by. It allows public and private address to be different. For example, use the `broadcast_address` parameter in topologies where not all nodes have access to other nodes by their private IP addresses.

If your Cassandra cluster is deployed across multiple Amazon EC2 regions and you use the `Ec2MultiRegionSnitch`, set the `broadcast_address` to public IP address of the node and the `listen_address` to the private IP.

initial_token

(Default: disabled) Used in the single-node-per-token architecture, where a node owns exactly one contiguous range in the ring space. Setting this property overrides [num_tokens](#).

If you not using vnodes or have [num_tokens](#) set it to 1 or unspecified (`#num_tokens`), you should always specify this parameter when setting up a production cluster for the first time and when adding capacity. For more information, see this parameter in the [Cassandra 1.1 Node and Cluster Configuration](#) documentation.

This parameter can be used with `num_tokens` (vnodes) in special cases such as [Restoring from a snapshot](#) on page 133.

num_tokens

(Default: 256) ^{note} Defines the number of tokens randomly assigned to this node on the ring when using [virtual nodes](#) (vnodes). The more tokens, relative to other nodes, the larger the proportion of data that the node stores. Generally all nodes should have the same number of tokens assuming equal hardware capability. The recommended value is 256. If unspecified (`#num_tokens`), Cassandra uses 1 (equivalent to `#num_tokens : 1`) for legacy compatibility and uses the `initial_token` setting.

If not using vnodes, comment `#num_tokens : 256` or set `num_tokens : 1` and use [initial_token](#). If you already have an existing cluster with one token per node and wish to migrate to vnodes, see [Enabling virtual nodes on an existing production cluster](#).

Note: If using DataStax Enterprise, the default setting of this property depends on the type of node and type of install.

allocate_tokens_keyspace

(Default: KEYSPACE) Automatic allocation of `num_tokens` tokens for this node is triggered. The allocation algorithm attempts to choose tokens in a way that optimizes replicated load over the nodes in the datacenter for the replication strategy used by the specified KEYSPACE. The load assigned to each node will be close to proportional to its number of vnodes.

partitioner

(Default: `org.apache.cassandra.dht.Murmur3Partitioner`) Distributes rows (by partition key) across all nodes in the cluster. Any `IPartitioner` may be used, including your own as long as it is in the class path. For new clusters use the default partitioner.

Cassandra provides the following partitioners for backwards compatibility:

- `RandomPartitioner`
- `ByteOrderedPartitioner` (deprecated)
- `OrderPreservingPartitioner` (deprecated)

Related information: [Partitioners](#) on page 18

storage_port

(Default: 7000) The port for inter-node communication.

tracetype_query_ttl

(Default: 86400) TTL for different trace types used during logging of the repair process

tracetype_repair_ttl

(Default: 604800) TTL for different trace types used during logging of the repair process.

Advanced automatic backup setting

auto_snapshot

(Default: true) Enable or disable whether a snapshot is taken of the data before keyspace truncation or dropping of tables. To prevent data loss, using the default setting is strongly advised. If you set to false, you will lose data on truncation or drop.

Key caches and global row properties

When creating or modifying tables, you enable or disable the key cache (partition key cache) or row cache for that table by setting the caching parameter. Other row and key cache tuning and configuration options are set at the global (node) level. Cassandra uses these settings to automatically distribute memory for each table on the node based on the overall workload and specific table usage. You can also configure the save periods for these caches globally.

Related information: [Configuring caches](#)

key_cache_keys_to_save

(Default: disabled - all keys are saved) ^{note} Number of keys from the key cache to save.

key_cache_save_period

(Default: 14400 seconds [4 hours]) Duration in seconds that keys are saved in cache. Caches are saved to [saved_caches_directory](#). Saved caches greatly improve cold-start speeds and has relatively little effect on I/O.

key_cache_size_in_mb

(Default: empty) A global cache setting for tables. It is the maximum size of the key cache in memory. When no value is set, the cache is set to the smaller of 5% of the available heap, or 100MB. To disable set to 0.

Related information: [setcachecapacity](#).

row_cache_class_name

(Default: disabled - row cache is not enabled)^{note} Specify which row cache provider to use, OHCPProvider or SerializingCacheProvider. OHCPProvider is fully off-heap, SerializingCacheProvider is partially off-heap.

row_cache_keys_to_save

(Default: disabled - all keys are saved)^{note} Number of keys from the row cache to save.

row_cache_size_in_mb

(Default: 0- disabled) Maximum size of the row cache in memory. Row cache can save more time than [key_cache_size_in_mb](#), but is space-intensive because it contains the entire row. Use the row cache only for hot rows or static rows. If you reduce the size, you may not get you hottest keys loaded on start up.

row_cache_save_period

(Default: 0- disabled) Duration in seconds that rows are saved in cache. Caches are saved to [saved_caches_directory](#). This setting has limited use as described in [row_cache_size_in_mb](#).

memory_allocator

(Default: NativeAllocator) The off-heap memory allocator. In addition to caches, this property affects storage engine meta data. Supported values:

- NativeAllocator
- JEMallocAllocator

Experiments show that jemalloc saves some memory compared to the native allocator because it is more fragmentation resistant. To use, install jemalloc as a library and modify `cassandra-env.sh`.

CAUTION: JEMalloc version 3.6.0 or later should be used with option. Known errors occur with earlier versions.

Counter caches properties

Counter cache helps to reduce counter locks' contention for hot counter cells. In case of RF = 1 a counter cache hit will cause Cassandra to skip the read before write entirely. With RF > 1 a counter cache hit will still help to reduce the duration of the lock hold, helping with hot counter cell updates, but will not allow skipping the read entirely. Only the local (clock, count) tuple of a counter cell is kept in memory, not the whole counter, so it's relatively cheap.

Note: Reducing the size counter cache may result in not getting the hottest keys loaded on start-up.

counter_cache_size_in_mb

(Default value: empty)^{note} When no value is specified a minimum of 2.5% of Heap or 50MB. If you perform counter deletes and rely on low [gc_grace_seconds](#), you should disable the counter cache. To disable, set to 0.

counter_cache_save_period

(Default: 7200 seconds [2 hours]) Duration after which Cassandra should save the counter cache (keys only). Caches are saved to [saved_caches_directory](#).

counter_cache_keys_to_save

(Default value: disabled)^{note} Number of keys from the counter cache to save. When disabled all keys are saved.

Tombstone settings

When executing a scan, within or across a partition, tombstones must be kept in memory to allow returning them to the coordinator. The coordinator uses them to ensure other replicas know about the deleted rows. Workloads that generate numerous tombstones may cause performance problems and exhaust the server heap. See [Cassandra anti-patterns: Queues and queue-like datasets](#). Adjust these thresholds only if you understand the impact and want to scan more tombstones. Additionally, you can adjust these thresholds at runtime using the `StorageServiceMBean`.

Related information: [Cassandra anti-patterns: Queues and queue-like datasets](#)

tombstone_warn_threshold

(Default: 1000) The maximum number of tombstones a query can scan before warning.

tombstone_failure_threshold

(Default: 100000) The maximum number of tombstones a query can scan before aborting.

Network timeout settings

range_request_timeout_in_ms

(Default: 10000 milliseconds) The time that the coordinator waits for sequential or index scans to complete.

read_request_timeout_in_ms

(Default: 5000 milliseconds) The time that the coordinator waits for read operations to complete.

counter_write_request_timeout_in_ms

(Default: 5000 milliseconds) The time that the coordinator waits for counter writes to complete.

cas_contention_timeout_in_ms

(Default: 1000 milliseconds) The time that the coordinator continues to retry a CAS (compare and set) operation that contends with other proposals for the same row.

truncate_request_timeout_in_ms

(Default: 60000 milliseconds) The time that the coordinator waits for truncates (remove all data from a table) to complete. The long default value allows for a snapshot to be taken before removing the data. If [auto_snapshot](#) is disabled (not recommended), you can reduce this time.

write_request_timeout_in_ms

(Default: 2000 milliseconds) The time that the coordinator waits for write operations to complete.

Related information: [Hinted Handoff: repair during write path](#) on page 137

request_timeout_in_ms

(Default: 10000 milliseconds) The default time for other, miscellaneous operations.

Related information: [Hinted Handoff: repair during write path](#) on page 137

Inter-node settings

cross_node_timeout

(Default: false) Enable or disable operation timeout information exchange between nodes (to accurately measure request timeouts). If disabled Cassandra assumes the request are forwarded to the replica instantly by the coordinator, which means that under overload conditions extra time is required for processing already-timed-out requests..

CAUTION: Before enabling this property make sure NTP (network time protocol) is installed and the times are synchronized between the nodes.

internode_send_buff_size_in_bytes

(Default: N/A)^{note} Sets the sending socket buffer size in bytes for inter-node calls.

When setting this parameter and [internode_recv_buff_size_in_bytes](#), the buffer size is limited by `net.core.wmem_max`. When unset, buffer size is defined by `net.ipv4.tcp_wmem`. See `man tcp` and:

- `/proc/sys/net/core/wmem_max`
- `/proc/sys/net/core/rmem_max`

Configuration

- `/proc/sys/net/ipv4/tcp_wmem`
- `/proc/sys/net/ipv4/tcp_wmem`

internode_recv_buff_size_in_bytes

(Default: N/A)^{note} Sets the receiving socket buffer size in bytes for inter-node calls.

internode_compression

(Default: all) Controls whether traffic between nodes is compressed. The valid values are:

- `all`
All traffic is compressed.
- `dc`
Traffic between data centers is compressed.
- `none`
No compression.

inter_dc_tcp_nodelay

(Default: false) Enable or disable `tcp_nodelay` for inter-data center communication. When disabled larger, but fewer, network packets are sent. This reduces overhead from the TCP protocol itself. However, if cross data-center responses are blocked, it will increase latency.

streaming_socket_timeout_in_ms

(Default: 3600000 - 1 hour)^{note} Enable or disable socket timeout for streaming operations. When a timeout occurs during streaming, streaming is retried from the start of the current file. Avoid setting this value too low, as it can result in a significant amount of data re-streaming.

Native transport (CQL Binary Protocol)

start_native_transport

(Default: true) Enable or disable the native transport server. Uses the same address as the `rpc_address`, but the port is different from the `rpc_port`. See `native_transport_port`.

native_transport_port

(Default: 9042) Port on which the CQL native transport listens for clients.

native_transport_max_threads

(Default: 128)^{note} The maximum number of thread handling requests. Similar to `rpc_max_threads` and differs as follows:

- Default is different (128 versus unlimited).
- No corresponding `native_transport_min_threads`.
- Idle threads are stopped after 30 seconds.

native_transport_max_frame_size_in_mb

(Default: 256MB) The maximum size of allowed frame. Frame (requests) larger than this are rejected as invalid.

native_transport_max_concurrent_connections

(Default: -1) Specifies the maximum number of concurrent client connections. The default value of -1 means unlimited.

native_transport_max_concurrent_connections_per_ip

(Default: -1) Specifies the maximum number of concurrent client connections per source IP address. The default value of -1 means unlimited.

RPC (remote procedure call) settings

Settings for configuring and tuning client connections.

broadcast_rpc_address

(Default: unset)^{note} RPC address to broadcast to drivers and other Cassandra nodes. This cannot be set to 0.0.0.0. If blank, it is set to the value of the [rpc_address](#) or [rpc_interface](#). If [rpc_address](#) or [rpc_interface](#) is set to 0.0.0.0, this property must be set.

rpc_port

(Default: 9160) Thrift port for client connections.

start_rpc

(Default: true) Starts the Thrift RPC server.

rpc_keepalive

(Default: true) Enable or disable keepalive on client connections (RPC or native).

rpc_max_threads

(Default: unlimited)^{note} Regardless of your choice of RPC server ([rpc_server_type](#)), the number of maximum requests in the RPC thread pool dictates how many concurrent requests are possible. However, if you are using the parameter `sync` in the `rpc_server_type`, it also dictates the number of clients that can be connected. For a large number of client connections, this could cause excessive memory usage for the thread stack. Connection pooling on the client side is highly recommended. Setting a maximum thread pool size acts as a safeguard against misbehaved clients. If the maximum is reached, Cassandra blocks additional connections until a client disconnects.

rpc_min_threads

(Default: 16)^{note} Sets the minimum thread pool size for remote procedure calls.

rpc_recv_buff_size_in_bytes

(Default: N/A)^{note} Sets the receiving socket buffer size for remote procedure calls.

rpc_send_buff_size_in_bytes

(Default: N/A)^{note} Sets the sending socket buffer size in bytes for remote procedure calls.

rpc_server_type

(Default: sync) Cassandra provides three options for the RPC server. On Windows, `sync` is about 30% slower than `hsha`. On Linux, `sync` and `hsha` performance is about the same, but `hsha` uses less memory.

- `sync`: (Default One thread per Thrift connection.)

For a very large number of clients, memory is the limiting factor. On a 64-bit JVM, 180KB is the minimum stack size per thread and corresponds to your use of virtual memory. Physical memory may be limited depending on use of stack space.

- `hsha`:

Half synchronous, half asynchronous. All Thrift clients are handled asynchronously using a small number of threads that does not vary with the number of clients and thus scales well to many clients. The RPC requests are synchronous (one thread per active request).

Note: When selecting this option, you must change the default value (unlimited) of [rpc_max_threads](#).

- Your own RPC server

You must provide a fully-qualified class name of an `o.a.c.t.TServerFactory` that can create a server instance.

Advanced fault detection settings

Settings to handle poorly performing or failing nodes.

dynamic_snitch_badness_threshold

(Default: 0.1) Sets the performance threshold for dynamically routing client requests away from a poorly performing node. Specifically, it controls how much worse a poorly performing node has to be before the [dynamic snitch](#) prefers other replicas over it. A value of 0.2 means Cassandra continues to prefer the static snitch values until the node response time is 20% worse than the best performing node. Until the threshold is reached, incoming requests are statically routed to the closest replica (as determined by the snitch). If

the value of this parameter is greater than zero and [read_repair_chance](#) is less than 1.0, cache capacity is maximized across the nodes.

dynamic_snitch_reset_interval_in_ms

(Default: 600000 milliseconds) Time interval to reset all node scores, which allows a bad node to recover.

dynamic_snitch_update_interval_in_ms

(Default: 100 milliseconds) The time interval for how often the snitch calculates node scores. Because score calculation is CPU intensive, be careful when reducing this interval.

hints_flush_period_in_ms

(Default: 10000) Set how often hints are flushed from internal buffers to disk.

hints_directory

(Default: \$CASSANDRA_HOME/data/hints) Set directory where hints are stored.

hinted_handoff_enabled

(Default: true) Enable or disable hinted handoff. To enable per data center, add data center list. For example: `hinted_handoff_enabled: DC1,DC2`. A hint indicates that the write needs to be replayed to an unavailable node. Where Cassandra writes the hint to a hints file on the coordinator node.

Related information: [Hinted Handoff: repair during write path](#) on page 137

hinted_handoff_disabled_datacenters

(Default: none) Specify blacklist of data centers that will not perform hinted handoffs. To enable per data center, add data center list. For example: `hinted_handoff_disabled_datacenters: - DC1 - DC2`.

Related information: [Hinted Handoff: repair during write path](#) on page 137

hinted_handoff_throttle_in_kb

(Default: 1024) Maximum throttle per delivery thread in kilobytes per second. This rate reduces proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread will use the maximum rate. If there are three, each node will throttle to half of the maximum, since the two nodes are expected to deliver hints simultaneously.

max_hint_window_in_ms

(Default: 10800000 milliseconds [3 hours]) Maximum amount of time that hints are generated for an unresponsive node. After this interval, new hints are no longer generated until the node is back up and responsive. If the node goes down again, a new interval begins. This setting can prevent a sudden demand for resources when a node is brought back online and the rest of the cluster attempts to replay a large volume of hinted writes.

Related information: [Failure detection and recovery](#) on page 14

max_hints_delivery_threads

(Default: 2) Number of threads with which to deliver hints. In multiple data-center deployments, consider increasing this number because cross data-center handoff is generally slower.

max_hints_file_size_in_mb

(Default: 128) The maximum size for a single hints file, in megabytes.

batchlog_replay_throttle_in_kb

(Default: 1024KB per second) Total maximum throttle. Throttling is reduced proportionally to the number of nodes in the cluster.

Request scheduler properties

Settings to handle incoming client requests according to a defined policy. If you need to use these properties, your nodes are overloaded and dropping requests. It is recommended that you add more nodes and not try to prioritize requests.

request_scheduler

(Default: `org.apache.cassandra.scheduler.NoScheduler`) Defines a scheduler to handle incoming client requests according to a defined policy. This scheduler is useful for throttling client requests in single

clusters containing multiple keyspaces. This parameter is specifically for requests from the client and does not affect inter-node communication. Valid values are:

- `org.apache.cassandra.scheduler.NoScheduler`
No scheduling takes place.
- `org.apache.cassandra.scheduler.RoundRobinScheduler`
Round robin of client requests to a node with a separate queue for each `request_scheduler_id` property.
- A Java class that implements the `RequestScheduler` interface.

request_scheduler_id

(Default: `keyspace`)^{note} An identifier on which to perform request scheduling. Currently the only valid value is `keyspace`. See [weights](#).

request_scheduler_options

(Default: disabled) Contains a list of properties that define configuration options for `request_scheduler`:

- `throttle_limit`
The number of in-flight requests per client. Requests beyond this limit are queued up until running requests complete. Recommended value is $((\text{concurrent_reads} + \text{concurrent_writes}) \times 2)$.
- `default_weight`: (Default: 1)^{note}
How many requests are handled during each turn of the `RoundRobin`.
- `weights`: (Default: `Keyspace: 1`)
Takes a list of keyspaces. It sets how many requests are handled during each turn of the `RoundRobin`, based on the `request_scheduler_id`.

Thrift interface properties

Legacy API for older clients. [CQL](#) is a simpler and better API for Cassandra.

thrift_framed_transport_size_in_mb

(Default: 15) Frame size (maximum field length) for Thrift. The frame is the row or part of the row that the application is inserting.

thrift_max_message_length_in_mb

(Default: 16) The maximum length of a Thrift message in megabytes, including all fields and internal Thrift overhead (1 byte of overhead for each frame). Message length is usually used in conjunction with batches. A frame length greater than or equal to 24 accommodates a batch with four inserts, each of which is 24 bytes. The required message length is greater than or equal to $24+24+24+24+4$ (number of frames).

Security properties

Server and client security settings.

authenticator

(Default: `AllowAllAuthenticator`) The authentication backend. It implements `IAuthenticator` for identifying users. The available authenticators are:

- `AllowAllAuthenticator`:
Disables authentication; no checks are performed.
- `PasswordAuthenticator`
Authenticates users with user names and hashed passwords stored in the `system_auth.credentials` table. If you use the default, 1, and the node with the lone replica goes down, you will not be able to log into the cluster because the `system_auth` keyspace was not replicated.

Related information: [Internal authentication](#) on page 100

internode_authenticator

(Default: enabled)^{note} Internode authentication backend. It implements `org.apache.cassandra.auth.AllowAllInternodeAuthenticator` to allow or disallow connections from peer nodes.

authorizer

(Default: `AllowAllAuthorizer`) The authorization backend. It implements `IAuthorizer` to limit access and provide permissions. The available authorizers are:

- `AllowAllAuthorizer`
Disables authorization; allows any action to any user.
- `CassandraAuthorizer`
Stores permissions in `system_auth.permissions` table. If you use the default, 1, and the node with the lone replica goes down, you will not be able to log into the cluster because the `system_auth` keyspace was not replicated.

Related information: [Object permissions](#) on page 101

role_manager

(Default: `CassandraRoleManager`) Part of the Authentication & Authorization backend that implements `IRoleManager` to maintain grants and memberships between roles. Out of the box, Cassandra provides `org.apache.cassandra.auth.CassandraRoleManager`, which stores role information in the `system_auth` keyspace. Most functions of the `IRoleManager` require an authenticated login, so unless the configured `IAuthorizer` actually implements authentication, most of this functionality will be unavailable. `CassandraRoleManager` stores role data in the `system_auth` keyspace. Please increase `system_auth` keyspace replication factor if you use the role manager.

roles_validity_in_ms

(Default: 2000) Fetching permissions can be an expensive operation depending on the authorizer, so this setting allows flexibility. Validity period for roles cache; set to 0 to disable. Granted roles are cached for authenticated sessions in `AuthenticatedUser` and after the period specified here, become eligible for (async) reload. Will be disabled automatically for `AllowAllAuthenticator`.

roles_update_interval_in_ms

(Default: 2000) Refresh interval for roles cache, if enabled. Defaults to the same value as `roles_validity_in_ms`. After this interval, cache entries become eligible for refresh. Upon next access, an async reload is scheduled and the old value returned until it completes. If `roles_validity_in_ms` is non-zero, then this must be also.

permissions_validity_in_ms

(Default: 2000) How long permissions in cache remain valid. Depending on the authorizer, such as `CassandraAuthorizer`, fetching permissions can be resource intensive. This setting disabled when set to 0 or when `AllowAllAuthorizer` is set.

Related information: [Object permissions](#) on page 101

permissions_update_interval_in_ms

(Default: same value as [permissions_validity_in_ms](#))^{note} Refresh interval for permissions cache (if enabled). After this interval, cache entries become eligible for refresh. On next access, an async reload is scheduled and the old value is returned until it completes. If `permissions_validity_in_ms`, then this property must be non-zero.

server_encryption_options

Enable or disable inter-node encryption. You must also generate keys and provide the appropriate key and trust store locations and passwords. No custom encryption options are currently enabled. The available options are:

- `internode_encryption`: (Default: none) Enable or disable encryption of inter-node communication using the `TLS_RSA_WITH_AES_128_CBC_SHA` cipher suite for authentication, key exchange, and encryption of data transfers. Use the DHE/ECDHE ciphers if running in (Federal Information Processing Standard) FIPS 140 compliant mode. The available inter-node options are:

- `all`
Encrypt all inter-node communications.
- `none`
No encryption.
- `dc`
Encrypt the traffic between the data centers (server only).
- `rack`
Encrypt the traffic between the racks (server only).
- `keystore`: (Default: `conf/.keystore`)
The location of a Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
- `keystore_password`: (Default: `cassandra`)
Password for the keystore.
- `truststore`: (Default: `conf/.truststore`)
Location of the truststore containing the trusted certificate for authenticating remote servers.
- `truststore_password`: (Default: `cassandra`)
Password for the truststore.

The passwords used in these options must match the passwords used when generating the keystore and truststore. For instructions on generating these files, see [Creating a Keystore to Use with JSSE](#).

The advanced settings are:

- `protocol`: (Default: `TLS`)
- `algorithm`: (Default: `SunX509`)
- `store_type`: (Default: `JKS`)
- `cipher_suites`:
TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA (Default: `TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA`)
- `require_client_auth`: (Default: `false`)
Enables or disables certificate authentication.

Related information: [Node-to-node encryption](#) on page 99

client_encryption_options

Enable or disable client-to-node encryption. You must also generate keys and provide the appropriate key and trust store locations and passwords. No custom encryption options are currently enabled. The available options are:

- `enabled`: (Default: `false`)
To enable, set to `true`.
- `keystore`: (Default: `conf/.keystore`)
The location of a Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
- `keystore_password`: (Default: `cassandra`)
Password for the keystore. This must match the password used when generating the keystore and truststore.
- `require_client_auth`: (Default: `false`)
Enables or disables certificate authentication. (Available starting with Cassandra 1.2.3.)

Configuration

- `truststore:` (Default: `conf/.truststore`)
Set if `require_client_auth` is true.
- `truststore_password:` `<truststore_password>`
Set if `require_client_auth` is true.

The advanced settings are:

- `protocol:` (Default: TLS)
- `algorithm:` (Default: SunX509)
- `store_type:` (Default: JKS)
- `cipher_suites:` (Default: TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA)

Related information: [Client-to-node encryption](#) on page 96

ssl_storage_port

(Default: 7001) The SSL port for encrypted communication. Unused unless enabled in `encryption_options`.

native_transport_port_ssl

(Default: 9142) In Cassandra 3.0 and later, an additional dedicated port can be designated for encryption. If client encryption is enabled and `native_transport_port_ssl` is disabled, the `native_transport_port` (default: 9042) will encrypt all traffic. To use both unencrypted and encrypted traffic, enable `native_transport_port_ssl`

Cassandra include file

To set environment variables, Cassandra can use the `cassandra.in.sh` file, located in:

- Tarball installations: `install_location/bin/cassandra.in.sh`
- Package installations: `/usr/share/cassandra/cassandra.in.sh`

Security

Securing Cassandra

Cassandra provides these security features to the open source community.

- [SSL encryption](#)

Cassandra includes secure communication from a client machine to a database cluster. Client to server SSL ensures data in flight is not compromised and is securely transferred. Client-to-node and node-to-node encryption can be configured.

- [Authentication based on internally controlled login accounts/passwords](#)

Administrators can create users and roles who can be authenticated to Cassandra database clusters using the `CREATE USER` or `CREATE ROLE` command. Internally, Cassandra manages user accounts and access to the database cluster using passwords. User accounts may be altered and dropped using CQL.

- [Object permission management](#)

Once authenticated into a database cluster using either internal authentication, the next security issue to be tackled is permission management. What can the user do inside the database? Authorization capabilities for Cassandra use the familiar `GRANT/REVOKE` security paradigm to manage object permissions.

SSL encryption

The Secure Socket Layer (SSL) is a cryptographic protocol used to secure communications between computers. For reference, see [SSL in wikipedia](#).

Briefly, it works in the following manner. A client and server are defined as two entities that are communicating with one another, either software or hardware. These entities must exchange information to set up trust between them. Each entity that will provide such information must have a generated key that consists of a private key that only the entity stores and a public key that can be exchanged with other entities. If the client wants to connect to the server, the client requests the secure connection and the server sends a certificate that includes its public key. The client checks the validity of the certificate by exchanging information with the server, which the server validates with its private key. If a two-way validation is desired, this process must be carried out in both directions. Private keys are stored in the `keystore` and public keys are stored in the `truststore`.

For Cassandra, the entities can be nodes or one of the tools such as `cqlsh` or `nodetool` running on either a local node or a remote node.

Preparing server certificates

To use SSL encryption for client-to-node encryption or node-to-node encryption, SSL certificates must be generated using [keytool](#). If you generate the certificates for one type of encryption, you do not need to generate them again for the other; the same certificates are used for both. All nodes must have all the relevant SSL certificates on all nodes. A keystore contains private keys. The truststore contains SSL certificates for each node. The certificates in the truststore don't require signing by a trusted and recognized public certification authority.

Procedure

- Generate a private and public key pair on each node of the cluster. Use an alias that identifies the node. Prompts for the keystore password, `dname` (first and last name, organizational unit, organization, city, state, country), and key password. The `dname` should be generated with the CN value as the IP address or FQDN for the node.

```
$ keytool -genkey -keyalg RSA -alias node0 -keystore keystore.node0
```

- The generation command can also include all prompted-for information in the command line. This example uses an alias of `node0`, a keystore name of `keystore.node0`, uses the same password of `cassandra` for both the keystore and the key, and a `dname` that identifies the IP address of `node0` as `172.31.10.22`.

```
$ keytool -genkey -keyalg RSA -alias node0 -keystore keystore.node0 -
storepass cassandra -keypass cassandra -dname "CN=172.31.10.22, OU=None,
O=None, L=None, C=None"
```

- Export the public part of the certificate to a separate file.

```
$ keytool -export -alias cassandra -file node0.cer -keystore .keystore
```

- Add the `node0.cer` certificate to the `node0` truststore of the node using the `keytool -import` command.

```
$ keytool -import -v -trustcacerts -alias node0 -file node0.cer -keystore
truststore.node0
```

- `cqlsh` does not work with the certificate in the format generated. `openssl` is used to generate a PEM file of the certificate with no keys, `node0.cer.pem`, and a PEM file of the key with no certificate, `node0.key.pem`. First, the keystore is imported in PKCS12 format to a destination keystore, `node0.p12`, in the example. This is followed by the two commands that convert create the two PEM files.

Configuration

```
$ keytool -importkeystore -srckeystore keystore.node0 -destkeystore node0.p12 -deststoretype PKCS12 -srcstorepass cassandra -deststorepass cassandra
openssl pkcs12 -in node0.p12 -nokeys -out node0.cer.pem -passin pass:cassandra
openssl pkcs12 -in node0.p12 -nodes -nocerts -out node0.key.pem -passin pass:cassandra
```

- For client-to-remote-node encryption or node-to-node encryption, use a copying tool such as `scp` to copy the `node0.cer` file to each node. Import the file into the truststore after copying to each node. The example imports the certificate for node0 into the truststore for node1.

```
$ keytool -import -v -trustcacerts -alias node0 -file node0.cer -keystore truststore.node1
```

- Make sure keystore file is readable only to the Cassandra daemon and not by any user of the system.
- Check that the certificates exist in the keystore and truststore files using `keytool -list`. The example shows checking for the node1 certificate in the keystore file and for the node0 and node1 certificates in the truststore file.

```
$ keytool -list -keystore keystore.node1
keytool -list -keystore truststore.node1
```

Adding new trusted users

How to add new users when client certificate authentication is enabled.

Prerequisites

The client certificate authentication must be enabled (`require_client_auth=true`).

Procedure

1. Generate the certificate as described in [Client-to-node encryption](#) on page 96.
2. Import the user's certificate into every node's truststore using `keytool`:

```
$ keytool -import -v -trustcacerts -alias <username> -file <certificate file> -keystore .truststore
```

Client-to-node encryption

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

Prerequisites

All nodes must have all the relevant SSL certificates on all nodes. See [Preparing server certificates](#) on page 95.

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

Procedure

On each node under `client_encryption_options`:

- Enable encryption.

- Set the appropriate paths to your `.keystore` and `.truststore` files.
- Provide the required passwords. The passwords must match the passwords used when generating the keystore and truststore.
- To enable client certificate authentication for two-way SSL encryption, set `require_client_auth` to `true`. Enabling this option allows tools like `cqlsh` to connect to a remote node. If only local access is required, such as running `cqlsh` on a local node with SSL encryption, this option is not required. If the option is set to `true`, then the truststore and truststore password must also be included. The password used for both the keystore and the truststore in this example is `cassandra`.

This example uses the password `cassandra`

```
client_encryption_options:
  enabled: true
  # The path to your keystore file; ex: conf/keystore.node0
  keystore: conf/keystore.node0
  # The password for your keystore file
  keystore_password: cassandra
  # The next 3 lines are included if 2-way SSL is desired
  require_client_auth: true
  # The path to your truststore file; ex: conf/truststore.node0
  truststore: conf/truststore.node0
  # The password for your truststore file
  truststore_password: cassandra
```

Note: Cassandra must be restarted after making changes to the `cassandra.yaml` file.

- Enabling client encryption will encrypt all traffic on the `native_transport_port` (default: 9042). If both encrypted and unencrypted traffic is required, an additional `cassandra.yaml` setting must be enabled. The `native_transport_port_ssl` (default: 9142) sets an additional dedicated port to carry encrypted transmissions, while `native_transport_port` carries unencrypted transmissions. It is beneficial to install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files if this option is enabled.

Using cqlsh with SSL encryption

Using a `cqlshrc` file is the easiest method of getting `cqlshrc` settings. The `cqlshrc.sample` provides an example that can be copied as a starting point.

Procedure

1. To run `cqlsh` with SSL encryption, create a `.cassandra/cqlshrc` file in your home or client program directory. The following settings must be added to the file. When `validate` is enabled, the host in the certificate is compared to the host of the machine that it is connected to verify that the certificate is trusted.

```
[authentication]
username = fred
password = !!bang!!$

[connection]
hostname = 127.0.0.1
port = 9042
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = ~/keys/node0.cer.pem
# Optional, true by default
validate = true
```

```
# The next 2 lines must be provided when require_client_auth = true in the
cassandra.yaml file
userkey = ~/node0.key.pem
usercert = ~/node0.cer.pem

[certfiles]
# Optional section, overrides the default certfile in the [ssl] section
for 2 way SSL
172.31.10.22 = ~/keys/node0.cer.pem
172.31.8.141 = ~/keys/node1.cer.pem
```

Note: The use of the same IP addresses in the `[certfiles]` as is used to generate the `dname` of the certificates is required for 2 way SSL encryption. Each node must have a line in the `[certfiles]` section for client-to-remote-node or node-to-node. The SSL certificate must be provided either in the configuration file or as an environment variable. The environment variables (`SSL_CERTFILE` and `SSL_VALIDATE`) override any options set in this file.

2. Start `cqlsh` with the `--ssl` option for `cqlsh` to local node encrypted connection.

```
$ cqlsh --ssl ## Package installations
$ install_location/bin/cqlsh --ssl ## Tarball installations
```

3. Start `cqlsh` with the `--ssl` option for `cqlsh` and an IP address for remote node encrypted connection.

```
$ cqlsh --ssl ## Package installations
$ install_location/bin/cqlsh --ssl 172.31.10.22 ## Tarball installations
```

Using nodetool (JMX) with SSL

Using `nodetool` with SSL requires some JMX setup. Changes to `cassandra-env.sh` are required, and a configuration file, `~/cassandra/nodetool-ssl.properties`, is created.

Procedure

1. First, follow steps #1-8 in [Enabling JMX authentication](#) on page 103.
2. To run `nodetool` with SSL encryption, some additional changes are required to `cassandra-env.sh`. The following settings must be added to the file. Use the file path to the keystore and truststore, and appropriate passwords for each file.

```
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=/home/automaton/
keystore.node0"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=cassandra"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=/home/automaton/
truststore.node0"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=cassandra"
JVM_OPTS="$JVM_OPTS -
Dcom.sun.management.jmxremote.ssl.need.client.auth=true"
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.registry.ssl=true"
```

3. [Restart Cassandra](#).
4. To run `nodetool` with SSL encryption, create a `.cassandra/nodetool-ssl.properties` file in your home or client program directory with the following settings.

```
-Djavax.net.ssl.keyStore=/home/automaton/keystore.node0
-Djavax.net.ssl.keyStorePassword=cassandra
-Djavax.net.ssl.trustStore=/home/automaton/truststore.node0
-Djavax.net.ssl.trustStorePassword=cassandra
-Dcom.sun.management.jmxremote.ssl.need.client.auth=true
-Dcom.sun.management.jmxremote.registry.ssl=true
```

5. Start `nodetool` with the `--ssl` option for encrypted connection for any `nodetool` operation.

```
$ nodetool --ssl info ## Package installations
$ install_location/bin/nodetool -ssl info ## Tarball installations
```

Node-to-node encryption

Node-to-node encryption protects data transferred between nodes in a cluster, including gossip communications, using SSL (Secure Sockets Layer).

Prerequisites

All nodes must have all the relevant SSL certificates on all nodes. See [Preparing server certificates](#) on page 95.

To enable node-to-node SSL, you must set the [server_encryption_options](#) in the `cassandra.yaml` file.

Procedure

On each node under `server_encryption_options`:

- Enable `internode_encryption`.
The available options are:
 - `all`
 - `none`
 - `dc`: Cassandra encrypts the traffic between the data centers.
 - `rack`: Cassandra encrypts the traffic between the racks.
- Set the appropriate paths to your `keystore` and `truststore` files.
- Provide the required passwords. The passwords must match the passwords used when generating the `keystore` and `truststore`.
- To enable 2 way certificate authentication, set `require_client_auth` to `true`.

Example

```
server_encryption_options:
  internode_encryption: all
  keystore: /conf/keystore.node0
  keystore_password: cassandra
  truststore: /conf/truststore.node0
  truststore_password: cassandra
  require_client_auth: true
```

What to do next

Cassandra must be restarted after making changes to the `cassandra.yaml` file. Use the `nodetool` utility to check if all nodes are up after making the changes.

```
$ cqlsh --ssl ## Package installations
$ install_location/bin/nodetool ring ## Tarball installations
```

Internal authentication

Internal authentication

Like [object permission management](#) using internal authorization, internal authentication is based on Cassandra-controlled login accounts and passwords. Internal authentication works for the following clients when you provide a user name and password to start up the client:

- Astyanax
- cqlsh
- [DataStax drivers](#) - produced and certified by DataStax to work with Cassandra.
- Hector
- pycassa

Internal authentication stores usernames and bcrypt-hashed passwords in the `system_auth.credentials` table.

PasswordAuthenticator is an IAuthenticator implementation that you can use to configure Cassandra for internal authentication out-of-the-box.

Configuring authentication

To configure Cassandra to use internal authentication, first make a change to the `cassandra.yaml` file and increase the replication factor of the `system_auth` keyspace, as described in this procedure. Next, start up Cassandra using the default user name and password (`cassandra/cassandra`), and start `cqlsh` using the same credentials. Finally, use these CQL statements to set up user accounts to authorize users to access the database objects:

- [ALTER ROLE](#)
- [ALTER USER](#)
- [CREATE ROLE](#)
- [CREATE USER](#)
- [DROP ROLE](#)
- [DROP USER](#)
- [LIST ROLES](#)
- [LIST USERS](#)

Note: To configure authorization, see [Internal authorization](#) on page 101.

Procedure

1. Change the authenticator option in the `cassandra.yaml` file to PasswordAuthenticator.

By default, the authenticator option is set to AllowAllAuthenticator.

```
authenticator: PasswordAuthenticator
```

2. [Increase the replication factor](#) for the `system_auth` keyspace to N (number of nodes).

If you use the default, 1, and the node with the lone replica goes down, you will not be able to log into the cluster because the `system_auth` keyspace was not replicated.

3. Restart the Cassandra client.

The default superuser name and password that you use to start the client is stored in Cassandra.

```
$ client_startup_string -u cassandra -p cassandra
```

4. Start `cqlsh` using the superuser name and password.

```
$ cqlsh -u cassandra -p cassandra
```

5. Create another superuser, not named `cassandra`. This step is optional but highly recommended.
6. Log in as that new superuser.
7. Change the `cassandra` user password to something long and incomprehensible, and then forget about it. It won't be used again.
8. Take away the `cassandra` user's superuser status.
9. Use the CQL statements listed previously to set up user accounts and then grant permissions to access the database objects.

Logging in using `cqlsh`

Typically, after configuring authentication, you log into `cqlsh` using the `-u` and `-p` options to the `cqlsh` command. To avoid having to enter credentials every time you launch `cqlsh`, you can create a `.cassandra/cqlshrc` file. When present, this file passes default login information to `cqlsh`. The `cqlshrc.sample` provides an example.

Procedure

1. Open a text editor and create a file that specifies a user name and password.

```
[authentication]
username = fred
password = !!bang!!$
```

2. Save the file in your `home/.cassandra` directory and name it `cqlshrc`.
3. Set permissions on the file.

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

Internal authorization

Object permissions

Cassandra provides the familiar relational database GRANT/REVOKE paradigm to grant or revoke permissions to access Cassandra data. A superuser grants initial permissions, and subsequently a user or role may or may not be given the permission to grant/revoke permissions. Object permission management is based on internal authorization.

Read access to these system tables is implicitly given to every authenticated user or role because the tables are used by most Cassandra tools:

- `system_schema.keyspaces`
- `system_schema.columns`
- `system_schema.tables`
- `system.local`
- `system.peers`

Configuring internal authorization

`CassandraAuthorizer` is one of many possible `IAuthorizer` implementations, and the one that stores permissions in the `system_auth.permissions` table to support all authorization-related CQL statements. Configuration consists mainly of changing the `authorizer` option in the `cassandra.yaml` to use the `CassandraAuthorizer`.

Note: To configure authentication, see [Configuring internal authorization](#) on page 101.

Procedure

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

```
authorizer: CassandraAuthorizer
```

You can use any authenticator except `AllowAll`.

2. [Configure the replication factor](#) for the `system_auth` keyspace to increase the replication factor to a number greater than 1.
3. Adjust the validity period for permissions caching by setting the [permissions_validity_in_ms](#) option in the `cassandra.yaml` file.
Alternatively, disable permission caching by setting this option to 0.

Results

CQL supports these authorization statements:

- [GRANT](#)
- [LIST PERMISSIONS](#)
- [REVOKE](#)

Configuring firewall port access

If you have a firewall running on the nodes in your Cassandra cluster, you must open up the following ports to allow communication between the nodes, including certain Cassandra ports. If this isn't done, when you start Cassandra on a node, the node acts as a standalone database server rather than joining the database cluster.

Table: Public port

Port number	Description
22	SSH port

Table: Cassandra inter-node ports

Port number	Description
7000	Cassandra inter-node cluster communication.
7001	Cassandra SSL inter-node cluster communication.
7199	Cassandra JMX monitoring port.

Table: Cassandra client ports

Port number	Description
9042	Cassandra client port.
9160	Cassandra client port (Thrift).

Related concepts

[Planning an Amazon EC2 cluster](#) on page 56

Enabling JMX authentication

The default settings for Cassandra make JMX accessible only from localhost. If you want to enable remote JMX connections, change the `LOCAL_JMX` setting in `cassandra-env.sh` and enable authentication and/or SSL. After enabling JMX authentication, ensure that tools that use JMX, such as [nodetool](#), are configured to use authentication.

Procedure

1. Open the `cassandra-env.sh` file for editing and update or add these lines:

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=/etc/
cassandra/jmxremote.password"
```

If the `LOCAL_JMX` setting is in your file, set:

```
LOCAL_JMX=no
```

2. Copy the `jmxremote.password.template` from `/jdk_install_location/lib/management/` to `/etc/cassandra/` and rename it to `jmxremote.password`:

```
$ cp jdk_install_location/lib/management/jmxremote.password.template /etc/
cassandra/jmxremote.password
```

Note: This is a sample path. Set the location of this file in `jdk_install_location/lib/management/management.properties`.

3. For the user running Cassandra, change the ownership of `jmxremote.password` and change permissions to read only:

```
$ chown cassandra:cassandra /etc/cassandra/jmxremote.password
$ chmod 400 /etc/cassandra/jmxremote.password
```

4. Edit `jmxremote.password` and add the user and password for JMX-compliant utilities:

```
monitorRole QED
controlRole R&D
cassandra cassandrapassword ## Specify the credentials for your
environment.
```

5. Add the `cassandra` user with read permission to `jdk_install_location/lib/management/jmxremote.access`:

```
monitorRole readonly
cassandra readwrite
controlRole readwrite \
create javax.management.monitor.,javax.management.timer. \
unregister
```

6. [Restart Cassandra](#).
7. Run `nodetool status` with the `cassandra` user and password.

```
$ nodetool status -u cassandra -pw cassandra
```

Example

If you run `nodetool status` without user and password, you see an error similar to:

```
Exception in thread "main" java.lang.SecurityException: Authentication failed!
Credentials required
at
  com.sun.jmx.remote.security.JMXPluggableAuthenticator.authenticationFailure(Unknown
  Source)
at com.sun.jmx.remote.security.JMXPluggableAuthenticator.authenticate(Unknown
  Source)
at sun.management.jmxremote.ConnectorBootstrap
  $AccessFileCheckerAuthenticator.authenticate(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl.doNewClient(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl.newClient(Unknown Source)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at sun.rmi.server.UnicastServerRef.dispatch(Unknown Source)
at sun.rmi.transport.Transport$1.run(Unknown Source)
at sun.rmi.transport.Transport$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at sun.rmi.transport.Transport.serviceCall(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport.handleMessages(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
at java.lang.Thread.run(Unknown Source)
at sun.rmi.transport.StreamRemoteCall.exceptionReceivedFromServer(Unknown
  Source)
at sun.rmi.transport.StreamRemoteCall.executeCall(Unknown Source)
at sun.rmi.server.UnicastRef.invoke(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl_Stub.newClient(Unknown Source)
at javax.management.remote.rmi.RMIConnector.getConnection(Unknown Source)
at javax.management.remote.rmi.RMIConnector.connect(Unknown Source)
at javax.management.remote.JMXConnectorFactory.connect(Unknown Source)
at org.apache.cassandra.tools.NodeProbe.connect(NodeProbe.java:146)
at org.apache.cassandra.tools.NodeProbe.<init>(NodeProbe.java:116)
at org.apache.cassandra.tools.NodeCmd.main(NodeCmd.java:1099)
```

Configuring gossip settings

When a node first starts up, it looks at its `cassandra.yaml` configuration file to determine the name of the Cassandra cluster it belongs to; which nodes (called *seeds*) to contact to obtain information about the other nodes in the cluster; and other parameters for determining port and range information.

Procedure

In the `cassandra.yaml` file, set the following parameters:

Property	Description
cluster_name	Name of the cluster that this node is joining. Must be the same for every node in the cluster.
listen_address	The IP address or hostname that Cassandra binds to for connecting to other Cassandra nodes.
(Optional) broadcast_address	The IP address a node tells other nodes in the cluster to contact it by. It allows public and private address to be different. For example, use the

Property	Description
	broadcast_address parameter in topologies where not all nodes have access to other nodes by their private IP addresses. The default is the listen_address.
seed_provider	A -seeds list is comma-delimited list of hosts (IP addresses) that gossip uses to learn the topology of the ring. Every node should have the same list of seeds. In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is not recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).
storage_port	The inter-node communication port (default is 7000). Must be the same for every node in the cluster.
initial_token	For legacy clusters. Used in the single-node-per-token architecture, where a node owns exactly one contiguous range in the ring space.
num_tokens	For new clusters. Defines the number of tokens randomly assigned to this node on the ring when using virtual nodes (vnodes).

Configuring the heap dump directory

Analyzing the heap dump file can help troubleshoot memory problems. Cassandra starts Java with the option `-XX:+HeapDumpOnOutOfMemoryError`. Using this option triggers a heap dump in the event of an out-of-memory condition. The heap dump file consists of references to objects that cause the heap to overflow. By default, Cassandra puts the file a subdirectory of the working, root directory when running as a service. If Cassandra does not have write permission to the root directory, the heap dump fails. If the root directory is too small to accommodate the heap dump, the server crashes.

For a heap dump to succeed and to prevent crashes, configure a heap dump directory that meets these requirements:

- Accessible to Cassandra for writing
- Large enough to accommodate a heap dump

Base the size of the directory on the value of the Java `-mx` option.

Procedure

Set the location of the heap dump in the `cassandra-env.sh` file.

Configuration

1. Open the `cassandra-env.sh` file for editing.

```
set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
```

2. Scroll down to the comment about the heap dump path:

```
set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
```

3. On the line after the comment, set the `CASSANDRA_HEAPDUMP_DIR` to the path you want to use:

```
set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR  
CASSANDRA_HEAPDUMP_DIR =<path>
```

4. Save the `cassandra-env.sh` file and restart.

Configuring virtual nodes

Enabling virtual nodes on a new cluster

Generally when all nodes have equal hardware capability, they should have the same number of virtual nodes (vnodes). If the hardware capabilities vary among the nodes in your cluster, assign a proportional number of vnodes to the larger machines. For example, you could designate your older machines to use 128 vnodes and your new machines (that are twice as powerful) with 256 vnodes.

Procedure

Set the number of tokens on each node in your cluster with the `num_tokens` parameter in the `cassandra.yaml` file.

The recommended value is 256. Do not set the `initial_token` parameter.

Related information

[Install locations](#) on page 74

Enabling virtual nodes on an existing production cluster

You cannot directly convert a single-token nodes to a vnode. However, you can configure another data center configured with vnodes already enabled and let Cassandra automatic mechanisms distribute the existing data into the new nodes. This method has the least impact on performance.

Procedure

1. [Add a new data center to the cluster.](#)
2. Once the new data center with vnodes enabled is up, switch your clients to use the new data center.
3. Run a full repair with [nodetool repair](#).

This step ensures that after you move the client to the new data center that any previous writes are added to the new data center and that nothing else, such as hints, is dropped when you remove the old data center.

4. Update your schema to no longer reference the old data center.
5. Remove the old data center from the cluster.

See [Decommissioning a data center](#) on page 126.

Using multiple network interfaces

How to configure Cassandra for multiple network interfaces or when using different regions in cloud implementations.

You must configure settings in both the `cassandra.yaml` file and the property file (`cassandra-rackdc.properties` or `cassandra-topology.properties`) used by the snitch.

Configuring `cassandra.yaml` for multiple networks or across regions in cloud implementations

In multiple networks or cross-region cloud scenarios, communication between data centers can only take place using an external IP address. The external IP address is defined in the `cassandra.yaml` file using the `broadcast_address` setting. Configure each node as follows:

1. In the `cassandra.yaml`, set the `listen_address` to the *private* IP address of the node, and the `broadcast_address` to the *public* address of the node.

This allows Cassandra nodes to bind to nodes in another network or region, thus enabling multiple data-center support. For intra- network or region traffic, Cassandra switches to the private IP after establishing a connection.

2. Set the addresses of the seed nodes in the `cassandra.yaml` file to that of the *public* IP. Private IP are not routable between networks. For example:

```
seeds: 50.34.16.33, 60.247.70.52
```

Note: Do not make all nodes seeds, see [Internode communications \(gossip\)](#) on page 13.

3. Be sure that the `storage_port` or `ssl_storage_port` is open on the public IP firewall.

CAUTION: Be sure to enable encryption and authentication when using public IPs. See [Node-to-node encryption](#) on page 99. Another option is to use a custom VPN to have local, inter-region/ data center IPs.

Configuring the snitch for multiple networks

External communication between the data centers can only happen when using the `broadcast_address` (public IP).

The [GossipingPropertyFileSnitch](#) on page 22 is recommended for production. The `cassandra-rackdc.properties` file defines the data centers used by this snitch.

For each node in the network, specify its data center in `cassandra-rackdc.properties` file.

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

Network A	Network B
Node and data center:	Node and data center:
<ul style="list-style-type: none"> • node0 dc=DC_A_cassandra rack=RAC1 • node1 	<ul style="list-style-type: none"> • node0 dc=DC_A_cassandra rack=RAC1 • node1

Network A	Network B
dc=DC_A_cassandra rack=RAC1 <ul style="list-style-type: none"> • node2 dc=DC_B_cassandra rack=RAC1 <ul style="list-style-type: none"> • node3 dc=DC_B_cassandra rack=RAC1 <ul style="list-style-type: none"> • node4 dc=DC_A_analytics rack=RAC1 <ul style="list-style-type: none"> • node5 dc=DC_A_search rack=RAC1	dc=DC_A_cassandra rack=RAC1 <ul style="list-style-type: none"> • node2 dc=DC_B_cassandra rack=RAC1 <ul style="list-style-type: none"> • node3 dc=DC_B_cassandra rack=RAC1 <ul style="list-style-type: none"> • node4 dc=DC_A_analytics rack=RAC1 <ul style="list-style-type: none"> • node5 dc=DC_A_search rack=RAC1

Configuring the snitch for cross-region communication in cloud implementations

Note: Be sure to use the appropriate [snitch](#) for your implementation. If your deploying on Amazon EC2, see the instructions in [Ec2MultiRegionSnitch](#) on page 23.

In cloud deployments, the region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the us-east-1 region, us-east is the data center name and 1 is the rack location. (Racks are important for distributing replicas, but not for data center naming.)

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

For each node, specify its data center in the cassandra-rackdc.properties. The dc_suffix option defines the data centers used by the snitch. Any other lines are ignored.

Region: us-east	Region: us-west
Node and data center: <ul style="list-style-type: none"> • node0 dc_suffix=_1_cassandra <ul style="list-style-type: none"> • node1 dc_suffix=_1_cassandra <ul style="list-style-type: none"> • node2 dc_suffix=_2_cassandra <ul style="list-style-type: none"> • node3 dc_suffix=_2_cassandra <ul style="list-style-type: none"> • node4 dc_suffix=_1_analytics <ul style="list-style-type: none"> • node5 dc_suffix=_1_search	Node and data center: <ul style="list-style-type: none"> • node0 dc_suffix=_1_cassandra <ul style="list-style-type: none"> • node1 dc_suffix=_1_cassandra <ul style="list-style-type: none"> • node2 dc_suffix=_2_cassandra <ul style="list-style-type: none"> • node3 dc_suffix=_2_cassandra <ul style="list-style-type: none"> • node4 dc_suffix=_1_analytics <ul style="list-style-type: none"> • node5 dc_suffix=_1_search

Region: us-east	Region: us-west
This results in four us-east data centers:	This results in four us-west data centers:
us-east_1_cassandra us-east_2_cassandra us-east_1_analytics us-east_1_search	us-west_1_cassandra us-west_2_cassandra us-west_1_analytics us-west_1_search

Configuring logging

Cassandra provides logging functionality using Simple Logging Facade for Java (SLF4J) with a [logback](#) backend. Logs are written to the `system.log` file in the Cassandra logging directory. You can configure logging [programmatically](#) or manually. Manual ways to configure logging are:

- Run the `nodetool setlogginglevel` command.
- Configure the `logback-test.xml` or `logback.xml` file installed with Cassandra.
- Use the [JConsole](#) tool to configure logging through JMX.

Logback looks for `logback-test.xml` first, and then for `logback.xml` file.

The XML configuration files looks like this:

```
<configuration scan="true">
  <jmxConfigurator />
  <appender name="FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${cassandra.logdir}/system.log</file>
    <rollingPolicy
class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
      <fileNamePattern>${cassandra.logdir}/system.log.%i.zip</
fileNamePattern>
      <minIndex>1</minIndex>
      <maxIndex>20</maxIndex>
    </rollingPolicy>

    <triggeringPolicy
class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <maxFileSize>20MB</maxFileSize>
    </triggeringPolicy>
    <encoder>
      <pattern>%-5level [%thread] %date{ISO8601} %F:%L - %msg%n</pattern>
      <!-- old-style log format
      <pattern>%5level [%thread] %date{ISO8601} %F (line %L) %msg%n</
pattern>
      -->
    </encoder>
  </appender>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%-5level %date{HH:mm:ss,SSS} %msg%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="FILE" />
    <appender-ref ref="STDOUT" />
  </root>
```

```
<logger name="com.thinkaurelius.thrift" level="ERROR"/>
</configuration>
```

The appender configurations specify where to print the log and its configuration. The first appender directs logs to a file. The second appender directs logs to the console. You can change the following logging functionality:

- Rolling policy
 - The policy for rolling logs over to an archive
 - Location and name of the log file
 - Location and name of the archive
 - Minimum and maximum file size to trigger rolling
- Format of the message
- The log level

Log levels

The valid values for setting the log level include ALL for logging information at all levels, TRACE through ERROR, and OFF for no logging. TRACE creates the most verbose log, and ERROR, the least.

- ALL
- TRACE
- DEBUG
- INFO (Default)
- WARN
- ERROR
- OFF

Note: Increasing logging levels can generate heavy logging output on a moderately trafficked cluster.

You can use the `nodetool getlogginglevels` command to see the current logging configuration.

```
$ nodetool getlogginglevels
Logger Name      Log Level
ROOT            INFO
com.thinkaurelius.thrift  ERROR
```

Migrating to logback from log4j

If you upgrade from a previous version of Cassandra that used log4j, you can convert *log4j.properties* files to *logback.xml* using the logback [PropertiesTranslator](#) web-application.

Using log file rotation

The default policy rolls the `system.log` file after the size exceeds 20MB. Archives are compressed in zip format. Logback names the log files `system.log.1.zip`, `system.log.2.zip`, and so on. For more information, see [logback documentation](#).

Commit log archive configuration

Cassandra provides commit log archiving and point-in-time recovery. The commit log is archived at node startup and when a commit log is written to disk, or at a specified point-in-time. You configure this feature in the `commitlog_archiving.properties` configuration file.

The commands `archive_command` and `restore_command` expect only a single command with arguments. The parameters must be entered verbatim. STDOUT and STDIN or multiple commands cannot be executed. To workaround, you can script multiple commands and add a pointer to this file. To disable a command, leave it blank.

Procedure

- Archive a commit log segment:

Command	archive_command=	
Parameters	%path	Fully qualified path of the segment to archive.
	%name	Name of the commit log.
Example	archive_command=/bin/ln %path /backup/%name	

- Restore an archived commit log:

Command	restore_command=	
Parameters	%from	Fully qualified path of the an archived commitlog segment fr
	%t	Name of live commit log directory.
Example	restore_command=cp -f %from %to	

- Set the restore directory location:

Command	restore_directories=
Format	restore_directories=restore_directory_location

- Restore mutations created up to and including the specified timestamp:

Command	restore_point_in_time=
Format	<timestamp> (YYYY:MM:DD HH:MM:SS)
Example	restore_point_in_time=2013:12:11 17:00:00

Restore stops when the first client-supplied timestamp is greater than the restore point timestamp. Because the order in which Cassandra receives mutations does not strictly follow the timestamp order, this can leave some mutations unrecovered.

Generating tokens

If not using virtual nodes (vnodes), you must calculate tokens for your cluster.

The following topics in the Cassandra 1.1 documentation provide conceptual information about tokens:

- [Data Distribution in the Ring](#)
- [Replication Strategy](#)

About calculating tokens for single or multiple data centers in Cassandra 1.2 and later

- Single data center deployments: calculate tokens by dividing the hash range by the number of nodes in the cluster.
- Multiple data center deployments: calculate the tokens for each data center so that the hash range is evenly divided for the nodes in each data center.

Configuration

For more explanation, see be sure to read the conceptual information mentioned above.

The method used for calculating tokens depends on the type of partitioner:

Calculating tokens for the Murmur3Partitioner

Use this method for generating tokens when you are **not** using virtual nodes (vnodes) and using the [Murmur3Partitioner](#) (default). This partitioner uses a maximum possible range of hash values from -2^{63} to $+2^{63}-1$. To calculate tokens for this partitioner:

```
$ python -c "print [str(((2**64 / number_of_tokens) * i) - 2**63) for i in range(number_of_tokens)]"
```

For example, to generate tokens for 6 nodes:

```
$ python -c "print [str(((2**64 / 6) * i) - 2**63) for i in range(6)]"
```

The command displays the token for each node:

```
[ '-9223372036854775808', '-6148914691236517206', '-3074457345618258604', '-2', '3074457345618258600', '6148914691236517202' ]
```

Calculating tokens for the RandomPartitioner

To calculate tokens when using the [RandomPartitioner](#) in Cassandra 1.2 clusters, use the [Cassandra 1.1 Token Generating Tool](#).

Hadoop support

Cassandra support for integrating Hadoop with Cassandra includes:

- MapReduce

Notice: Apache Pig is no longer supported as of Cassandra 3.0.

You can use Cassandra 2.1 with Hadoop 2.x or 1.x with some restrictions.

- [Isolate Cassandra and Hadoop](#) nodes in separate data centers.
- Before starting the data centers of Cassandra/Hadoop nodes, disable virtual nodes (vnodes).

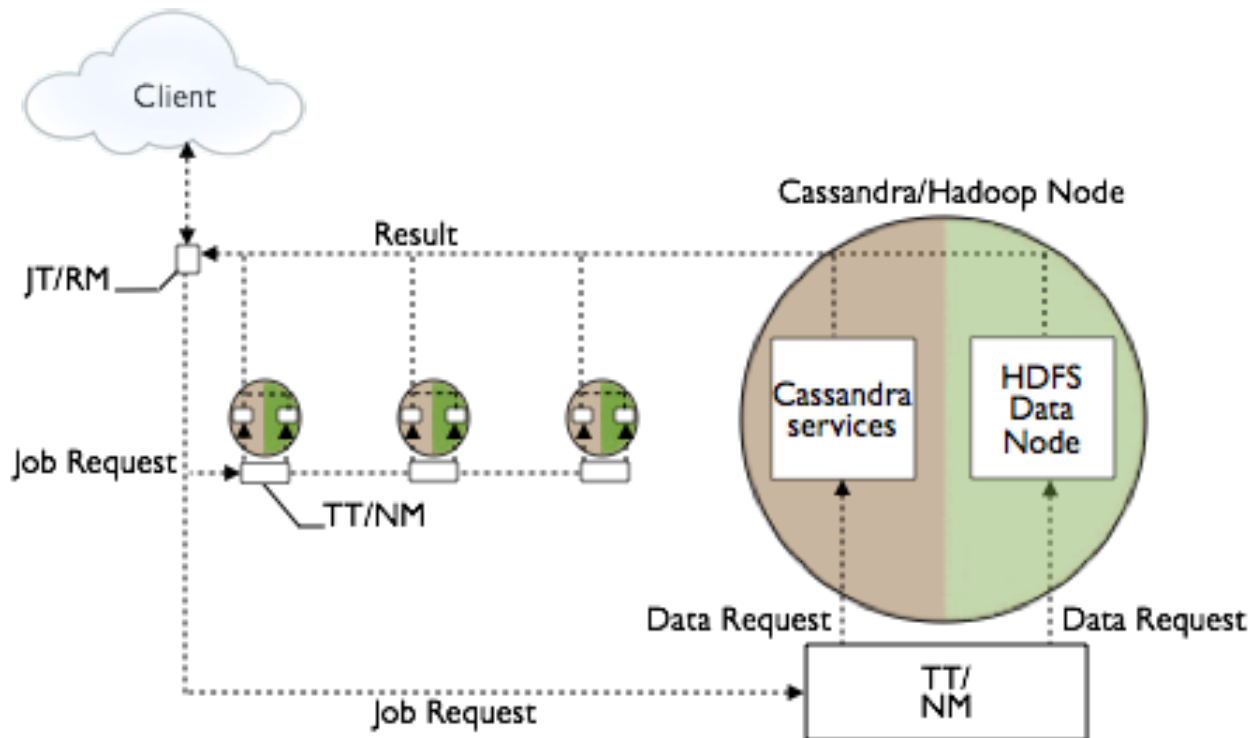
To disable virtual nodes:

1. In the `cassandra.yaml` file, set `num_tokens` to 1.
2. Uncomment the `initial_token` property and set it to 1 or to the value of a generated token for a multi-node cluster.
3. Start the cluster for the first time.

Do not disable or enable vnodes on an existing cluster.

Setup and configuration, described in the [Apache docs](#), involves overlaying a Hadoop cluster on Cassandra nodes, configuring a separate server for the Hadoop NameNode/JobTracker, and installing a Hadoop TaskTracker and Data Node on each Cassandra node. The nodes in the Cassandra data center can draw from data in the HDFS Data Node as well as from Cassandra. The Job Tracker/Resource Manager (JT/RM) receives MapReduce input from the client application. The JT/RM sends a MapReduce job request to the Task Trackers/Node Managers (TT/NM) and an optional clients MapReduce. The data is written to Cassandra and results sent back to the client.

MapReduce Process in a Cassandra/Hadoop Cluster



The Apache docs also cover how to get configuration and integration support.

Input and Output Formats

Hadoop jobs can receive data from CQL tables and indexes and you can load data into Cassandra from a Hadoop job. Cassandra 2.1 supports the following formats for these tasks:

- CQL partition input format: ColumnFamilyInputFormat class
- BulkOutputFormat class

Cassandra 2.1.1 and later supports the CqlOutputFormat, which is the CQL-compatible version of the [BulkOutputFormat class](#). The CqlOutputFormat acts as a Hadoop-specific OutputFormat. Reduce tasks can store keys (and corresponding bound variable values) as CQL rows (and respective columns) in a given CQL table.

Running the wordcount example

Wordcount example JARs are located in the `examples` directory of the Cassandra source code installation. There are CQL and legacy examples in the `hadoop_cql3_word_count` and `hadoop_word_count` subdirectories, respectively. Follow instructions in the readme files.

Isolating Hadoop and Cassandra workloads

When you create a keyspace using CQL, Cassandra creates a virtual data center for a cluster, even a one-node cluster, automatically. You assign nodes that run the same type of workload to the same data center. The separate, virtual data centers for different types of nodes segregate workloads running Hadoop from those running Cassandra. Segregating workloads ensures that only one type of workload is active per data center. Separating nodes running a sequential data load, from nodes running any other type of workload, such as Cassandra real-time OLTP queries is a best practice.

Initializing a cluster

Initializing a multiple node cluster (single data center)

This topic contains information for deploying a Cassandra cluster with a single data center. If you're new to Cassandra, and haven't set up a cluster, see [Cassandra and DataStax Enterprise Essentials](#) or [10 Minute Cassandra Walkthrough](#).

Prerequisites

Each node must be correctly configured before starting the cluster. You must determine or perform the following before starting the cluster:

- A good understanding of how Cassandra works. Be sure to read at least [Understanding the architecture](#) on page 11, [Data replication](#) on page 17, and [Cassandra's rack feature](#).
- Install Cassandra on each node.
- Choose a name for the cluster.
- Get the IP address of each node.
- Determine which nodes will be seed nodes. **Do not make all nodes seed nodes.** Please read [Internode communications \(gossip\)](#) on page 13.
- Determine the [snitch](#) and [replication strategy](#). The [GossipingPropertyFileSnitch](#) on page 22 and [NetworkTopologyStrategy](#) are recommended for production environments.
- If using multiple data centers, determine a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102. Choose the name carefully; renaming a data center is not possible.
- Other possible configuration settings are described in the [cassandra.yaml configuration file](#) and property files such as `cassandra-rackdc.properties`.

This example describes installing a 6 node cluster spanning 2 racks in a single data center. Each node is configured to use the [GossipingPropertyFileSnitch](#) and 256 virtual nodes (vnodes).

In Cassandra, the term data center is a grouping of nodes. Data center is synonymous with replication group, that is, a grouping of nodes configured together for replication purposes.

Procedure

1. Suppose you install Cassandra on these nodes:

```
node0 110.82.155.0 (seed1)
node1 110.82.155.1
node2 110.82.155.2
node3 110.82.156.3 (seed2)
node4 110.82.156.4
node5 110.82.156.5
```

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

2. If you have a firewall running in your cluster, you must open certain ports for communication between the nodes. See [Configuring firewall port access](#) on page 102.
3. If Cassandra is running, you must stop the server and clear the data:

Doing this removes the default `cluster_name` (Test Cluster) from the system table. All nodes must use the same cluster name.

Package installations:

a) Stop Cassandra:

```
$ sudo service cassandra stop
```

b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

Tarball installations:

a) Stop Cassandra:

```
$ ps aux | grep cassandra
$ sudo kill pid
```

b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

4. Set the properties in the `cassandra.yaml` file for each node:

Note: After making any changes in the `cassandra.yaml` file, you must restart the node for the changes to take effect.

Properties to set:

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

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

- `listen_address`:
If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the `listen_address`.
- `endpoint_snitch`: *name of snitch* (See [endpoint_snitch](#).) If you are changing snitches, see [Switching snitches](#) on page 127.
- `auto_bootstrap`: `false` (Add this setting **only** when initializing a fresh cluster with no data.)

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

Example:

```
cluster_name: 'MyCassandraCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "110.82.155.0,110.82.155.3"
listen_address:
rpc_address: 0.0.0.0
endpoint_snitch: GossipingPropertyFileSnitch
```

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

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

Initializing a cluster

```
rack=RAC1
```

6. After you have installed and configured Cassandra on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

Note: If the node has restarted because of automatic restart, you must first stop the node and clear the data directories, as described [above](#).

Package installations:

```
$ sudo service cassandra start
```

Tarball installations:

```
$ cd install_location
$ bin/cassandra
```

7. To check that the ring is up and running, run:

Package installations:

```
$ nodetool status
```

Tarball installations:

```
$ cd install_location
$ bin/nodetool status
```

Each node should be listed and its status and state should be UN (Up Normal).

```
paul@ubuntu:~/cassandra-2.1.0$ bin/nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address                Load          Tokens       Owns    Host ID                               Rack
UN  10.194.171.160          53.98 KB      256         0.8%    a9fa31c7-f3c0-44d1-b8e7-a2628867840c rack1
UN  10.196.14.48            93.62 KB      256         9.9%    f5bb146c-db51-475c-a44f-9facf2f1ad6e rack1
UN  10.196.14.239           83.98 KB      256         8.2%    b8e6748f-ec11-410d-c94f-b8e7d88a28e7 rack1
...
```

Related information

[Install locations](#) on page 74

Initializing a multiple node cluster (multiple data centers)

This topic contains information for deploying a Cassandra cluster with multiple data centers. If you're new to Cassandra, and haven't set up a cluster, see [Cassandra and DataStax Enterprise Essentials](#) or [10 Minute Cassandra Walkthrough](#).

This example describes installing a six node cluster spanning two data centers. Each node is configured to use the [GossipingPropertyFileSnitch](#) (multiple rack aware) and 256 virtual nodes (vnodes).

In Cassandra, the term data center is a grouping of nodes. Data center is synonymous with replication group, that is, a grouping of nodes configured together for replication purposes.

Prerequisites

Each node must be correctly configured before starting the cluster. You must determine or perform the following before starting the cluster:

- A good understanding of how Cassandra works. Be sure to read at least [Understanding the architecture](#) on page 11, [Data replication](#) on page 17, and [Cassandra's rack feature](#).
- Install Cassandra on each node.
- Choose a name for the cluster.
- Get the IP address of each node.
- Determine which nodes will be seed nodes. **Do not make all nodes seed nodes.** Please read [Internode communications \(gossip\)](#) on page 13.
- Determine the [snitch](#) and [replication strategy](#). The [GossipingPropertyFileSnitch](#) on page 22 and [NetworkTopologyStrategy](#) are recommended for production environments.
- If using multiple data centers, determine a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102. Choose the name carefully; renaming a data center is not possible.
- Other possible configuration settings are described in the [cassandra.yaml configuration file](#) and property files such as `cassandra-rackdc.properties`.

Procedure

1. Suppose you install Cassandra on these nodes:

```
node0 10.168.66.41 (seed1)
node1 10.176.43.66
node2 10.168.247.41
node3 10.176.170.59 (seed2)
node4 10.169.61.170
node5 10.169.30.138
```

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

2. If you have a firewall running in your cluster, you must open certain ports for communication between the nodes. See [Configuring firewall port access](#) on page 102.
3. If Cassandra is running, you must stop the server and clear the data:

Doing this removes the default `cluster_name` (Test Cluster) from the system table. All nodes must use the same cluster name.

Package installations:

- a) Stop Cassandra:

```
$ sudo service cassandra stop
```

- b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

Tarball installations:

- a) Stop Cassandra:

```
$ ps aux | grep cassandra
$ sudo kill pid
```

- b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

4. Set the properties in the `cassandra.yaml` file for each node:

Note: After making any changes in the `cassandra.yaml` file, you must restart the node for the changes to take effect.

Properties to set:

Initializing a cluster

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

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

- `listen_address`:
If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the `listen_address`.
- `endpoint_snitch`: *name of snitch* (See [endpoint_snitch](#).) If you are changing snitches, see [Switching snitches](#) on page 127.
- `auto_bootstrap`: `false` (Add this setting **only** when initializing a fresh cluster with no data.)

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

Example:

```
cluster_name: 'MyCassandraCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "10.168.66.41,10.176.170.59"
listen_address:
endpoint_snitch: GossipingPropertyFileSnitch
```

Note: Include at least one node from *each* data center.

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

Nodes 0 to 2

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

Nodes 3 to 5

```
## Indicate the rack and dc for this node
dc=DC2
rack=RAC1
```

6. After you have installed and configured Cassandra on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

Note: If the node has restarted because of automatic restart, you must first stop the node and clear the data directories, as described [above](#).

Package installations:

```
$ sudo service cassandra start
```

Tarball installations:

```
$ cd install_location
$ bin/cassandra
```

7. To check that the ring is up and running, run:

Package installations:


```
$ nodetool status
```

Tarball installations:

```
$ cd install_location
$ bin/nodetool status
```

Each node should be listed and it's status and state should be UN (Up Normal).

```
paul@ubuntu:~/cassandra-2.1.0$ bin/nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address                Load          Tokens       Owns    Host ID                               Rack
UN  10.194.171.160          53.98 KB      256          0.8%    a9fa31c7-f3c0-44d1-b8e7-a2628867840c rack1
UN  10.196.14.48             93.62 KB      256          9.9%    f5bb146c-db51-475c-a44f-9facf2f1ad6e rack1
UN  10.196.14.239            83.98 KB      256          8.2%    b8e6748f-ec11-410d-c94f-b8e7d88a28e7 rack1
...

```

Related information

[Install locations](#) on page 74

Starting and stopping Cassandra

Starting Cassandra as a service

Start the Cassandra Java server process for packaged installations.

Startup scripts are provided in the `/etc/init.d` directory. The service runs as the *cassandra* user.

Procedure

You must have root or sudo permissions to start Cassandra as a service.

On initial start-up, each node must be started one at a time, starting with your seed nodes:

```
$ sudo service cassandra start
```

On Enterprise Linux systems, the Cassandra service runs as a java process.

Starting Cassandra as a stand-alone process

Start the Cassandra Java server process for tarball installations.

Procedure

On initial start-up, each node must be started one at a time, starting with your seed nodes.

- To start Cassandra in the background:

```
$ cd install_location
$ bin/cassandra
```

- To start Cassandra in the foreground:

Initializing a cluster

```
$ cd install_location
$ bin/cassandra -f
```

Stopping Cassandra as a service

Stopping the Cassandra Java server process on packaged installations.

Procedure

You must have root or sudo permissions to stop the Cassandra service:

```
$ sudo service cassandra stop
```

Stopping Cassandra as a stand-alone process

Stop the Cassandra Java server process on tarball installations.

Procedure

Find the Cassandra Java process ID (PID), and then kill the process using its PID number:

```
$ ps aux | grep cassandra
$ sudo kill pid
```

Clearing the data as a service

Remove all data from a package installation.

Procedure

To clear the data from the **default** directories:

After [stopping](#) the service, run the following command:

```
$ sudo rm -rf /var/lib/cassandra/*
```

Note: If you are clearing data from an AMI installation for restart, you need to [preserve the log files](#).

Clearing the data as a stand-alone process

Remove all data from a tarball installation.

Procedure

To clear all data from the **default** directories, including the commitlog and saved_caches:

1. After [stopping](#) the process, run the following command from the install directory:

```
$ cd install_location
$ sudo rm -rf data/*
```

2. To clear the only the data directory:

```
$ cd install_location
$ sudo rm -rf data/data/*
```

Operations

Adding or removing nodes, data centers, or clusters

Adding nodes to an existing cluster

Virtual nodes (vnodes) greatly simplify adding nodes to an existing cluster:

- Calculating tokens and assigning them to each node is no longer required.
- Rebalancing a cluster is no longer necessary because a node joining the cluster assumes responsibility for an even portion of the data.

For a detailed explanation about how vnodes work, see [Virtual nodes](#) on page 16.

If you are using racks, you can safely bootstrap two nodes at a time when both nodes are on the same rack.

Note: If you do not use vnodes, see [Adding or replacing single-token nodes](#) on page 130.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See [Installing earlier releases](#).

1. Install Cassandra on the new nodes, but do not start Cassandra.

For Debian or Windows installations, Cassandra starts automatically and you must [stop](#) the node and [clear](#) the data.

2. Set the following properties in the `cassandra.yaml` and, depending on the snitch, the `cassandra-topology.properties` or `cassandra-rackdc.properties` configuration files:

- [auto_bootstrap](#) - If this option has been set to false, you must set it to true. This option is not listed in the default `cassandra.yaml` configuration file and defaults to true.
- [cluster_name](#) - The name of the cluster the new node is joining.
- [listen_address/broadcast_address](#) - Can usually be left blank. Otherwise, use IP address or host name that other Cassandra nodes use to connect to the new node.
- [endpoint_snitch](#) - The snitch Cassandra uses for locating nodes and routing requests.
- [num_tokens](#) - The number of vnodes to assign to the node. If the hardware capabilities vary among the nodes in your cluster, you can assign a proportional number of vnodes to the larger machines.
- [seed_provider](#) - Make sure that the new node lists at least one node in the existing cluster. The `-seeds` list determines which nodes the new node should contact to learn about the cluster and establish the gossip process.

Note: Seed nodes cannot bootstrap. Make sure the new node is not listed in the `-seeds` list. **Do not make all nodes seed nodes.** Please read [Internode communications \(gossip\)](#) on page 13.

- Change any other non-default settings you have made to your existing cluster in the `cassandra.yaml` file and `cassandra-topology.properties` or `cassandra-`

`rackdc.properties` files. Use the `diff` command to find and merge (by head) any differences between existing and new nodes.

The location of the `cassandra-topology.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-topology.properties</code>
Tarball installations	<code>install_location/conf/cassandra-topology.properties</code>
Windows installations	

The location of the `cassandra-rackdc.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-rackdc.properties</code>
Tarball installations	<code>install_location/conf/cassandra-rackdc.properties</code>
Windows installations	

3. Use [nodetool status](#) to verify that the node is fully bootstrapped and all other nodes are up (UN) and not in any other state.
4. After all new nodes are running, run [nodetool cleanup](#) on each of the previously existing nodes to remove the keys that no longer belong to those nodes. Wait for cleanup to complete on one node before running `nodetool cleanup` on the next node.

Cleanup can be safely postponed for low-usage hours.

Related tasks

[Starting Cassandra as a service](#) on page 119

[Starting Cassandra as a stand-alone process](#) on page 119

Related reference

[The nodetool utility](#) on page 160

Related information

[Install locations](#) on page 74

Adding a data center to a cluster

Steps for adding a data center to an existing cluster.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See [Installing earlier releases](#).

1. Ensure that you are using [NetworkTopologyStrategy](#) for all of your keyspaces.
2. For each node, set the following properties in the `cassandra.yaml` file:
 - a) Add (or edit) `auto_bootstrap: false`.

By default, this setting is true and not listed in the `cassandra.yaml` file. Setting this parameter to false prevents the new nodes from attempting to get all the data from the other nodes in the data center. When you run [nodetool rebuild](#) in the last step, each node is properly mapped.

- b) Set other properties, such as `-seeds` and `endpoint_snitch`, to match the cluster settings.

For more guidance, see [Initializing a multiple node cluster \(multiple data centers\)](#) on page 116.

Note: Do not make all nodes seeds, see [Internode communications \(gossip\)](#) on page 13.

c) If you want to enable vnodes, set `num_tokens`.

The recommended value is 256. Do not set the `initial_token` parameter.

3. Update the relevant property file for snitch used on all servers to include the new nodes. You do not need to restart.

- [GossipingPropertyFileSnitch](#) on page 22: `cassandra-rackdc.properties`
- [PropertyFileSnitch](#): `cassandra-topology.properties`

4. Ensure that your clients are configured correctly for the new cluster:

- If your client uses the DataStax Java, C#, or Python driver, set the load-balancing policy to `DCAwareRoundRobinPolicy` ([Java](#), [C#](#), [Python](#)).
- If you are using another client such as Hector, make sure it does not auto-detect the new nodes so that they aren't contacted by the client until explicitly directed. For example if you are using Hector, use `setHostConfig.setAutoDiscoverHosts(false)`; . If you are using Astyanax, use `ConnectionPoolConfigurationImpl.setLocalDatacenter("<data center name">")` to ensure you are connecting to the specified data center.
- If you are using Astyanax 2.x, with integration with the DataStax Java Driver 2.0, you can set the load-balancing policy to `DCAwareRoundRobinPolicy` by calling `JavaDriverConfigBuilder.withLoadBalancingPolicy()`.

```
AstyanaxContext<Keyspace> context = new AstyanaxContext.Builder()
    ...
    .withConnectionPoolConfiguration(new JavaDriverConfigBuilder()
        .withLoadBalancingPolicy(new TokenAwarePolicy(new
            DCAwareRoundRobinPolicy()))
        .build())
    ...
```

5. If using a QUORUM [consistency level](#) for reads or writes, check the LOCAL_QUORUM or EACH_QUORUM consistency level to see if the level meets your requirements for multiple data centers.

6. Start Cassandra on the new nodes.

7. After all nodes are running in the cluster:

a) Change the [keyspace properties](#) to specify the desired replication factor for the new data center.

For example, set strategy options to DC1:2, DC2:2.

For more information, see [ALTER KEYSPACE](#).

b) Run [nodetool rebuild](#) specifying the existing data center on all nodes in the new data center:

```
$ nodetool rebuild -- name_of_existing_data_center
```

Otherwise, requests to the new data center with LOCAL_ONE or ONE consistency levels may fail if the existing data centers are not completely in-sync.

You can run rebuild on one or more nodes at the same time. The choices depends on whether your cluster can handle the extra IO and network pressure of running on multiple nodes. Running on one node at a time has the least impact on the existing cluster.

Attention: If you don't specify the existing data center in the command line, the new nodes will appear to rebuild successfully, but will not contain any data.

8. Change to true or remove [auto_bootstrap: false](#) in the `cassandra.yaml` file.

Returns this parameter to its normal setting so the nodes can get all the data from the other nodes in the data center if restarted.

Related tasks

[Starting Cassandra as a service](#) on page 119

[Starting Cassandra as a stand-alone process](#) on page 119

Related information

[Install locations](#) on page 74

Replacing a dead node or dead seed node

Steps to replace a node that has died for some reason, such as hardware failure. Prepare and start the replacement node, then attach it to the cluster. After the replacement node is running in the cluster, [remove the dead node](#).

Replacing a dead seed node

1. Promote an existing node to a seed node by adding its IP address to [-seeds](#) list and remove (demote) the IP address of the dead seed node from the `cassandra.yaml` file for each node in the cluster.
2. Replace the dead node, as described in the next section.

Replacing a dead node

You must prepare and start the replacement node, integrate it into the cluster, and then remove the dead node.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See [Installing earlier releases](#).

1. Confirm that the node is dead using [nodetool status](#):

The `nodetool` command shows a down status for the dead node (DN):

```
paul@ubuntu:~/cassandra-2.1.0$ bin/nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address                      Load          Tokens      Owns    Host ID                               Rack
UN  10.194.171.160                53.98 KB      256         0.8%    a9fa31c7-f3c0-44d1-b8e7-a2628867840c rack1
UN  10.196.14.48                   93.62 KB      256         9.9%    f5bb146c-db51-475c-a44f-9facf2f1ad6e rack1
DN  10.196.14.239                  ?             256         8.2%    null
```

2. Note the `Address` of the dead node; it is used in [step 5](#).
3. Install Cassandra on the new node, but do not start Cassandra.

If using the Debian/Ubuntu install, Cassandra starts automatically and you must [stop](#) the node and [clear](#) the data.

4. Set the following properties in the `cassandra.yaml` and, depending on the snitch, the `cassandra-topology.properties` or `cassandra-rackdc.properties` configuration files:
 - [auto_bootstrap](#) - If this option has been set to false, you must set it to true. This option is not listed in the default `cassandra.yaml` configuration file and defaults to true.
 - [cluster_name](#) - The name of the cluster the new node is joining.
 - [listen_address/broadcast_address](#) - Can usually be left blank. Otherwise, use IP address or host name that other Cassandra nodes use to connect to the new node.
 - [endpoint_snitch](#) - The snitch Cassandra uses for locating nodes and routing requests.

- [num_tokens](#) - The number of vnodes to assign to the node. If the hardware capabilities vary among the nodes in your cluster, you can assign a proportional number of vnodes to the larger machines.
- [seed_provider](#) - Make sure that the new node lists at least one node in the existing cluster. The -seeds list determines which nodes the new node should contact to learn about the cluster and establish the gossip process.

Note: Seed nodes cannot bootstrap. Make sure the new node is not listed in the -seeds list. **Do not make all nodes seed nodes.** Please read [Internode communications \(gossip\)](#) on page 13.

- Change any other non-default settings you have made to your existing cluster in the `cassandra.yaml` file and `cassandra-topology.properties` or `cassandra-rackdc.properties` files. Use the `diff` command to find and merge (by head) any differences between existing and new nodes.

5. Start the replacement node with the [replace_address](#) option:

- Package installations: Add the following option to `cassandra-env.sh` file:

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

- Tarball installations: Start Cassandra with this option:

```
$ sudo bin/cassandra -Dcassandra.replace_address=address_of_dead_node
```

6. If using a packaged install, after the new node finishes bootstrapping, remove the option you added in [step 5](#).

What to do next

- Remove the old node's IP address from the `cassandra-topology.properties` or `cassandra-rackdc.properties` file.

CAUTION: Wait at least 72 hours to ensure that old node information is removed from [gossip](#). If removed from the property file too soon, problems may result.

- [Remove the node.](#)

The location of the `cassandra-topology.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-topology.properties</code>
Tarball installations	<code>install_location/conf/cassandra-topology.properties</code>
Windows installations	

Replacing a running node

Steps to replace a node with a new node, such as when updating to newer hardware or performing proactive maintenance.

You must prepare and start the replacement node, integrate it into the cluster, and then decommission the old node.

Note: To change the IP address of a node, simply change the IP of node and then restart Cassandra. If you change the IP address of a seed node, you must update the -seeds parameter in the [seed_provider](#) for each node in the `cassandra.yaml` file.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See [Installing earlier releases](#).

1. Prepare and start the replacement node, as described in [Adding nodes to an existing cluster](#).

Note: If not using vnodes, see [Adding or replacing single-token nodes](#) on page 130.

2. Confirm that the replacement node is alive:

- Run `nodetool ring` if not using vnodes.
- Run `nodetool status` if using vnodes.

The status should show:

- `nodetool ring: Up`
- `nodetool status: UN`

3. Note the `Host ID` of the node; it is used in the next step.
4. Using the `Host ID` of the original node, decommission the original node from the cluster using the `nodetool decommission` command.

Related tasks

[Removing a node](#) on page 127

Moving a node from one rack to another

A common task is moving a node from one rack to another. For example, when using `GossipPropertyFileSnitch`, a common error is mistakenly placing a node in the wrong rack. To correct the error, use one of the following procedures.

- The preferred method is to [decommission the node](#) and re-add it to the correct rack and data center.
 - This method takes longer to complete than the alternative method. Data is moved that the decommissioned node doesn't need anymore. Then the node gets new data while bootstrapping. The alternative method does both operations simultaneously.
- An alternative method is to [update the node's topology](#) and restart the node. Once the node is up, run a [full repair](#) on the cluster.

Note: This method is not preferred because until the repair is completed, the node might blindly handle requests for data the node doesn't yet have.

Decommissioning a data center

Steps to properly remove a data center so no information is lost.

Procedure

1. Make sure no clients are still writing to any nodes in the data center.
2. Run a full repair with `nodetool repair`.

This ensures that all data is propagated from the data center being decommissioned.

3. [Change all keyspaces](#) so they no longer reference the data center being removed.
4. Run `nodetool decommission` on every node in the data center being removed.

Removing a node

Use these instructions when you want to remove nodes to reduce the size of your cluster, not for [replacing a dead node](#).

Attention: If you are not using [virtual nodes](#) (vnodes), you must rebalance the cluster.

Procedure

- Check whether the node is up or down using [nodetool status](#):

The `nodetool` command shows the status of the node (UN=up, DN=down):

```
paul@ubuntu:~/cassandra-1.2.0$ bin/nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
||/ State=Normal/Leaving/Joining/Moving
-- Address                Load          Tokens     Owns    Host ID                               Rack
UN  10.194.171.160         53.98 KB      256        0.8%    a9fa31c7-f3c0-44d1-b8e7-a2628867840c rack1
UN  10.196.14.48           93.62 KB      256        9.9%    f5bb146c-db51-475c-a44f-9facf2f1ad6e rack1
DN  10.196.14.239          ?             256        8.2%    null
```

- If the node is up, run [nodetool decommission](#).

This assigns the ranges that the node was responsible for to other nodes and replicates the data appropriately.

Use [nodetool netstats](#) to monitor the progress.

- If the node is down, choose the appropriate option:
 - If the cluster uses vnodes, remove the node using the [nodetool removemode](#) command.
 - If the cluster does not use vnodes, before running the [nodetool removemode](#) command, [adjust your tokens to evenly distribute](#) the data across the remaining nodes to avoid creating a hot spot.
- If the node does not stop streaming data to other nodes because gossip has stale state data for the node, run [nodetool assassinate](#).

Switching snitches

Because [snitches](#) determine how Cassandra distributes replicas, the procedure to switch snitches depends on whether or not the topology of the cluster will change:

- If data has not been inserted into the cluster, there is no change in the network topology. This means that you only need to set the snitch; no other steps are necessary.
- If data has been inserted into the cluster, it's possible that the topology has changed and you will need to perform additional steps.

A change in topology means that there is a change in the data centers and/or racks where the nodes are placed. Topology changes may occur when the replicas are placed in different places by the new snitch. Specifically, the replication strategy places the replicas based on the information provided by the new snitch. The following examples demonstrate the differences:

- No topology change**

Suppose you have 5 nodes using the [SimpleSnitch](#) in a single data center and you change to 5 nodes in 1 data center and 1 rack using a network snitch such as the [GossipingPropertyFileSnitch](#).

- Topology change**

Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 2 data centers using the [PropertyFileSnitch](#).

Note: If splitting from one data center to two, you need to change the schema for the [keyspace](#) that are splitting. Additionally, the data center names must change accordingly.

- **Topology change**

Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 1 data center and 2 racks using the [RackInferringSnitch](#).

Procedure

1. Create a properties file with data center and rack information.

- `cassandra-rackdc.properties`

[GossipingPropertyFileSnitch](#) on page 22, [Ec2Snitch](#), and [Ec2MultiRegionSnitch](#) only.

- `cassandra-topology.properties`

All other network snitches.

2. Copy the `cassandra-rackdc.properties` or `cassandra-topology.properties` file to the Cassandra configuration directory on all the cluster's nodes. They won't be used until the new snitch is enabled.

The location of the `cassandra-topology.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-topology.properties</code>
Tarball installations	<code>install_location/conf/cassandra-topology.properties</code>
Windows installations	

The location of the `cassandra-rackdc.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-rackdc.properties</code>
Tarball installations	<code>install_location/conf/cassandra-rackdc.properties</code>
Windows installations	

3. Change the snitch for each node in the cluster in the node's `cassandra.yaml` file. For example:

```
endpoint_snitch: GossipingPropertyFileSnitch
```

4. If the topology has not changed, you can restart each node one at a time.

Any change in the `cassandra.yaml` file requires a node restart.

5. If the topology of the network has changed:

- a) Shut down all the nodes, then restart them.
- b) Run a [sequential repair](#) and [nodetool cleanup](#) on each node.

Related concepts

[Snitches](#) on page 20

Changing keyspace strategy

A keyspace is created with a strategy. For development work, the `SimpleStrategy` class is acceptable. For production work, the `NetworkTopologyStrategy` class must be set. To change the strategy, two steps are required.

Procedure

- [Change the snitch](#) to a network-aware setting.
- Alter the keyspace properties using the `ALTER KEYSPACE` command. For example, the keyspace cycling set to `SimpleStrategy` is switched to `NetworkTopologyStrategy`.

```
cqlsh> ALTER KEYSPACE cycling WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'DC1' : 3, 'DC2' : 2 };
```

Edge cases for transitioning or migrating a cluster

The information in this topic is intended for the following types of scenarios (without any interruption of service):

- Transition a cluster on EC2 to a cluster on Amazon virtual private cloud (VPC).
- Migrate from a cluster when the network separates the current cluster from the future location.
- Migrate from an early Cassandra cluster to a recent major version.

Procedure

The following method ensures that if something goes wrong with the new cluster, you still have the existing cluster until you no longer need it.

1. Set up and configure the new cluster as described in [Initializing a cluster](#) on page 114.

Note: If you're not using vnodes, be sure to configure the token ranges in the new nodes to match the ranges in the old cluster.

2. Set up the schema for the new cluster using [CQL](#).
3. Configure your client to write to both clusters.

Depending on how the writes are done, code changes may be needed. Be sure to use identical consistency levels.

4. Ensure that the data is flowing to the new nodes so you won't have any gaps when you copy the snapshots to the new cluster in step 6.
5. [Snapshot](#) the old EC2 cluster.
6. Copy the data files from your keyspaces to the nodes.

- If not using vnodes and the if the node ratio is 1:1, it's simpler and more efficient to simply copy the data files to their matching nodes.
- If the clusters are different sizes or if you are using vnodes, use the [sstableloader \(Cassandra bulk loader\)](#) on page 246 (`sstableloader`).

7. You can either switch to the new cluster all at once or perform an incremental migration.

For example, to perform an incremental migration, you can set your client to designate a percentage of the reads that go to the new cluster. This allows you to test the new cluster before decommissioning the old cluster.

- Decommission the old cluster, as described in [Decommissioning a data center](#) on page 126.

Adding or replacing single-token nodes

This topic applies only to clusters using single-token architecture, not vnodes.

About adding Capacity to an Existing Cluster

Cassandra allows you to add capacity to a cluster by introducing new nodes to the cluster in stages and by adding an entire data center. When a new node joins an existing cluster, it needs to know:

- Its position in the ring and the range of data it is responsible for, which is assigned by the [initial_token](#) and the [partitioner](#).
- The [seed](#) nodes to contact for learning about the cluster and establish the [gossip process](#).
- The name of the cluster it is joining and how the node should be addressed within the cluster.
- Any other non-default settings made to `cassandra.yaml` on your existing cluster.

When you add one or more nodes to a cluster, you must calculate the tokens for the new nodes. Use one of the following approaches:

Add capacity by doubling the cluster size

Adding capacity by doubling (or tripling or quadrupling) the number of nodes is less complicated when assigning tokens. Existing nodes can keep their existing token assignments, and new nodes are assigned tokens that bisect (or trisect) the existing token ranges. For example, when you generate tokens for six nodes, three of the generated token values will be the same as if you generated for three nodes. To clarify, you first obtain the token values that are already in use, and then assign the newly calculated token values to the newly added nodes.

Recalculate new tokens for all nodes and move nodes around the ring

When increases capacity by a non-uniform number of nodes, you must recalculate tokens for the entire cluster, and then use [nodetool move](#) to assign the new tokens to the existing nodes. After all nodes are restarted with their new token assignments, run a [nodetool cleanup](#) to remove unused keys on all nodes. These operations are resource intensive and should be done during low-usage times.

Add one node at a time and leave the `initial_token` property empty

When the [initial_token](#) is empty, Cassandra splits the token range of the heaviest loaded node and places the new node into the ring at that position. This approach is unlikely to result in a perfectly balanced ring, but will alleviate hot spots.

Adding Nodes to a Cluster

- Install Cassandra on the new nodes, but do not start them.
- Calculate the tokens for the nodes based on the expansion strategy you are using the [Token Generating Tool](#). *You can skip this step if you want the new nodes to automatically pick a token range when joining the cluster.*
- Set the `cassandra.yaml` for the new nodes.
- Set the [initial_token](#) according to your token calculations (or leave it unset if you want the new nodes to automatically pick a token range when joining the cluster).
- Start Cassandra on each new node. Allow **two** minutes between node initializations. You can monitor the startup and data streaming process using [nodetool netstats](#).
- After the new nodes are fully bootstrapped, assign the new `initial_token` property value to the nodes that required new tokens, and then run `nodetool move new_token`, one node at a time.
- After all nodes have their new tokens assigned, run [nodetool cleanup](#) one node at a time for each node. Wait for cleanup to complete before doing the next node. This step removes the keys that no longer belong to the previously existing nodes.

Note: Cleanup may be safely postponed for low-usage hours.

Adding a Data Center to a Cluster

Before starting this procedure, please read the guidelines in [Adding Capacity to an Existing Cluster](#) above.

1. Ensure that you are using [NetworkTopologyStrategy](#) for all of your keyspaces.
2. For each new node, edit the configuration properties in the `cassandra.yaml` file:
 - Set `auto_bootstrap` to `False`.
 - Set the `initial_token`. Be sure to offset the tokens in the new data center, see [Generating tokens](#) on page 111.
 - Set the `cluster_name`.
 - Set any other non-default settings.
 - Set the [seed lists](#). Every node in the cluster must have the same list of seeds and include at least one node from each data center. Typically one to three seeds are used per data center.
3. Update either the `cassandra-rackdc.properties` (`GossipingPropertyFileSnitch`) or `cassandra-topology.properties` (`PropertyFileSnitch`) on all servers to include the new nodes. You do not need to restart.
4. Ensure that your client does not auto-detect the new nodes so that they aren't contacted by the client until explicitly directed. For example in Hector, set

```
hostConfig.setAutoDiscoverHosts(false);
```

5. If using a QUORUM [consistency level](#) for reads or writes, check the `LOCAL_QUORUM` or `EACH_QUORUM` consistency level to make sure that the level meets the requirements for multiple data centers.
6. [Start the new nodes](#).
7. After all nodes are running in the cluster:
 - a. Change the [replication factor](#) for your keyspace for the expanded cluster.
 - b. Run [nodetool rebuild](#) on each node in the new data center.

Replacing a Dead Node

1. Confirm that the node is dead using the [nodetool ring](#) command on any live node in the cluster.

The `nodetool ring` command shows a *Down* status for the token value of the dead node:

```
$ nodetool ring -h localhost
```

Address	DC	Rack	Status	State	Load	Owns	Token
10.46.123.11	datacenter1	rack1	Up	Normal	179.58 KB	16.67%	0
10.46.123.12	datacenter1	rack1	Down	Normal	315.21 KB	16.67%	28356863910078205288614550619314017621
10.46.123.13	datacenter1	rack1	Up	Normal	267.71 KB	16.67%	56713727820156410577229101238628035242
10.46.123.14	datacenter1	rack1	Up	Normal	315.21 KB	16.67%	85070591730234615865843651857942052863
10.46.123.15	datacenter1	rack1	Up	Normal	292.36 KB	16.67%	113427455640312821154458202477256070485
10.46.123.16	datacenter1	rack1	Up	Normal	300.02 KB	16.67%	141784319550391026443072753096570088106

2. [Install Cassandra](#) on the replacement node.
3. Remove any preexisting Cassandra data on the replacement node:

```
$ sudo rm -rf /var/lib/cassandra/*
```

4. Set `auto_bootstrap: true`. (If `auto_bootstrap` is not in the `cassandra.yaml` file, it automatically defaults to `true`.)
5. Set the `initial_token` in the `cassandra.yaml` file to the value of the dead node's token -1. Using the value from the above graphic, this is `28356863910078205288614550619314017621-1`:

```
initial_token: 28356863910078205288614550619314017620
```

6. Configure any non-default settings in the node's `cassandra.yaml` to match your existing cluster.
7. [Start the new node](#).

8. After the new node has finished bootstrapping, check that it is marked up using the `nodetool ring` command.
9. Run `nodetool repair` on each keyspace to ensure the node is fully consistent. For example:

```
$ nodetool repair -h 10.46.123.12 keyspace_name
```

10. Remove the dead node.

Backing up and restoring data

About snapshots

Cassandra backs up data by taking a snapshot of all on-disk data files (SSTable files) stored in the data directory. You can take a snapshot of all keyspaces, a single keyspace, or a single table while the system is online.

Using a parallel ssh tool (such as `pssh`), you can snapshot an entire cluster. This provides an *eventually consistent* backup. Although no one node is guaranteed to be consistent with its replica nodes at the time a snapshot is taken, a restored snapshot resumes consistency using Cassandra's built-in consistency mechanisms.

After a system-wide snapshot is performed, you can enable incremental backups on each node to backup data that has changed since the last snapshot: each time an SSTable is flushed, a hard link is copied into a `/backups` subdirectory of the data directory (provided JNA is enabled).

Note: If JNA is enabled, snapshots are performed by hard links. If not enabled, I/O activity increases as the files are copied from one location to another, which significantly reduces efficiency.

Taking a snapshot

Snapshots are taken per node using the `nodetool snapshot` command. To take a global snapshot, run the `nodetool snapshot` command using a parallel ssh utility, such as `pssh`.

A snapshot first flushes all in-memory writes to disk, then makes a hard link of the SSTable files for each keyspace. You must have enough free disk space on the node to accommodate making snapshots of your data files. A single snapshot requires little disk space. However, snapshots can cause your disk usage to grow more quickly over time because a snapshot prevents old obsolete data files from being deleted. After the snapshot is complete, you can move the backup files to another location if needed, or you can leave them in place.

Note: Cassandra can only restore data from a snapshot when the table schema exists. It is recommended that you also backup the schema.

Procedure

Run the `nodetool snapshot` command, specifying the hostname, JMX port, and keyspace. For example:

```
$ nodetool -h localhost -p 7199 snapshot mykeyspace
```

Results

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

For example:

- Package installations: `/var/lib/cassandra/data/mykeyspace/users-081a1500136111e482d09318a3b15cc2/snapshots/1406227071618/mykeyspace-users-ka-1-Data.db`
- Tarball installations: `install_location/data/data/mykeyspace/users-081a1500136111e482d09318a3b15cc2/snapshots/1406227071618/mykeyspace-users-ka-1-Data.db`

Deleting snapshot files

When taking a snapshot, previous snapshot files are not automatically deleted. You should remove old snapshots that are no longer needed.

The `nodetool clearsnapshot` command removes all existing snapshot files from the snapshot directory of each keyspace. You should make it part of your back-up process to clear old snapshots before taking a new one.

Procedure

To delete all snapshots for a node, run the `nodetool clearsnapshot` command. For example:

```
$ nodetool -h localhost -p 7199 clearsnapshot
```

To delete snapshots on all nodes at once, run the `nodetool clearsnapshot` command using a parallel ssh utility.

Enabling incremental backups

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

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

Procedure

Edit the `cassandra.yaml` configuration file on each node in the cluster and change the value of `incremental_backups` to `true`.

Restoring from a snapshot

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

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

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

- Method 1
 1. Restore the snapshot, as described below.
 2. Recreate the schema.
- Method 2
 1. Recreate the schema.
 2. Restore the snapshot, as described below.
 3. Run [nodetool refresh](#).

Procedure

You can restore a snapshot in several ways:

- Use the [sstableloader](#) tool.
- Copy the snapshot SSTable directory (see [Taking a snapshot](#)) to the `data/keyspace/table_name-UUID` directory and then call the JMX method `loadNewSSTables()` in the column family MBean for each column family through JConsole. You can use [nodetool refresh](#) instead of the `loadNewSSTables()` call.

The location of the data directory depends on the type of installation:

- Package installations: `/var/lib/cassandra/data`
- Tarball installations: `install_location/data/data`
- Use the Node Restart Method described below.

Node restart method

If restoring a single node, you must first shutdown the node. If restoring an entire cluster, you must shut down all nodes, restore the snapshot data, and then start all nodes again.

Note: Restoring from snapshots and incremental backups temporarily causes intensive CPU and I/O activity on the node being restored.

The location of the `commitlog` directory depends on the type of installation:

Package installations	<code>/var/lib/cassandra/commitlog</code>
Tarball installations	<code>install_location/data/commitlog</code>
Windows installations	

Procedure

1. Shut down the node.
2. Clear all files in the commitlog directory.

This prevents the commitlog replay from putting data back, which would defeat the purpose of restoring data to a particular point in time.

3. Delete all `*.db` files in the `data_directory/keyspace_name/keyspace_name-keyspace_name` directory, but **DO NOT** delete the `snapshots` and `backups` subdirectories.

where `data_directory` is:

- Package installations: `/var/lib/cassandra/data`
- Tarball installations: `install_location/data/data`

4. Locate the most recent snapshot folder in this directory:


```
data_directory/keyspace_name/table_name-UUID/snapshots/snapshot_name
```

5. Copy its contents into this directory:

```
data_directory/keyspace_name/table_name-UUID directory.
```

6. If using incremental backups, copy all contents of this directory:

```
data_directory/keyspace_name/table_name-UUID/backups
```

7. Paste it into this directory:

```
data_directory/keyspace_name/table_name-UUID
```

8. Restart the node.

Restarting causes a temporary burst of I/O activity and consumes a large amount of CPU resources.

9. Run [nodetool repair](#).

Related tasks

[Starting Cassandra as a service](#) on page 119

[Starting Cassandra as a stand-alone process](#) on page 119

[Stopping Cassandra as a service](#) on page 120

[Stopping Cassandra as a stand-alone process](#) on page 120

Related reference

[The nodetool utility](#) on page 160

Related information

[Repairing nodes](#) on page 137

Restoring a snapshot into a new cluster

Suppose you want to copy a snapshot of SSTable data files from a three node Cassandra cluster with vnodes enabled (256 tokens) and recover it on another newly created three node cluster (256 tokens). The token ranges will not match, because the token ranges cannot be exactly the same in the new cluster. You need to specify the tokens for the new cluster that were used in the old cluster.

Note: This procedure assumes you are familiar with [restoring a snapshot](#) and configuring and initializing a cluster. If not, see [Initializing a cluster](#) on page 114.

Procedure

To recover the snapshot on the new cluster:

1. From the old cluster, retrieve the list of tokens associated with each node's IP:

```
$ nodetool ring | grep ip_address_of_node | awk '{print $NF ","}' | xargs
```

2. In the `cassandra.yaml` file for each node in the new cluster, add the list of tokens you obtained in the previous step to the `initial_token` parameter using the same `num_tokens` setting as in the old cluster.
3. Make any other necessary changes in the new cluster's `cassandra.yaml` and property files so that the new nodes match the old cluster settings. Make sure the seed nodes are set for the new cluster.
4. Clear the system table data from each new node:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

This allows the new nodes to use the initial tokens defined in the `cassandra.yaml` when they restart.

5. Start each node using the specified list of token ranges in new cluster's `cassandra.yaml`:

```
initial_token: -9211270970129494930, -9138351317258731895,
-8980763462514965928, ...
```

6. Create schema in the new cluster. All the schema from the old cluster must be reproduced in the new cluster.
7. Stop the node. Using `nodetool refresh` is unsafe because files within the data directory of a running node can be silently overwritten by identically named just-flushed SSTables from memtable flushes or compaction. Copying files into the data directory and restarting the node will not work for the same reason.
8. Restore the SSTable files snapshotted from the old cluster onto the new cluster using the same directories, while noting that the UUID component of target directory names has changed. Without restoration, the new cluster will not have data to read upon restart.
9. Restart the node.

Recovering from a single disk failure using JBOD

How to recover from a single disk failure in a disk array using JBOD (just a bunch of disks).

Node can restart

1. [Stop Cassandra](#) and shut down the node.
2. Replace the failed disk.
3. Start the node and [Cassandra](#).
4. Run `nodetool repair` on the node.

Node cannot restart

If the node cannot restart, it is possible the system directory is corrupted. If the node cannot restart after completing these steps, see [Replacing a dead node or dead seed node](#) on page 124.

If using the node uses vnodes:

1. [Stop Cassandra](#) and shut down the node.
2. Replace the failed disk.
3. On a healthy node run the following command:

```
$ nodetool ring | grep ip_address_of_node | awk ' {print $NF ","} ' | xargs
```

4. On the node with the new disk, add the list of tokens from the previous step (separated by commas), under `initial_token` in the `cassandra.yaml` file.
5. Clear each `system` directory for every functioning drive:

Assuming disk1 has failed and the `data_file_directories` setting in the `cassandra.yaml` for each drive is:

```
- /mnt1/cassandra/data
- /mnt2/cassandra/data
- /mnt3/cassandra/data
```

Run the following commands:

```
$ rm -fr /mnt2/cassandra/data/system
$ rm -fr /mnt3/cassandra/data/system
```

6. Start the node and [Cassandra](#).
7. Run `nodetool repair`.
8. After the node is fully integrated into the cluster, it is recommended to return to normal vnode settings:

- `num_tokens`: *number_of_tokens*
- `#initial_token`

If the node uses assigned tokens (single-token architecture):

1. **Stop Cassandra** and shut down the node.
2. Replace the failed disk.
3. Clear each `system` directory for every functioning drive:

Assuming disk1 has failed and the `data_file_directories` setting in the `cassandra.yaml` for each drive is:

```
- /mnt1/cassandra/data
- /mnt2/cassandra/data
- /mnt3/cassandra/data
```

Run the following commands:

```
$ rm -fr /mnt2/cassandra/data/system
$ rm -fr /mnt3/cassandra/data/system
```

4. Start the node and **Cassandra**.
5. Run **nodetool repair** on the node.

Repairing nodes

Over time, data in a replica can become inconsistent with other replicas due to the distributed nature of the database. Node repair makes data on a replica consistent with data on other nodes and is important for every Cassandra cluster. Repair is the process of correcting the inconsistencies so that eventually, all nodes have the same and most up-to-date data.

Repair can occur in the following ways:

- **Hinted Handoff**

During the write path, if a node that should receive data is unavailable, hints are written to the coordinator. When the node comes back online, the coordinator can hand off the hints so that the node can catch up and write the data.

- **Read Repair**

During the read path, a query acquires data from several nodes. The acquired data from each node is checked against each other node. If a node has outdated data, the most recent data is written back to the node.

- **Anti-Entropy Repair**

For maintenance purposes or recovery, manually run anti-entropy repair to rectify inconsistencies on any nodes.

Cassandra settings or Cassandra tools can be used to configure each type of repair. Depending on other conditions of the cluster, when to use each type of repair and how to configure them varies.

Hinted Handoff: repair during write path

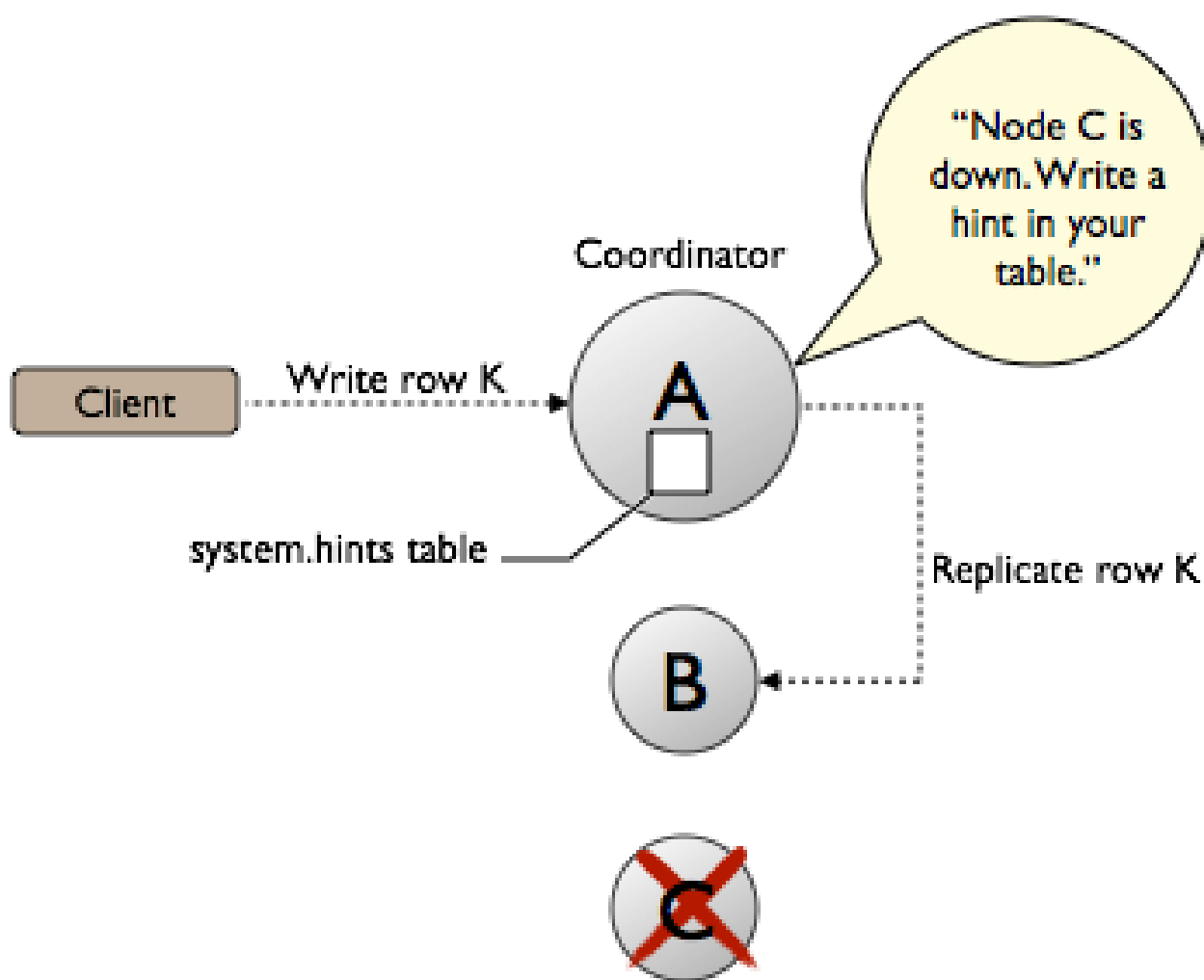
On occasion, a node becomes unresponsive while data is being written. Reasons for unresponsiveness are hardware problems, network issues, or overloaded nodes that experience long garbage collection (GC) pauses. By design, hinted handoff inherently allows Cassandra to continue performing the same number of writes even when the cluster is operating at reduced capacity.

After the failure detector marks a node as down, missed writes are stored by the coordinator for a period of time, if hinted handoff is [enabled](#) in the `cassandra.yaml` file. In Cassandra 3.0 and later, the hint is stored in a local hints directory on each node for improved replay. The hint consists of a target ID for the downed node, a hint ID that is a time UUID for the data, a message ID that identifies the Cassandra version, and the data itself as a blob. Hints are flushed to disk every 10 seconds, reducing the staleness of the hints. Hints are replayed and upon successful replay, the file gets deleted. If the maximum time to store hints has not been exceeded, gossip discovers when a node comes back online. The coordinator uses the hint to write the data to the newly-returned node. If a node is down for longer than [max_hint_window_in_ms](#) (3 hours by default), the coordinator node discards the stored hints.

The coordinator also checks every ten minutes for hints corresponding to writes that timed out during an outage too brief for the failure detector to notice through gossip. If a replica node is overloaded or unavailable, and the failure detector has not yet marked the node as down, then expect most or all writes to that node to fail after the timeout triggered by [write_request_timeout_in_ms](#), (10 seconds by default). The coordinator returns a `TimeoutException` exception, and the write will fail but a hint will be stored. If several nodes experience brief outages simultaneously, substantial memory pressure can build up on the coordinator. The coordinator tracks how many hints it is currently writing, and if the number increases too much, the coordinator refuses writes and throws the `withOverloadedException` exception.

The consistency level of a write request affects whether hints are written and a write request subsequently fails. If the cluster consists of two nodes, A and B, with a replication factor of 1, each row is stored on only one node. Suppose node A is down when a row K is written to it with a consistency level of ONE. In this case, the consistency level specified cannot be met, and since node A is the coordinator, it cannot store a hint. Node B cannot write the data, because it has not received the data as the coordinator nor has a hint been stored. The coordinator checks the number of replicas that are up and will not attempt to write the hint if the consistency level specified by a client cannot be met. A hinted handoff failure occurs and will return a `UnavailableException` exception. The write request fails and the hint is not written.

In general, the recommendation is to have enough nodes in the cluster and a replication factor sufficient to avoid write request failures. For example, consider a cluster consisting of three nodes, A, B, and C, with a replication factor of 2. When a row K is written to the coordinator (node A in this case), even if node C is down, the consistency level of ONE or QUORUM can be met. Why? Both nodes A and B will receive the data, so the consistency level requirement is met. A hint is stored for node C and written when node C comes up.



Immsvi

For applications that want Cassandra to accept writes when all the normal replicas are down and consistency level ONE cannot be satisfied, Cassandra provides consistency level ANY. ANY guarantees that the write is durable and readable after an appropriate replica target becomes available and receives the hint replay.

Nodes that die might have stored undelivered hints, because any node can be a coordinator. The data on the dead node will be stale after a long outage as well. If a node has been down for an extended period of time, a [manual repair](#) should be run.

At first glance, it seems that hinted handoff eliminates the need for manual repair, but this is not true because hardware failure is inevitable and has the following ramifications:

- Loss of the historical data necessary to tell the rest of the cluster exactly what data is missing.
- Loss of hints-not-yet-replayed from requests that the failed node coordinated.

When removing a node from the cluster by decommissioning the node or by using the [nodetool removemode](#) command, Cassandra automatically removes hints targeting the node that no longer exists. Cassandra also removes hints for dropped tables.

For more explanation about hint storage, see [Modern hinted handoff](#).

Read Repair: repair during read path

Read repair is an important component of keeping data consistent in a Cassandra cluster, because every time a read request occurs, it provides an opportunity for consistency improvement. As a background process, read repair generally puts little strain on the cluster.

When data is read to satisfy a query and return a result, all replicas are queried for the data needed. The first replica node receives a direct read request and supplies the full data. The other nodes contacted receive a digest request and return a digest, or hash of the data. A digest is requested because generally the hash is smaller than the data itself. A comparison of the digests allows the coordinator to return the most up-to-date data to the query. If the digests are the same for enough replicas to meet the consistency level, the data is returned. If the consistency level of the read query is `ALL`, the comparison must be completed before the results are returned; otherwise for all lower consistency levels, it is done in the background.

The coordinator compares the digests, and if a mismatch is discovered, a request for the full data is sent to the mismatched nodes. The most current data found in a full data comparison is used to reconcile any inconsistent data on other replicas.

Read repair can be configured per table for non-`QUORUM` consistency levels (using `read_repair_chance`) and is enabled by default.

The compaction strategy [DateTieredCompactionStrategy](#) precludes using read repair, because of the way timestamps are checked for DTCS compaction. In this case, you must set `read_repair_chance` to zero. For other compaction strategies, read repair should be enabled with a `read_repair_chance` value of 0.2 being typical.

Manual repair: Anti-entropy repair

Anti-entropy node repairs are important for every Cassandra cluster. Frequent data deletions and downed nodes are common causes of data inconsistency. Use anti-entropy repair for routine maintenance and when a cluster needs fixing by running the [nodetool repair](#) command.

How does anti-entropy repair work?

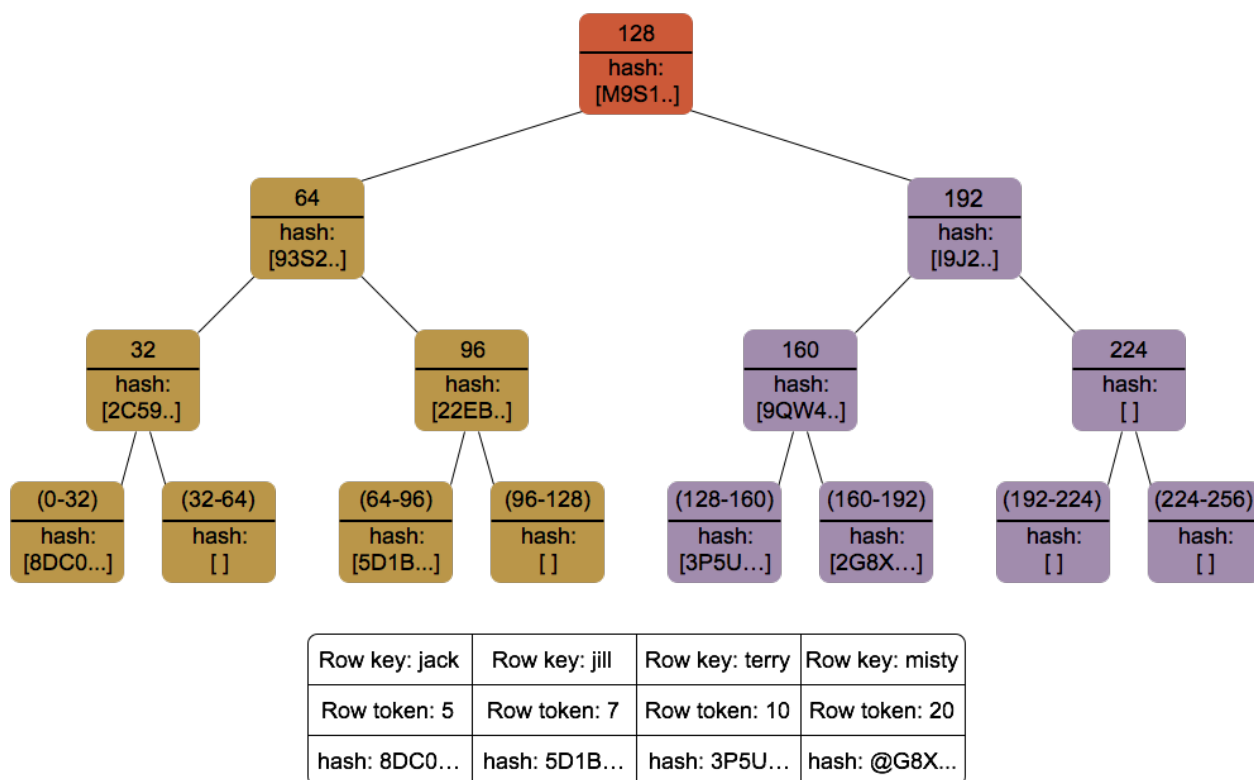
Cassandra accomplishes anti-entropy repair using Merkle trees, similar to Dynamo and Riak. Anti-entropy is a process of comparing the data of all replicas and updating each replica to the newest version. Cassandra has two phases to the process:

1. Merkle trees are built for each replica
2. The Merkle trees are compared to discover where differences are present

The `nodetool repair` command can be run on either a specified node or on all nodes if a node is not specified. The node that initiates the repair becomes the coordinator node for the operation. To build the Merkle trees, the coordinator node determines peer nodes with matching ranges of data. A major, or validation, compaction is triggered on the peer nodes. The validation compaction reads and generates a hash for every row in the stored column families, adds the result to a Merkle tree, and returns the tree to the initiating node. Merkle trees use hashes of the data, because in general, hashes will be smaller than the data itself. [Repair in Cassandra](#) discusses this process in more detail.

Merkle trees are binary hash trees where leaves are hashes of the individual key values. In the case of Cassandra, a leaf will hold the hash of a row value. Parent nodes higher in the tree are hashes of their respective children. Because higher nodes in the Merkle tree represent data further down the tree, each branch of the tree can be checked independently without requiring nodes to download the entire data set. The structure of Merkle trees allows the anti-entropy repair Cassandra employs to use a compact tree version with a depth of 15 ($2^{15} = 32K$ leaf nodes). For example, a node containing a million partitions with one damaged partition, about 30 partitions are streamed, which is the number that fall into each of the leaves of the tree. Two consequences of building smaller Merkle trees are reducing memory usage to store

the trees and minimizing the amount of data required to transfer a Merkle tree to another node during the comparison process.



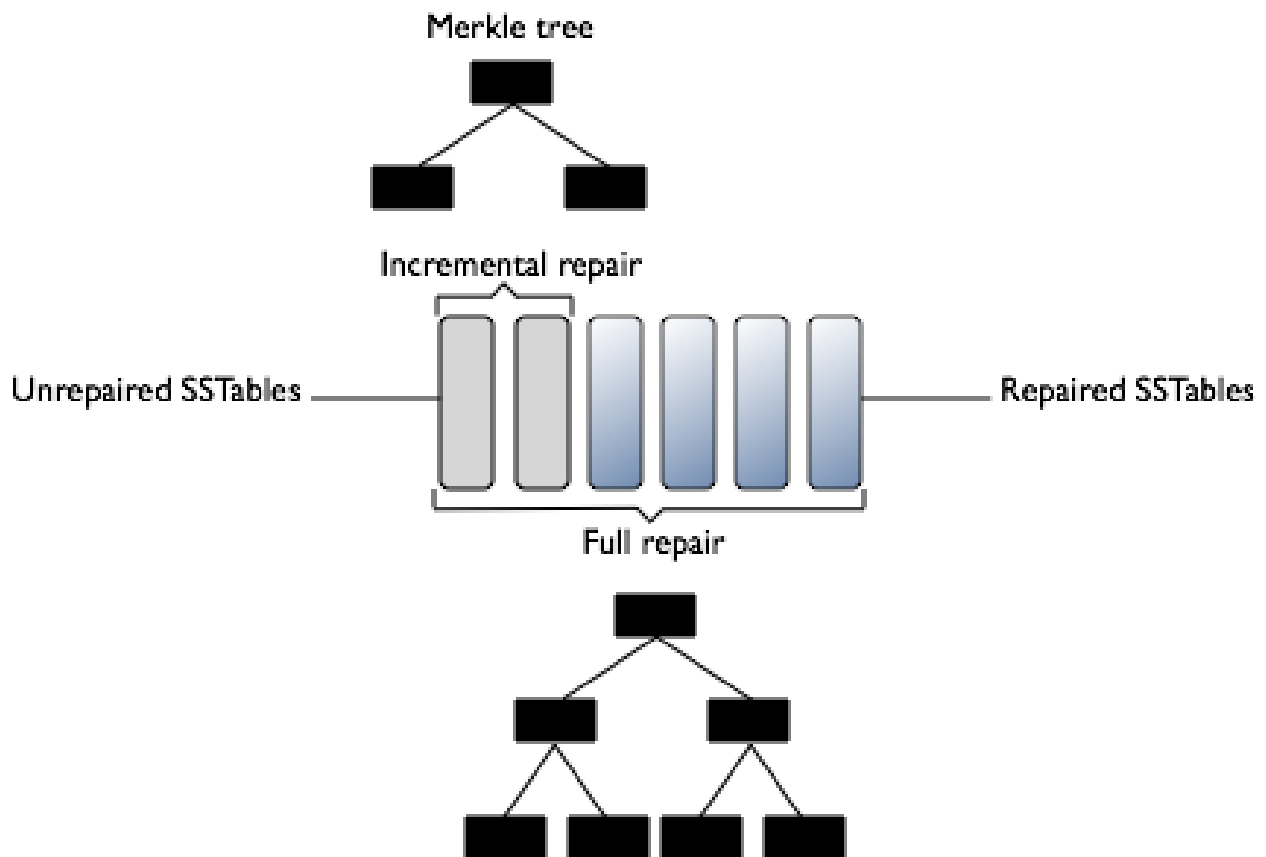
After the initiating node receives the Merkle trees from the participating peer nodes, the initiating node compares every tree to every other tree. If a difference is detected, the differing nodes exchange data for the conflicting range(s), and the new data is written to SSTables. The comparison begins with the top node of the Merkle tree. If no difference is detected, the process proceeds to the left child node and compares and then the right child node. When a node is found to differ, inconsistent data exists for the range that pertains to that node. All data that corresponds to the leaves below that Merkle tree node will be replaced with new data.

Merkle tree building is quite resource intensive, stressing disk I/O and using memory. Some of the options discussed here help lessen the impact on the cluster performance.

Full vs Incremental repair

The process described above represents what occurs for a full repair of a node's data. All SSTables for that node are compared and repaired, if necessary. In Cassandra 2.1 and later, incremental repair is also available. An incremental repair makes already repaired data persistent, and only calculates a Merkle tree for unrepaired SSTables. New metadata was introduced that keeps track of the repaired status of an SSTable and identify repaired and unrepaired data.

Merkle Trees of an Incremental Versus a Full Repair



Reducing the size of the Merkle tree improves the performance of the incremental repair process, assuming repairs are run frequently. Incremental repairs work like full repairs, with an initiating node requesting Merkle trees from peer nodes with the same unrepaired data, and then comparing the Merkle trees to discover mismatches. Once the data has been reconciled and new SSTables built, the initiating node issues an anti-compaction command. Anti-compaction is the process of segregating repaired and unrepaired ranges into separate SSTables, unless the SSTable fits entirely within the repaired range. In the latter case, the SSTable metadata is updated to reflect its repaired status.

Anti-compaction is handled differently, depending on the compaction strategy assigned to the data.

- Size-tiered compaction (STCS) splits repaired and unrepaired data into separate pools for separate compactions. A major compaction generates two SSTables, one for each pool of data.
- Leveled compaction (LCS) performs size-tiered compaction on unrepaired data. After repair completes, Cassandra moves data from the set of unrepaired SSTables to L0.
- Date-tiered (DTCS) splits repaired and unrepaired data into separate pools for separate compactions. A major compaction generates two SSTables, one for each pool of data. DTCS compaction should not use incremental repair.

Full repair is the default in Cassandra 2.1 and earlier. Incremental repair is the default for Cassandra 2.2 and later. In Cassandra 2.2 and later, when a full repair is run, SSTables are marked as repaired and anti-compacted.

Parallel vs Sequential

Sequential repair takes action on one node after another. Parallel repair will repair all nodes with the same replica data at the same time.

Sequential repair takes a snapshot of each replica. Snapshots are hardlinks to existing SSTables. They are immutable and require almost no disk space. The snapshots are live until the repair is completed and then they are removed. The coordinator node constructs the Merkle trees for one replica after the other, and finally repairs each replica one by one from the snapshots. For example, if you have RF=3 and A, B and C represents three replicas, this command takes a snapshot of each replica immediately and then sequentially repairs each replica from the snapshots (A<->B, A<->C, B<->C).

A parallel repair does the repair of nodes A, B, and C all at once. During this type of repair, the [dynamic snitch](#) maintains performance for your application using a replica in the snapshot that is not undergoing repair.

Snapshots are hardlinks to existing SSTables. Snapshots are immutable and require almost no disk space. Repair requires intensive disk I/O because validation compaction occurs during Merkle tree construction. For any given replica set, only one replica at a time performs the validation compaction.

Sequential repair is the default in Cassandra 2.1 and earlier. Parallel repair is the default for Cassandra 2.2 and later.

Note: Sequential and incremental do not work together in Cassandra 2.1.

Partitioner range (`-pr`)

In a Cassandra cluster, a particular range of data will be stored on multiple nodes. If you consider running `nodetool repair` on each node, the same range of data will be repaired several times based on the replication factor used in the keyspace. The partitioner range option will only repair a particular range of data once, rather than repeating the repair operation needlessly. This will decrease the strain on network resources, although Merkle trees still must be built for each replica.

Note: If you use this option, you must run `nodetool repair -pr` on EVERY node in the cluster to repair all data. Otherwise, some ranges of data will not be repaired.

It is recommended that the partitioner range option be used for routine maintenance. If the tool is used on a downed node, however, do not use this option.

Local (`-local`, `--in-local-dc`) vs Data center (`-dc`, `--in-dc`) vs Cluster-wide

Using `nodetool repair` across data centers versus within a local data center requires some careful consideration. Run repair on a node, and using `-local` or `--in-local-dc`, only nodes within the same center as the node running `nodetool repair` will be repaired. Otherwise, all nodes with a replica, whether in the same data center or not, will be repaired. Network traffic between data centers can increase tremendously and cause cluster issues. Using the `-dc` or `--in-dc` options, a specific data center can be specified to run repair. In this case, nodes in other data centers may hold replicas and would be repaired, but repair will be initiated only on nodes within the data center. This option can decrease network traffic while repairing more nodes than the local options.

The `-pr` option becomes especially important to use across multiple data centers, as the number of replicas can grow quite cumbersome. If two data centers, DC1 and DC2, are being repaired, and each has a replication factor of 3, each range repaired will need to build Merkle tables for 6 nodes. This number increases linearly for additional data centers.

Note: The `-local` option cannot be used with `-pr` unless a data center's nodes have all the data for all ranges.

The `-local` option should only be done with full repair and not incremental.

Cassandra 2.2+ includes a `-dcpair` or `--dc-parallel` option to repair data centers in parallel. For multiple data centers, the recommendation is to run in parallel.

Endpoint range vs Subrange repair (`-st`, `--start-token`, `-et` `--end-token`)

A repair operation runs on all partition ranges, or endpoint range, stored on a node unless the subrange option is used. Subrange repair specifies a start token and a end token so that only some partition ranges are repaired. Generally, subrange repair is not recommended because it requires the use of generated token ranges. However, if a known partition has an error, that partition range can be targeted for repair. A problem known as overstreaming, which can tie up resources by sending repairs to a range over and over, can be relieved with the subrange repair option.

Subrange repair involves more than just the `nodetool repair` command. A Java `describe_splits` call to ask for a split containing 32k partitions can be iterated throughout the entire range incrementally or in parallel to eliminate the overstreaming behavior. Once the tokens are generated for the split, they are passed to `nodetool repair -st <start_token> -et <end_token>`. The `-local` option can be used to repair only within a local data center to reduce cross data center transfer.

When to run anti-entropy repair

When to run anti-entropy repair is dependent on the characteristics of a Cassandra cluster. General guidelines are presented here, and should be tailored to each particular case.

When is repaired needed?

Run repair in these situations:

- To routinely maintain node health.

Note: Even if deletions never occur, schedule regular repairs. Setting a column to null is a delete.
- To recover a node after a failure while bringing it back into the cluster.
- To update data on a node containing data that is not read frequently, and therefore does not get read repair.
- To update data on a node that has been down.
- To recover missing data or corrupted SSTables. A non-incremental repair is required.

Guidelines for running routine node repair include:

- Run incremental repair daily, run full repairs weekly to monthly. Monthly is generally sufficient, but run more frequently if warranted.

Important: Full repair is useful for maintaining data integrity, even if deletions never occur.

- Use the parallel and partitioner range options, unless precluded by the scope of the repair.
- Run a full repair to eliminate anti-compaction. Anti-compaction is the process of splitting an SSTable into two SSTables, one with repaired data and one with non-repaired data. This has [compaction strategy implications](#).

Note: [Migrating to incremental repairs](#) is recommended if you use leveled compaction.

- Run repair frequently enough that every node is repaired before reaching the time specified in the `gc_grace_seconds` setting. Deleted data is properly handled in the cluster if this requirement is met.
- Schedule routine node repair to minimize cluster disruption.
 - If possible, schedule repair operation for low-usage hours.
 - If possible, schedule repair operations on single nodes at a time.
- Increase the time value setting of `gc_grace_seconds` if data is seldom deleted or overwritten. For these tables, changing the setting will:
 - Minimizes impact to disk space.
 - Allow longer interval between repair operations.
- Mitigate heavy disk usage by configuring `nodetool` compaction throttling options ([setcompactionthroughput](#) and [setcompactionthreshold](#)) before running a repair.

Guidelines for running repair on a downed node:

- Do not use partitioner range, `-pr`.

Migrating to incremental repairs

Migrating to incremental repairs

Migrating to incremental repairs by using the `sstablerepairedset` utility is recommended only under the following conditions:

- You are doing an incremental repair for the first time.
- You are using the leveled compaction strategy.

Full, sequential repairs are the default because until the first incremental repair, Cassandra does not know the repaired state of SSTables. After an incremental repair, anticompaaction marks SSTables as repaired or not. If you use the leveled compaction strategy and perform an incremental repair for the first time, Cassandra performs size-tiering on all SSTables because the repair/unrepaired status is unknown. This operation can take a long time. To save time, migrate to incremental repair one node at a time. The migration procedure, covered in the next section, uses utilities in the `tools/bin` directory of installations other than RHEL and Debian:

- `sstablemetadata` for checking the repaired or unrepaired status of an SStable
- `sstablerepairedset` for manually marking an SStable as repaired

The syntax of these commands is:

```
$ sstablemetadata <sstable filenames>
```

```
$ sstablerepairedset [--is-repaired | --is-unrepaired] [-f <sstable-list> | <sstables>]
```

In Cassandra 2.1.1, `sstablerepairedset` can take as arguments a list of SSTables on the command line or a [file of SSTables with a "-f" flag](#).

Note: In [RHEL](#) and [Debian](#) installations, you must install the tools packages.

This example shows how to use `sstablerepairedset` to clear the repaired state of an SStable, rendering the SStable unrepaired. As mentioned above, because until the first incremental repair, Cassandra does not know the repaired state of SSTables, this example shows how to use `sstablerepairedset` to clear the repaired state of an SStable, rendering the SStable unrepaired.

1. Stop the node.
2. Run this command:

```
$ sstablerepairedset --is-unrepaired -f list_of_sstable_names.txt
```

3. Restart the node.

All data is changed to an unrepaired state.

Procedure for migrating to incremental repairs

To migrate to incremental repair, one node at a time:

1. Disable compaction on the node using `nodetool disableautocompaction`.
2. Run the default full, sequential repair.
3. Stop the node.
4. Use the tool `sstablerepairedset` to mark all the SSTables that were created before you disabled compaction.
5. Restart cassandra

SSTables remain in a repaired state after running a full, but not a partition range, repair if you make no changes to the SSTables.

Monitoring Cassandra

Monitoring a Cassandra cluster

Understanding the performance characteristics of a Cassandra cluster is critical to diagnosing issues and planning capacity.

Cassandra exposes a number of statistics and management operations via Java Management Extensions (

JMX). JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX.

During normal operation, Cassandra outputs information and statistics that you can monitor using JMX-compliant tools, such as:

- The Cassandra [nodetool utility](#)
- JConsole

Using the same tools, you can perform certain administrative commands and operations such as flushing caches or doing a node repair.

Monitoring using the nodetool utility

The [nodetool utility](#) is a command-line interface for monitoring Cassandra and performing routine database operations. Included in the Cassandra distribution, nodetool and is typically run directly from an operational Cassandra node.

The nodetool utility supports the most important JMX metrics and operations, and includes other useful commands for Cassandra administration, such as the [proxyhistogram command](#). This example shows the output from nodetool proxyhistograms after running 4,500 insert statements and 45,000 select statements on a three [ccm](#) node-cluster on a local computer.

```
$ nodetool proxyhistograms

proxy histograms
Percentile      Read Latency      Write Latency      Range Latency
                  (micros)          (micros)           (micros)
50%              1502.50           375.00             446.00
75%              1714.75           420.00             498.00
95%              31210.25          507.00             800.20
98%              36365.00          577.36             948.40
99%              36365.00          740.60            1024.39
Min               616.00           230.00             311.00
Max              36365.00          55726.00          59247.00
```

For a summary of the ring and its current state of general health, use the [status command](#). For example:

```
$ nodetool proxyhistograms

Note: Ownership information does not include topology; for complete
information, specify a keyspace
Datacenter: datacenter1
=====
```

```

Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load           Tokens     Owns    Host ID
    Rack
UN  127.0.0.1    47.66 KB      1           33.3%   aaalb7c1-6049-4a08-
ad3e-3697a0e30e10 rack1
UN  127.0.0.2    47.67 KB      1           33.3%   1848c369-4306-4874-
afdf-5c1e95b8732e rack1
UN  127.0.0.3    47.67 KB      1           33.3%   49578bf1-728f-438d-b1c1-
d8dd644b6f7f rack1

```

The `nodetool` utility provides commands for viewing detailed metrics for tables, server metrics, and compaction statistics:

- `nodetool tablestats` displays statistics for each table and keyspace.
- `nodetool cfhistograms` provides statistics about a table, including read/write latency, row size, column count, and number of SSTables.
- `nodetool netstats` provides statistics about network operations and connections.
- `nodetool tpstats` provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool.

Monitoring using JConsole

JConsole is a JMX-compliant tool for monitoring Java applications such as Cassandra. It is included with Sun JDK 5.0 and higher. JConsole consumes the JMX metrics and operations exposed by Cassandra and displays them in a well-organized GUI. For each node monitored, JConsole provides these six separate tab views:

- Overview
Displays overview information about the Java VM and monitored values.
- Memory
Displays information about memory use.
- Threads
Displays information about thread use.
- Classes
Displays information about class loading.
- VM Summary
Displays information about the Java Virtual Machine (VM).
- Mbeans
Displays information about MBeans.

The Overview and Memory tabs contain information that is very useful for Cassandra developers. The Memory tab allows you to compare heap and non-heap memory usage, and provides a control to immediately perform Java garbage collection.

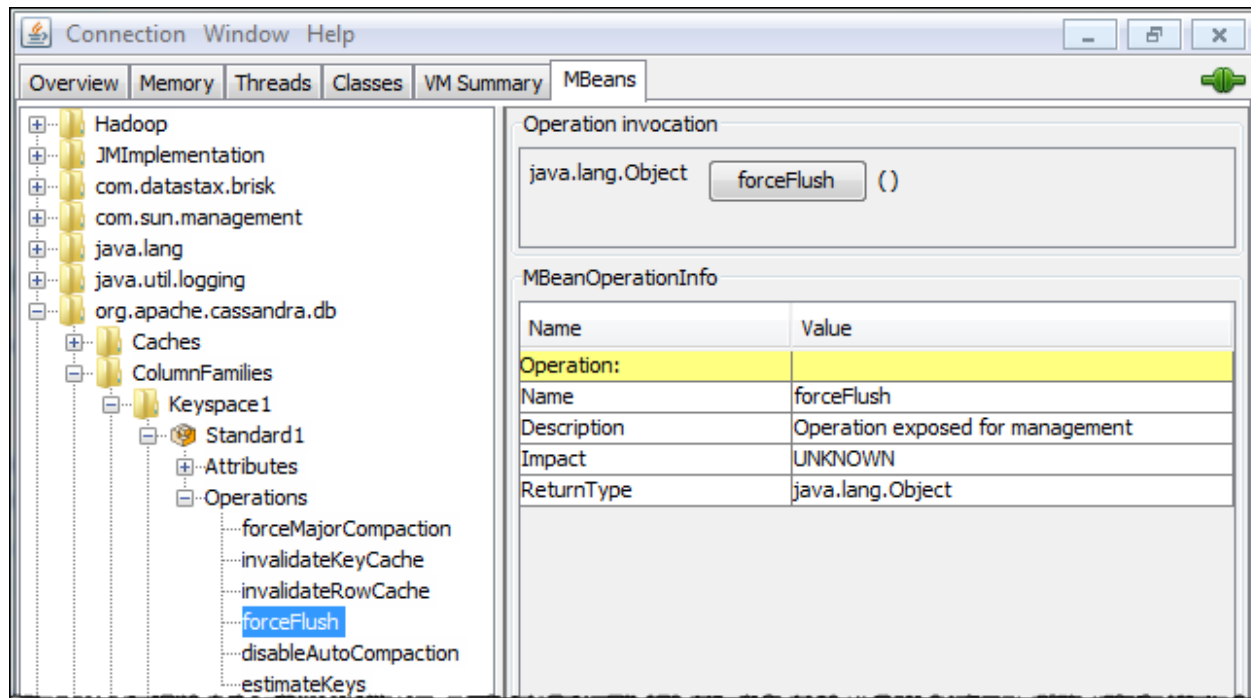
For specific Cassandra metrics and operations, the most important area of JConsole is the MBeans tab. This tab lists the following Cassandra MBeans:

- `org.apache.cassandra.auth`
Includes permissions cache.
- `org.apache.cassandra.db`
Includes caching, table metrics, and compaction.
- `org.apache.cassandra.internal`
Internal server operations such as gossip, hinted handoff, and Memtable values.

Operations

- `org.apache.cassandra.metrics`
Includes metrics on CQL, clients, keyspaces, read repair, storage, and threadpools and other topics.
- `org.apache.cassandra.net`
Inter-node communication including `FailureDetector`, `MessagingService` and `StreamingManager`.
- `org.apache.cassandra.request`
Tasks related to read, write, and replication operations.
- `org.apache.cassandra.service`
Includes `GCInspector`.

When you select an MBean in the tree, its MBeanInfo and MBean Descriptor are displayed on the right, and any attributes, operations or notifications appear in the tree below it. For example, selecting and expanding the `org.apache.cassandra.db` MBean to view available actions for a table results in a display like the following:



If you choose to monitor Cassandra using JConsole, keep in mind that JConsole consumes a significant amount of system resources. For this reason, DataStax recommends running JConsole on a remote machine rather than on the same host as a Cassandra node.

The JConsole `CompactionManagerMBean` exposes [compaction metrics](#) that can indicate when you need to add capacity to your cluster.

Compaction metrics

Monitoring compaction performance is an important aspect of knowing when to add capacity to your cluster. The following attributes are exposed through `CompactionManagerMBean`:

Table: Compaction Metrics

Attribute	Description
CompletedTasks	Number of completed compactions since the last start of this Cassandra instance

Attribute	Description
PendingTasks	Number of estimated tasks remaining to perform
ColumnFamilyInProgress	The table currently being compacted. This attribute is null if no compactions are in progress.
BytesTotalInProgress	Total number of data bytes (index and filter are not included) being compacted. This attribute is null if no compactions are in progress.
BytesCompacted	The progress of the current compaction. This attribute is null if no compactions are in progress.

Thread pool and read/write latency statistics

Cassandra maintains distinct thread pools for different stages of execution. Each of the thread pools provide statistics on the number of tasks that are active, pending, and completed. Trends on these pools for increases in the pending tasks column indicate when to add additional capacity. After a baseline is established, configure alarms for any increases above normal in the pending tasks column. Use [nodetool tpstats](#) on the command line to view the thread pool details shown in the following table.

Table: Compaction Metrics

Thread Pool	Description
AE_SERVICE_STAGE	Shows anti-entropy tasks.
CONSISTENCY-MANAGER	Handles the background consistency checks if they were triggered from the client's consistency level.
FLUSH-SORTER-POOL	Sorts flushes that have been submitted.
FLUSH-WRITER-POOL	Writes the sorted flushes.
GOSSIP_STAGE	Activity of the Gossip protocol on the ring.
LB-OPERATIONS	The number of load balancing operations.
LB-TARGET	Used by nodes leaving the ring.
MEMTABLE-POST-FLUSHER	Memtable flushes that are waiting to be written to the commit log.
MESSAGE-STREAMING-POOL	Streaming operations. Usually triggered by bootstrapping or decommissioning nodes.
MIGRATION_STAGE	Tasks resulting from the call of system_* methods in the API that have modified the schema.
MISC_STAGE	
MUTATION_STAGE	API calls that are modifying data.
READ_STAGE	API calls that have read data.
RESPONSE_STAGE	Response tasks from other nodes to message streaming from this node.
STREAM_STAGE	Stream tasks from this node.

Read/Write latency metrics

Cassandra tracks latency (averages and totals) of read, write, and slicing operations at the server level through `StorageProxyMBean`.

Table statistics

For individual tables, ColumnFamilyStoreMBean provides the same general latency attributes as StorageProxyMBean. Unlike StorageProxyMBean, ColumnFamilyStoreMBean has a number of other statistics that are important to monitor for performance trends. The most important of these are:

Table: Compaction Metrics

Attribute	Description
MemtableDataSize	The total size consumed by this table's data (not including metadata).
MemtableColumnsCount	Returns the total number of columns present in the memtable (across all keys).
MemtableSwitchCount	How many times the memtable has been flushed out.
RecentReadLatencyMicros	The average read latency since the last call to this bean.
RecentWriterLatencyMicros	The average write latency since the last call to this bean.
LiveSSTableCount	The number of live SSTables for this table.

The recent read latency and write latency counters are important in making sure operations are happening in a consistent manner. If these counters start to increase after a period of staying flat, you probably need to add capacity to the cluster.

You can set a threshold and monitor LiveSSTableCount to ensure that the number of SSTables for a given table does not become too great.

Tuning Java resources

Consider tuning Java resources in the event of a performance degradation or high memory consumption.

The `cassandra-env.sh` control environment settings, such as Java Virtual Machine (JVM) configuration settings, for Cassandra.

Heap sizing options

If you decide to change the Java heap sizing, set both `MAX_HEAP_SIZE` and `HEAP_NEWSIZE` together in `cassandra-env.sh`.

- `MAX_HEAP_SIZE`
Sets the maximum heap size for the JVM. The same value is also used for the minimum heap size. This allows the heap to be locked in memory at process start to keep it from being swapped out by the OS.
- `HEAP_NEWSIZE`
The size of the young generation. The larger this is, the longer GC pause times will be. The shorter it is, the more expensive GC will be (usually). A good guideline is 100 MB per CPU core.

Tuning the Java heap

Because Cassandra spends significant time interacting with the operating system's I/O infrastructure through the JVM, so a well-tuned Java heap size is important. Cassandra's default configuration opens the JVM with a heap size that is based on the total amount of system memory:

System Memory	Heap Size
Less than 2GB	1/2 of system memory

System Memory	Heap Size
2GB to 4GB	1GB
Greater than 4GB	1/4 system memory, but not more than 8GB

Many users new to Cassandra are tempted to turn up Java heap size too high, which consumes the majority of the underlying system's RAM. In most cases, increasing the Java heap size is actually detrimental for these reasons:

- The capability of Java to gracefully handle garbage collection above 8GB quickly diminishes.
- Modern operating systems maintain the OS page cache for frequently accessed data and are very good at keeping this data in memory, but can be prevented from doing its job by an elevated Java heap size.

If you have more than 2GB of system memory, keep the size of the Java heap relatively small to allow more memory for the page cache.

Some Solr users have reported that increasing the stack size improves performance under Tomcat. To increase the stack size, uncomment and modify the default setting in the `cassandra-env.sh` file. Also, decreasing the memtable space to make room for Solr caches can improve performance. Modify the memtable space using the `memtable_total_space_in_mb` property in the `cassandra.yaml` file.

Because MapReduce runs outside the JVM, changes to the JVM do not affect Analytics/Hadoop operations directly.

How Cassandra uses memory

You can typically allocate about 8GB of memory on the heap before garbage collection pause time starts to become a problem. Modern machines have much more memory than that and Cassandra makes use of additional memory as page cache when files on disk are accessed. Allocating more than 8GB of memory on the heap poses a problem due to the amount of Cassandra metadata about data on disk. The Cassandra metadata resides in memory and is proportional to total data. Some of the components [grow proportionally to the size of total memory](#).

In Cassandra 1.2 and later, the Bloom filter and compression offset map that store this metadata reside off-heap, greatly increasing the capacity per node of data that Cassandra can handle efficiently. The partition summary also resides off-heap.

About the off-heap row cache

Cassandra can store cached rows in native memory, outside the Java heap. This results in both a smaller per-row memory footprint and reduced JVM heap requirements, which helps keep the heap size in the sweet spot for JVM garbage collection performance.

Tuning Java garbage collection

In Cassandra 2.2 and later, the default JVM garbage collection is the Concurrent-Mark-Sweep (CMS) garbage collector. The G1 garbage collector can be configured. The G1 garbage collector is more performant than the Concurrent-Sweep-Mark (CMS) garbage collector for a heap size of 4GB or larger. It will become the default garbage collector for Java 9 in the future. G1 scans the regions of the heap that contain the most garbage object first. It also compacts the heap on-the-go, while the CMS garbage collector only compacts during full stop-the-world garbage collection. To configure G1, remove all CMS parameters from `$CASSANDRA_HOME/conf/cassandra-env.sh` and add the following:

```
JVM_OPTS="$env:JVM_OPTS -XX:+UseG1GC"
JVM_OPTS="$env:JVM_OPTS -XX:G1RSetUpdatingPauseTimePercent=5"
```

Cassandra's `GCInspector` class logs information about garbage collection whenever a garbage collection takes longer than 200ms. Garbage collections that occur frequently and take a moderate length of time to complete (such as `ConcurrentMarkSweep` taking a few seconds), indicate that there is a lot of garbage

collection pressure on the JVM. Remedies include adding nodes, lowering cache sizes, or adjusting the JVM options regarding garbage collection.

JMX options

Cassandra exposes a number of statistics and management operations via Java Management Extensions (JMX). JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX. [JConsole](#) and the `nodetool` utility are examples of JMX-compliant management tools.

By default, you can modify the following properties in the `cassandra-env.sh` file to configure JMX to listen on port 7199 without authentication.

- `com.sun.management.jmxremote.port`

The port on which Cassandra listens from JMX connections.

- `com.sun.management.jmxremote.ssl`

Enable/disable SSL for JMX.

- `com.sun.management.jmxremote.authenticate`

Enable/disable remote authentication for JMX.

- `-Djava.rmi.server.hostname`

Sets the interface hostname or IP that JMX should use to connect. Uncomment and set if you are having trouble connecting.

Data caching

Configuring data caches

Cassandra includes integrated caching and distributes cache data around the cluster. When a node goes down, the client can read from another cached replica of the data. The integrated architecture also facilitates troubleshooting because there is no separate caching tier, and cached data matches what is in the database exactly. The integrated cache alleviates the cold start problem by saving the cache to disk periodically. Cassandra reads contents back into the cache and distributes the data when it restarts. The cluster does not start with a cold cache.

In Cassandra 2.1, the saved key cache files include the ID of the table in the file name. A saved key cache file name for the `users` table in the `mykeyspace` keyspace in a Cassandra 2.1 looks something like this:

```
mykeyspace-users.users_name_idx-19bd7f80352c11e4aa6a57448213f97f-KeyCache-  
b.db2046071785672832311.tmp
```

You can configure partial or full caching of each partition by setting the `rows_per_partition` table option. Previously, the caching mechanism put the entire partition in memory. If the partition was larger than the cache size, Cassandra never read the data from the cache. Now, you can specify the number of rows to cache per partition to increase cache hits. You configure the cache using the [CQL caching property](#).

About the partition key cache

The partition key cache is a cache of the [partition index](#) for a Cassandra table. Using the key cache instead of relying on the OS page cache decreases seek times. However, enabling just the key cache results in disk (or OS page cache) activity to actually read the requested data rows.

About the row cache

You can configure the number of rows to cache in a partition. To cache rows, if the row key is not already in the cache, Cassandra reads the first portion of the partition, and puts the data in the cache. If the newly cached data does not include all cells configured by user, Cassandra performs another read. The actual size of the row-cache depends on the workload. You should properly benchmark your application to get "the best" row cache size to configure.

There are two row cache options, the old serializing cache provider and a new off-heap cache (OHC) provider. The new OHC provider has been benchmarked as performing about 15% better than the older option.

Typically, you enable either the partition key or row cache for a table, except archive tables, which are infrequently read. Disable caching entirely for archive tables.

Enabling and configuring caching

Use CQL to enable or disable caching by configuring the caching [table property](#). Set parameters in the `cassandra.yaml` file to configure global caching properties:

- [Partition key cache size](#)
- [Row cache size](#)
- How often Cassandra [saves partition key caches](#) to disk
- How often Cassandra [saves row caches](#) to disk

Set the caching property using [CREATE TABLE](#) or [ALTER TABLE](#). For example, configuring the `row_cache_size_in_mb` determines how much space in memory Cassandra allocates to store rows from the most frequently read partitions of the table.

Procedure

Set the table caching property that configures the partition key cache and the row cache.

```
CREATE TABLE users (
  userid text PRIMARY KEY,
  first_name text,
  last_name text,
)
WITH caching = { 'keys' : 'NONE', 'rows_per_partition' : '120' };
```

Tips for efficient cache use

[Tuning the row cache in Cassandra 2.1](#) describes best practices of using the built-in caching mechanisms and designing an effective data model. Some tips for efficient cache use are:

- Store lower-demand data or data with extremely long partitions in a table with minimal or no caching.
- Deploy a large number of Cassandra nodes under a relatively light load per node.
- Logically separate heavily-read data into discrete tables.

When you query a table, [turn on tracing](#) to check that the table actually gets data from the cache rather than from disk. The first time you read data from a partition, the trace shows this line below the query because the cache has not been populated yet:

```
Row cache miss [ReadStage:41]
```

In subsequent queries for the same partition, look for a line in the trace that looks something like this:

```
Row cache hit [ReadStage:55]
```

This output means the data was found in the cache and no disk read occurred. Updates invalidate the cache. If you query rows in the cache plus uncached rows, request more rows than the global limit allows, or the query does not grab the beginning of the partition, the trace might include a line that looks something like this:

```
Ignoring row cache as cached value could not satisfy query [ReadStage:89]
```

This output indicates that an insufficient cache caused a disk read. Requesting rows not at the beginning of the partition is a likely cause. Try removing constraints that might cause the query to skip the beginning of the partition, or [place a limit](#) on the query to prevent results from overflowing the cache. To ensure that the query hits the cache, try increasing the cache size limit, or restructure the table to position frequently accessed rows at the head of the partition.

Monitoring and adjusting caching

Make changes to cache options in small, incremental adjustments, then monitor the effects of each change using the [nodetool utility](#). The output of the `nodetool info` command shows the following row cache and key cache metrics, which are configured in the `cassandra.yaml` file:

- Cache size in bytes
- Capacity in bytes
- Number of hits
- Number of requests
- Recent hit rate
- Duration in seconds after which Cassandra saves the key cache.

For example, on start-up, the information from `nodetool info` might look something like this:

```
ID : 387d15ba-7103-491b-9327-1a691dbb504a
Gossip active : true
Thrift active : true
Native Transport active: true
Load : 65.87 KB
Generation No : 1400189757
Uptime (seconds) : 148760
Heap Memory (MB) : 392.82 / 1996.81
Data Center : datacenter1
Rack : rack1
Exceptions : 0
Key Cache : entries 10, size 728 (bytes), capacity 103809024 (bytes),
  93 hits, 102 requests, 0.912 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 51380224 (bytes), 0
  hits, 0 requests, NaN recent hit rate, 7200 save period in seconds
Token : -9223372036854775808
```

In the event of high memory consumption, consider tuning data caches.

Configuring memtable throughput

Configuring memtable throughput can improve write performance. Cassandra flushes memtables to disk, creating SSTables when the [commit log space threshold](#) has been exceeded. Configure the commit log space threshold per node in the `cassandra.yaml`. How you tune memtable thresholds depends on your data and write load. Increase memtable throughput under either of these conditions:

- The write load includes a high volume of updates on a smaller set of data.

- A steady stream of continuous writes occurs. This action leads to more efficient compaction.

Allocating memory for memtables reduces the memory available for caching and other internal Cassandra structures, so tune carefully and in small increments.

Configuring compaction

As discussed in the [Compaction](#) on page 27 topic, the compaction process merges keys, combines columns, evicts tombstones, consolidates SSTables, and creates a new index in the merged SSTable.

In the `cassandra.yaml` file, you configure these global compaction parameters:

- [snapshot_before_compaction](#)
- [concurrent_compactors](#)
- [compaction_throughput_mb_per_sec](#)

The `compaction_throughput_mb_per_sec` parameter is designed for use with large partitions because compaction is throttled to the specified total throughput across the entire system.

Cassandra provides a start-up option for [testing compaction strategies](#) without affecting the production workload.

Using CQL, you configure a compaction strategy:

- `SizeTieredCompactionStrategy` (STCS): The default compaction strategy. This strategy triggers a minor compaction when there are a number of similar sized SSTables on disk as configured by the table subproperty, `min_threshold`. A minor compaction does not involve all the tables in a keyspace. Also see [STCS compaction subproperties](#).
- `DateTieredCompactionStrategy` (DTCS): This strategy is particularly useful for [time series data](#). `DateTieredCompactionStrategy` stores data written within a certain period of time in the same SSTable. For example, Cassandra can store your last hour of data in one SSTable *time window*, and the next 4 hours of data in another time window, and so on. Compactions are triggered when the `min_threshold` (4 by default) for SSTables in those windows is reached. The most common queries for time series workloads retrieve the last hour/day/month of data. Cassandra can limit SSTables returned to those having the relevant data. Also, Cassandra can store data that has been set to expire using TTL in an SSTable with other data scheduled to expire at approximately the same time. Cassandra can then drop the SSTable without doing any compaction. Also see [DTCS compaction subproperties](#) and [DateTieredCompactionStrategy: Compaction for Time Series Data](#).

Note: Disabling read repair when using DTCS is recommended. Use full repair as necessary.

- `LeveledCompactionStrategy` (LCS): The leveled compaction strategy creates SSTables of a fixed, relatively small size (160 MB by default) that are grouped into levels. Within each level, SSTables are guaranteed to be non-overlapping. Each level (L0, L1, L2 and so on) is 10 times as large as the previous. Disk I/O is more uniform and predictable on higher than on lower levels as SSTables are continuously being compacted into progressively larger levels. At each level, row keys are merged into non-overlapping SSTables. This can improve performance for reads, because Cassandra can determine which SSTables in each level to check for the existence of row key data. This compaction strategy is modeled after [Google's leveldb](#) implementation. Also see [LCS compaction subproperties](#).

To configure the compaction strategy property and [CQL compaction subproperties](#), such as the maximum number of SSTables to compact and minimum SSTable size, use [CREATE TABLE](#) or [ALTER TABLE](#).

Procedure

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

```
ALTER TABLE users WITH
```

```
compaction = { 'class' : 'LeveledCompactionStrategy' }
```

2. Change the [compaction strategy property](#) to `SizeTieredCompactionStrategy` and specify the minimum number of SSTables to trigger a compaction using the CQL `min_threshold` attribute.

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

Results

You can monitor the results of your configuration using compaction metrics, see [Compaction metrics](#) on page 148.

Compression

Compression maximizes the storage capacity of Cassandra nodes by reducing the volume of data on disk and disk I/O, particularly for read-dominated workloads. Cassandra quickly finds the location of rows in the SSTable index and decompresses the relevant row chunks.

Write performance is not negatively impacted by compression in Cassandra as it is in traditional databases. In traditional relational databases, writes require overwrites to existing data files on disk. The database has to locate the relevant pages on disk, decompress them, overwrite the relevant data, and finally recompress. In a relational database, compression is an expensive operation in terms of CPU cycles and disk I/O. Because Cassandra SSTable data files are immutable (they are not written to again after they have been flushed to disk), there is no recompression cycle necessary in order to process writes. SSTables are compressed only once when they are written to disk. Writes on compressed tables can show up to a 10 percent performance improvement.

In Cassandra 2.2 and later, the commit log can also be compressed and write performance can be improved 6-12%. For more information, see [Updates to Cassandra's Commit Log in 2.2](#).

When to compress data

Compression is best suited for tables that have many rows and each row has the same columns, or at least as many columns, as other rows. For example, a table containing user data such as username, email, and state, is a good candidate for compression. The greater the similarity of the data across rows, the greater the compression ratio and gain in read performance.

A table that has rows of different sets of columns is not well-suited for compression.

Don't confuse table compression with [compact storage](#) of columns, which is used for backward compatibility of old applications with CQL.

Depending on the data characteristics of the table, compressing its data can result in:

- 2x-4x reduction in data size
- 25-35% performance improvement on reads
- 5-10% performance improvement on writes

After configuring compression on an existing table, subsequently created SSTables are compressed. Existing SSTables on disk are not compressed immediately. Cassandra compresses existing SSTables when the normal Cassandra compaction process occurs. Force existing SSTables to be rewritten and compressed by using [nodetool upgradesstables](#) (Cassandra 1.0.4 or later) or [nodetool scrub](#).

Configuring compression

You configure a table property and subproperties to manage compression. The [CQL table properties documentation](#) describes the types of compression options that are available. Compression is enabled by default.

Procedure

1. Disable compression, using CQL to set the compression parameter `enabled` to `false`.

```
CREATE TABLE DogTypes (
    block_id uuid,
    species text,
    alias text,
    population varint,
    PRIMARY KEY (block_id)
)
WITH compression = { 'enabled' : false };
```

2. Enable compression on an existing table, using `ALTER TABLE` to set the compression algorithm `class` to `LZ4Compressor` (Cassandra 1.2.2 and later), `SnappyCompressor`, or `DeflateCompressor`.

```
CREATE TABLE DogTypes (
    block_id uuid,
    species text,
    alias text,
    population varint,
    PRIMARY KEY (block_id)
)
WITH compression = { 'class' : 'LZ4Compressor' };
```

3. Change compression on an existing table, using `ALTER TABLE` and setting the compression algorithm `class` to `DeflateCompressor`.

```
ALTER TABLE CatTypes
WITH compression = { 'class' : 'DeflateCompressor',
    'chunk_length_in_kb' : 64 }
```

You tune data compression on a per-table basis using CQL to alter a table.

Testing compaction and compression

Write survey mode is a Cassandra startup option for testing new compaction and compression strategies. In write survey mode, you can test out new compaction and compression strategies on that node and benchmark the write performance differences, without affecting the production cluster.

Write survey mode adds a node to a database cluster. The node accepts all write traffic as if it were part of the normal Cassandra cluster, but the node does not officially join the ring.

Also use write survey mode to try out a new Cassandra version. The nodes you add in write survey mode to a cluster must be of the same major release version as other nodes in the cluster. The write survey mode relies on the streaming subsystem that transfers data between nodes in bulk and differs from one major release to another.

If you want to see how read performance is affected by modifications, stop the node, bring it up as a standalone machine, and then benchmark read operations on the node.

Procedure

Start the Cassandra node using the `write_survey` option:

- Package installations: Add the following option to `cassandra-env.sh` file:

```
JVM_OPTS="$JVM_OPTS -Dcassandra.write_survey=true
```

- Tarball installations: Start Cassandra with this option:

```
$ cd install_location
$ sudo bin/cassandra -Dcassandra.write_survey=true
```

The location of the `cassandra-topology.properties` file depends on the type of installation:

Package installations	<code>/etc/cassandra/cassandra-topology.properties</code>
Tarball installations	<code>install_location/conf/cassandra-topology.properties</code>
Windows installations	

Tuning Bloom filters

Cassandra uses Bloom filters to determine whether an SSTable has data for a particular row. Bloom filters are unused for range scans, but are used for index scans. Bloom filters are probabilistic sets that allow you to trade memory for accuracy. This means that higher Bloom filter attribute settings `bloom_filter_fp_chance` use less memory, but will result in more disk I/O if the SSTables are highly fragmented. Bloom filter settings range from 0 to 1.0 (disabled). The default value of `bloom_filter_fp_chance` depends on the [compaction strategy](#). The `LeveledCompactionStrategy` uses a higher default value (0.1) than the `SizeTieredCompactionStrategy` or `DateTieredCompactionStrategy`, which have a default of 0.01. Memory savings are nonlinear; going from 0.01 to 0.1 saves about one third of the memory. SSTables using LCS contain a relatively smaller ranges of keys than those using STCS, which facilitates efficient exclusion of the SSTables even without a bloom filter; however, adding a small bloom filter helps when there are many levels in LCS.

The settings you choose depend the type of workload. For example, to run an analytics application that heavily scans a particular table, you would want to inhibit the Bloom filter on the table by setting it high.

To view the observed Bloom filters false positive rate and the number of SSTables consulted per read use [tablestats](#) in the `nodetool` utility.

Bloom filters are stored off-heap so you don't need include it when determining the `-Xmx` settings (the maximum memory size that the heap can reach for the JVM).

To change the [bloom filter property](#) on a table, use CQL. For example:

```
ALTER TABLE addamsFamily WITH bloom_filter_fp_chance = 0.1;
```

After updating the value of `bloom_filter_fp_chance` on a table, Bloom filters need to be regenerated in one of these ways:

- [Initiate compaction](#)
- [Upgrade SSTables](#)

You do not have to restart Cassandra after regenerating SSTables.

Moving data to or from other databases

Cassandra offers several solutions for migrating from other databases:

- The [COPY command](#), which mirrors what the PostgreSQL RDBMS uses for file/export import.
- The [Cassandra bulk loader](#) provides the ability to bulk load external data into a cluster.

About the COPY command

You can use COPY in Cassandra's CQL shell to load flat file data into Cassandra (nearly all relational databases have unload utilities that allow table data to be written to OS files) as well as data to be written out to OS files.

ETL Tools

If you need more sophistication applied to a data movement situation (more than just extract-load), then you can use any number of extract-transform-load (ETL) solutions that now support Cassandra. These tools provide excellent transformation routines that allow you to manipulate source data in literally any way you need and then load it into a Cassandra target. They also supply many other features such as visual, point-and-click interfaces, scheduling engines, and more.

Many ETL vendors who support Cassandra supply community editions of their products that are free and able to solve many different use cases. Enterprise editions are also available that supply many other compelling features that serious enterprise data users need.

You can freely download and try ETL tools from Jaspersoft, Pentaho, and Talend that all work with Cassandra.

Purging gossip state on a node

Gossip information is persisted locally by each node to use immediately on node restart without having to wait for gossip communications.

Procedure

In the unlikely event you need to correct a problem in the gossip state:

1. Using MX4J or [JConsole](#), connect to the node's JMX port and then use the JMX method `Gossiper.unsafeAssassinateEndpoints(ip_address)` to assassinate the problem node.

This takes a few seconds to complete so wait for confirmation that the node is deleted.

2. If the JMX method above doesn't solve the problem, stop your client application from sending writes to the cluster.
3. Take the entire cluster offline:
 - a) [Drain](#) each node.

```
$ nodetool options drain
```

- b) Stop each node:

- Package installations:

```
$ sudo service cassandra stop
```

- Tarball installations:

```
$ sudo service cassandra stop
```

4. Clear the data from the `peers` directory:

```
$ sudo rm -r /var/lib/cassandra/data/system/peers/*
```

CAUTION:

Use caution when performing this step. The action clears internal system data from Cassandra and may cause application outage without careful execution and validation of the results. To validate the results, run the following query individually on each node to confirm that all of the nodes are able to see all other nodes.

```
select * from system.peers;
```

5. Clear the gossip state when the node starts:

- For tarball installations, you can use a command line option or edit the `cassandra-env.sh`. To use the command line:

```
$ install_location/bin/cassandra -Dcassandra.load_ring_state=false
```

- For package installations or if you are not using the command line option [above](#), add the following line to the `cassandra-env.sh` file:

```
JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"
```

- Package installations: `/usr/share/cassandra/cassandra-env.sh`
- Tarball installations: `install_location/conf/cassandra-env.sh`

6. Bring the cluster online one node at a time, starting with the seed nodes.

- Package installations:

```
$ sudo service cassandra start
```

- Tarball installations:

```
$ cd install_location
$ bin/cassandra
```

What to do next

Remove the line you added in the `cassandra-env.sh` file.

Cassandra tools

The nodetool utility

The nodetool utility is a command line interface for managing a cluster.

Command formats

```
$ nodetool [options] command [args]
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- The repair and rebuild commands can affect multiple nodes in the cluster.
- Most `nodetool` commands operate on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Example

```
$ nodetool -u cassandra -pw cassandra describering demo_keyspace
```

Getting nodetool help

nodetool help

Provides a listing of `nodetool` commands.

nodetool help *command name*

Provides help on a specific command. For example:

```
$ nodetool help upgradesstables
```

nodetool assassinate

Forcefully removes a dead node without re-replicating any data. It is a last resort tool if you cannot successfully use [nodetool removemode](#).

Synopsis

```
$ nodetool [options] assassinate [args]
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool assassinate` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool assassinate` command is a tool of last resort. Only use this tool to remove a node from a cluster when `removenode` is not successful.

Examples

```
$ nodetool -u cassandra -pw cassandra assassinate 192.168.100.2
```

nodetool bootstrap

Monitor and manage a node's bootstrap process.

Synopsis

```
$ nodetool [options] bootstrap [resume]
```

Table: Options

Short	Long	Description
<code>-h</code>	<code>--host</code>	Hostname or IP address
<code>-p</code>	<code>--port</code>	Port number
<code>-pwf</code>	<code>--password-file</code>	Password file path
<code>-pw</code>	<code>--password</code>	Password
<code>-u</code>	<code>--username</code>	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.

- `nodetool bootstrap` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool bootstrap` command can be used to monitor and manage a node's bootstrap process. If no argument is defined, the help information is displayed. If the argument `resume` is used, bootstrap streaming is resumed.

Examples

```
$ nodetool -u cassandra -pw cassandra bootstrap resume
```

nodetool cfhistograms

This tool has been renamed as [tablehistograms](#).

nodetool cfstats

This tool has been renamed as [nodetool tablestats](#).

nodetool cleanup

Cleans up keyspaces and partition keys no longer belonging to a node.

Synopsis

```
$ nodetool <options> cleanup -- <keyspace> (<table> ...)
```

- Options are:
 - (`-h` | `--host`) `<host name>` | `<ip address>`
 - (`-p` | `--port`) `<port number>`
 - (`-pw` | `--password`) `<password >`
 - (`-u` | `--username`) `<user name>`
 - (`-pwf` `<passwordFilePath>` | `--password-file` `<passwordFilePath>`)
- `--` separates an option from an argument that could be mistaken for a option.
- `keyspace` is a keyspace name.
- `table` is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Use this command to remove unwanted data after adding a new node to the cluster. Cassandra does not automatically remove data from nodes that lose part of their partition range to a newly added node. Run `nodetool cleanup` on the source node and on neighboring nodes that shared the same subrange after the new node is up and running. Failure to run this command after adding a node causes Cassandra to include the old data to rebalance the load on that node. Running the `nodetool cleanup` command causes a temporary increase in disk space usage proportional to the size of your largest SSTable. Disk I/O occurs when running this command.

Running this command affects nodes that use a counter column in a table. Cassandra assigns a new counter ID to the node.

Optionally, this command takes a list of table names. If you do not specify a keyspace, this command cleans all keyspaces no longer belonging to a node.

nodetool clearsnapshot

Removes one or more snapshots.

Synopsis

```
$ nodetool <options> clearsnapshot -t <snapshot> -- ( <keyspace> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -t means the following file contains the snapshot.
- snapshot is the name of the snapshot.
- -- separates an option from an argument that could be mistaken for a option.
- keyspace is one or more keyspace names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Deletes snapshots in one or more keyspaces. To remove all snapshots, omit the snapshot name.

nodetool compact

Forces a major compaction on one or more tables.

Synopsis

```
$ nodetool <options> compact <keyspace> ( <table> ... )
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address.
-p	--port	Remote JMX agent port number.
-pw	--password	Password.
-pwf	--password-file	Password file path.
-s	--split-output	Split output of STCS files to 50%-25%-12.5% etc.of the total size.
-u	--username	Remote JMX agent user name.

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the *cassandra-env.sh* file for the host, then you must specify credentials.
- No -s will create one large SSTable for STCS.
- -s will not affect DTCS; it will create one large SSTable.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command starts the [compaction process](#) on tables using SizeTieredCompactionStrategy (STCS), DateTieredCompactionStrategy (DTCS), or Leveled compaction (LCS):

- If you do not specify a keyspace or table, a major compaction is run on all keyspaces and tables.
- If you specify only a keyspace, a major compaction is run on all tables in that keyspace.
- If you specify one or more tables, a major compaction is run on those tables.

Major compactions may behave differently depending which compaction strategy is used for the affected tables:

- Size-tiered compaction (STCS) splits repaired and unrepaired data into separate pools for separate compactions. A major compaction generates two SSTables, one for each pool of data.
- Leveled compaction (LCS) performs size-tiered compaction on unrepaired data. After repair completes, Cassandra moves data from the set of unrepaired SSTables to L0.
- Date-tiered (DTCS) splits repaired and unrepaired data into separate pools for separate compactions. A major compaction generates two SSTables, one for each pool of data.

For more details, see [How is data maintained?](#) and [Configuring compaction](#).

Note: A major compaction can cause considerably more disk I/O than minor compactons.

nodetool compactionhistory

Provides the history of compaction operations.

Synopsis

```
$ nodetool <options> compactionhistory
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Example

The actual output of compaction history is seven columns wide. The first three columns show the id, keyspace name, and table name of the compacted SSTable.

```
$ nodetool compactionhistory
```

```
Compaction History:
id                keyspace_name
columnfamily_name
d06f7080-07a5-11e4-9b36-abc3a0ec9088  system
schema_columnfamilies
d198ae40-07a5-11e4-9b36-abc3a0ec9088  libdata      users
0381bc30-07b0-11e4-9b36-abc3a0ec9088  Keyspace1    Standard1
74eb69b0-0621-11e4-9b36-abc3a0ec9088  system      local
e35dd980-07ae-11e4-9b36-abc3a0ec9088  system
compactions_in_progress
8d5cf160-07ae-11e4-9b36-abc3a0ec9088  system
compactions_in_progress
ba376020-07af-11e4-9b36-abc3a0ec9088  Keyspace1    Standard1
d18cc760-07a5-11e4-9b36-abc3a0ec9088  libdata      libout
64009bf0-07a4-11e4-9b36-abc3a0ec9088  libdata      libout
d04700f0-07a5-11e4-9b36-abc3a0ec9088  system      sstable_activity
c2a97370-07a9-11e4-9b36-abc3a0ec9088  libdata      users
cb928a80-07ae-11e4-9b36-abc3a0ec9088  Keyspace1    Standard1
cd8d1540-079e-11e4-9b36-abc3a0ec9088  system      schema_columns
62ced2b0-07a4-11e4-9b36-abc3a0ec9088  system      schema_keyspaces
d19cccf0-07a5-11e4-9b36-abc3a0ec9088  system
compactions_in_progress
640bbf80-07a4-11e4-9b36-abc3a0ec9088  libdata      users
6cd54e60-07ae-11e4-9b36-abc3a0ec9088  Keyspace1    Standard1
```


c29241f0-07a9-11e4-9b36-abc3a0ec9088	libdata	libout
c2a30ad0-07a9-11e4-9b36-abc3a0ec9088	system	
compactions_in_progress		
e3a6d920-079d-11e4-9b36-abc3a0ec9088	system	schema_keyspaces
62c55cd0-07a4-11e4-9b36-abc3a0ec9088	system	
schema_columnfamilies		
62b07540-07a4-11e4-9b36-abc3a0ec9088	system	schema_columns
cdd038c0-079e-11e4-9b36-abc3a0ec9088	system	schema_keyspaces
b797af00-07af-11e4-9b36-abc3a0ec9088	Keyspace1	Standard1
8c918b10-07ae-11e4-9b36-abc3a0ec9088	Keyspace1	Standard1
377d73f0-07ae-11e4-9b36-abc3a0ec9088	system	
compactions_in_progress		
62b9c410-07a4-11e4-9b36-abc3a0ec9088	system	local
d0566a40-07a5-11e4-9b36-abc3a0ec9088	system	schema_columns
ba637930-07af-11e4-9b36-abc3a0ec9088	system	
compactions_in_progress		
cdbc1480-079e-11e4-9b36-abc3a0ec9088	system	
schema_columnfamilies		
e3456f80-07ae-11e4-9b36-abc3a0ec9088	Keyspace1	Standard1
d086f020-07a5-11e4-9b36-abc3a0ec9088	system	schema_keyspaces
d06118a0-07a5-11e4-9b36-abc3a0ec9088	system	local
cdaafd80-079e-11e4-9b36-abc3a0ec9088	system	local
640fde30-07a4-11e4-9b36-abc3a0ec9088	system	
compactions_in_progress		
37638350-07ae-11e4-9b36-abc3a0ec9088	Keyspace1	Standard1

The four columns to the right of the table name show the timestamp, size of the SSTable before and after compaction, and the number of partitions merged. The notation means {tables:rows}. For example: {1:3, 3:1} means 3 rows were taken from one SSTable (1:3) and 1 row taken from 3 SSTables (3:1) to make the one SSTable in that compaction operation.

compacted_at	bytes_in	bytes_out	rows_merged
1404936947592	8096	7211	{1:3, 3:1}
1404936949540	144	144	{1:1}
1404941328243	1305838191	1305838191	{1:4647111}
1404770149323	5864	5701	{4:1}
1404940844824	573	148	{1:1, 2:2}
1404940700534	576	155	{1:1, 2:2}
1404941205282	766331398	766331398	{1:2727158}
1404936949462	8901649	8901649	{1:9315}
1404936336175	8900821	8900821	{1:9315}
1404936947327	223	108	{1:3, 2:1}
1404938642471	144	144	{1:1}
1404940804904	383020422	383020422	{1:1363062}
1404933936276	4889	4177	{1:4}
1404936334171	441	281	{1:3, 2:1}
1404936949567	379	79	{2:2}
1404936336248	144	144	{1:1}
1404940645958	307520780	307520780	{1:1094380}
1404938642319	8901649	8901649	{1:9315}
1404938642429	416	165	{1:3, 2:1}
1404933543858	692	281	{1:3, 2:1}
1404936334109	7760	7186	{1:3, 2:1}
1404936333972	4860	4724	{1:2, 2:1}
1404933936715	441	281	{1:3, 2:1}
1404941200880	1269180898	1003196133	{1:2623528, 2:946565}
1404940699201	297639696	297639696	{1:1059216}
1404940556463	592	148	{1:2, 2:2}
1404936334033	5760	5680	{2:1}
1404936947428	8413	5316	{1:2, 3:1}
1404941205571	429	42	{2:2}

```

. . . 1404933936584      7994      6789      {1:4}
. . . 1404940844664      306699417    306699417    {1:1091457}
. . . 1404936947746      601      281      {1:3, 3:1}
. . . 1404936947498      5840     5680     {3:1}
. . . 1404933936472      5861     5680     {3:1}
. . . 1404936336275      378      80      {2:2}
. . . 1404940556293      302170540 281000000    {1:924660, 2:75340}

```

nodetool compactionstats

Provide statistics about a compaction.

Synopsis

```
$ nodetool <options> compactionstats -H
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- data center is the name of an arbitrarily chosen data center from which to select sources for streaming.
- H converts bytes to a human readable form: kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). (Cassandra 2.1.1)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The total column shows the total number of uncompressed bytes of SSTables being compacted. The system log lists the names of the SSTables compacted.

Example

```
$ nodetool compactionstats
```

```

pending tasks: 5
      compaction type      keyspace      table      completed
      total      unit      progress
      302170540      Compaction      Keyspace1      Standard1      282310680
      bytes      93.43%
      307520780      Compaction      Keyspace1      Standard1      58457931
      bytes      19.01%
Active compaction remaining time : 0h00m16s

```

nodetool decommission

Deactivates a node by streaming its data to another node.

Synopsis

```
$ nodetool <options> decommission
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Causes a live node to decommission itself, streaming its data to the next node on the ring. Use [netstats](http://wiki.apache.org/cassandra/NodeProbe#Decommission) to monitor the progress, as described on <http://wiki.apache.org/cassandra/NodeProbe#Decommission> and http://wiki.apache.org/cassandra/Operations#Removing_nodes_entirely.

nodetool describcluster

Provide the name, snitch, partitioner and schema version of a cluster

Synopsis

```
$ nodetool <options> describcluster -- <data center>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- data center is the name of an arbitrarily chosen data center from which to select sources for streaming.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Describe cluster is typically used to validate the schema after upgrading. If a schema disagreement occurs, check for and [resolve schema disagreements](#).

Example

```
$ nodetool describecluster
```

```
Cluster Information:
Name: Test Cluster
Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
  65e78f0e-e81e-30d8-a631-a65dff93bf82: [127.0.0.1]
```

If a schema disagreement occurs, the last line of the output includes information about unreachable nodes.

```
$ nodetool describecluster
```

```
Cluster Information:
Name: Production Cluster
Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
  UNREACHABLE: 1176b7ac-8993-395d-85fd-41b89ef49fbb:
[10.202.205.203]
```

nodetool describering

Provides the partition ranges of a keyspace.

Synopsis

```
$ nodetool <options> describering -- <keyspace>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- separates an option from an argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Example

This example shows the sample output of the command on a three-node cluster.

```
$ nodetool describering demo_keyspace
```

```
Schema Version:1b04bd14-0324-3fc8-8bcb-9256d1e15f82
TokenRange:
  TokenRange(start_token:3074457345618258602,
    end_token:-9223372036854775808,
      endpoints:[127.0.0.1, 127.0.0.2, 127.0.0.3],
      rpc_endpoints:[127.0.0.1, 127.0.0.2, 127.0.0.3],
      endpoint_details:[EndpointDetails(host:127.0.0.1,
datacenter:datacenter1, rack:rack1),
      EndpointDetails(host:127.0.0.2, datacenter:datacenter1,
rack:rack1),
      EndpointDetails(host:127.0.0.3, datacenter:datacenter1,
rack:rack1)])
  TokenRange(start_token:-3074457345618258603,
    end_token:3074457345618258602,
      endpoints:[127.0.0.3, 127.0.0.1, 127.0.0.2],
      rpc_endpoints:[127.0.0.3, 127.0.0.1, 127.0.0.2],
      endpoint_details:[EndpointDetails(host:127.0.0.3,
datacenter:datacenter1, rack:rack1),
      EndpointDetails(host:127.0.0.1, datacenter:datacenter1,
rack:rack1),
      EndpointDetails(host:127.0.0.2, datacenter:datacenter1,
rack:rack1)])
  TokenRange(start_token:-9223372036854775808,
    end_token:-3074457345618258603,
      endpoints:[127.0.0.2, 127.0.0.3, 127.0.0.1],
      rpc_endpoints:[127.0.0.2, 127.0.0.3, 127.0.0.1],
      endpoint_details:[EndpointDetails(host:127.0.0.2,
datacenter:datacenter1, rack:rack1),
      EndpointDetails(host:127.0.0.3, datacenter:datacenter1,
rack:rack1),
      EndpointDetails(host:127.0.0.1, datacenter:datacenter1,
rack:rack1)])
```

If a schema disagreement occurs, the last line of the output includes information about unreachable nodes.

```
$ nodetool describecluster
```

```
Cluster Information:
  Name: Production Cluster
    Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
    Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
    Schema versions:
      UNREACHABLE: 1176b7ac-8993-395d-85fd-41b89ef49fbb:
[10.202.205.203]
```

nodetool disableautocompaction

Disables autocompaction for a keyspace and one or more tables.

Synopsis

```
$ nodetool <options> disableautocompaction -- <keyspace> ( <table> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >

- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The keyspace can be followed by one or more tables.

nodetool disablebackup

Disables incremental backup.

Synopsis

```
$ nodetool <options> disablebackup
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool disablebinary

Disables the native transport.

Synopsis

```
$ nodetool <options> disablebinary
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >

- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Disables the binary protocol, also known as the native transport.

nodetool disablegossip

Disables the gossip protocol.

Synopsis

```
$ nodetool <options> disablegossip
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command effectively marks the node as being down.

nodetool disablehandoff

Disables storing of future hints on the current node.

Synopsis

```
$ nodetool <options> disablehandoff
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >

- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool disablehintsfordc

Disable hints for a data center.

Synopsis

```
$ nodetool [options] disablehintsfordc [--] <datacenter>
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the *cassandra-env.sh* file for the host, then you must specify credentials.
- `nodetool disablehintsfordc` operates on a single node in the cluster if -h is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the -h option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using -h.
- [--] can be used to separate command-line options from the list of arguments, when the list might be mistaken for options.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool disablehintsfordc` command is used to turn off hints for a data center. This can be useful if there is a downed data center, but hints should continue on other data centers. Another common case is during data center failover, when hints will put unnecessary pressure on the data center.

Examples

```
$ nodetool -u cassandra -pw cassandra disablehintsfordc DC2
```

nodetool disablethrift

Disables the Thrift server.

Synopsis

```
$ nodetool [options] disablethrift [args]
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool disablethrift` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

`nodetool disablethrift` will disable thrift on a node preventing the node from acting as a coordinator. The node can still be a replica for a different coordinator and data read at consistency level ONE could be stale. To cause a node to ignore read requests from other coordinators, `nodetool disablegossip` would also need to be run. However, if both commands are run, the node will not perform repairs, and the node will continue to store stale data. If the goal is to repair the node, set the read operations to a

consistency level of QUORUM or higher while you run repair. An alternative approach is to delete the node's data and restart the Cassandra process.

Note that the `nodetool` commands using the `-h` option will not work remotely on a disabled node until `nodetool enablethrift` and `nodetool enablegossip` are run locally on the disabled node.

Examples

```
$ nodetool -u cassandra -pw cassandra disablethrift 192.168.100.1
```

nodetool drain

Drains the node.

Synopsis

```
$ nodetool <options> drain
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Flushes all memtables from the node to SSTables on disk. Cassandra stops listening for connections from the client and other nodes. You need to restart Cassandra after running `nodetool drain`. You typically use this command before upgrading a node to a new version of Cassandra. To simply flush memtables to disk, use `nodetool flush`.

nodetool enableautocompaction

Enables autocompaction for a keyspace and one or more tables.

Synopsis

```
$ nodetool <options> enableautocompaction -- <keyspace> ( <table> ... )
```

- Options are:
 - (`-h` | `--host`) <host name> | <ip address>
 - (`-p` | `--port`) <port number>
 - (`-pw` | `--password`) <password >
 - (`-u` | `--username`) <user name>
 - (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)

- -- separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is the name of one or more keyspaces, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The keyspace can be followed by one or more tables. Enables compaction for the named keyspace or the current keyspace, and one or more named tables, or all tables.

nodetool enablebackup

Enables incremental backup.

Synopsis

```
$ nodetool <options> enablebackup
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool enablebinary

Re-enables native transport.

Synopsis

```
$ nodetool <options> enablebinary
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Re-enables the binary protocol, also known as native transport.

nodetool enablegossip

Re-enables gossip.

Synopsis

```
$ nodetool <options> enablegossip
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool enablehandoff

Re-enables the storing of future hints on the current node.

Synopsis

```
$ nodetool <options> enablehandoff
```

- options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- <dc-name>,<dc-name> means enable hinted handoff only for these data centers

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool enablehintsfordc

Enable hints for a data center.

Synopsis

```
$ nodetool [options] enablehintsfordc [--] <datacenter>
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the *cassandra-env.sh* file for the host, then you must specify credentials.
- `nodetool enablehintsfordc` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.
- `--` can be used to separate command-line options from the list of arguments, when the list might be mistaken for options.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool enablehintsfordc` command is used to turn on hints for a data center. The `cassandra.yaml` file has a parameter, `hinted_handoff_disabled_datacenters` that will blacklist data centers on startup. If a data center can be enabled later with `nodetool enablehintsfordc`.

Examples

```
$ nodetool -u cassandra -pw cassandra enablehintsfordc DC2
```

nodetool enablethrift

Re-enables the Thrift server.

Synopsis

```
$ nodetool <options> enablethrift
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool flush

Flushes one or more tables from the memtable.

Synopsis

```
$ nodetool <options> flush -- <keyspace> ( <table> ... )
```

- Options are:
 - (`-h` | `--host`) <host name> | <ip address>
 - (`-p` | `--port`) <port number>
 - (`-pw` | `--password`) <password >
 - (`-u` | `--username`) <user name>
 - (`-pwf` <passwordFilePath | `--password-file` <passwordFilePath>)
- `--` separates an option and argument that could be mistaken for a option.
- `keyspace` is the name of a keyspace.
- `table` is the name of one or more tables, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

You can specify a keyspace followed by one or more tables that you want to flush from the memtable to SSTables on disk.

nodetool gcstats

Print garbage collection (GC) statistics.

Synopsis

```
$ nodetool [options] gcstats
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the *cassandra-env.sh* file for the host, then you must specify credentials.
- `nodetool gcstats` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool gcstats` command will print garbage collection statistics that returns values based on all the garbage collection that has run since the last time `nodetool gcstats` was run. Statistics identify the interval time, some GC elapsed time measures, the disk space reclaimed (in MB), number of garbage collections that took place, and direct memory bytes.

Examples

```
$ nodetool -u cassandra -pw cassandra gcstats
```

nodetool getcompactionthreshold

Provides the minimum and maximum compaction thresholds in megabytes for a table.

Synopsis

```
$ nodetool <options> getcompactionthreshold -- <keyspace> <table>
```

- Options are:
 - (`-h` | `--host`) <host name> | <ip address>
 - (`-p` | `--port`) <port number>
 - (`-pw` | `--password`) <password >
 - (`-u` | `--username`) <user name>
 - (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)
- `--` separates an option and argument that could be mistaken for a option.
- `keyspace` is the name of a keyspace.
- `table` is the name of a table.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool getcompactionthroughput

Print the throughput cap (in MB/s) for compaction in the system.

Synopsis

```
$ nodetool [options] getcompactionthroughput
```

Table: Options

Short	Long	Description
<code>-h</code>	<code>--host</code>	Hostname or IP address
<code>-p</code>	<code>--port</code>	Port number
<code>-pwf</code>	<code>--password-file</code>	Password file path

Short	Long	Description
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool getcompactionthroughput` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool getcompactionthroughput` command prints the current compaction throughput.

Examples

```
$ nodetool -u cassandra -pw cassandra getcompactionthroughput
```

nodetool getendpoints

Provides the IP addresses or names of replicas that own the partition key.

Synopsis

```
$ nodetool <options> getendpoints -- <keyspace> <table> key
```

- Options are:
 - (`-h` | `--host`) <host name> | <ip address>
 - (`-p` | `--port`) <port number>
 - (`-pw` | `--password`) <password >
 - (`-u` | `--username`) <user name>
 - (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)
- `--` separates an option and argument that could be mistaken for a option.
- `keyspace` is a keyspace name.
- `table` is a table name.
- `key` is the partition key of the end points you want to get.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Example

For example, which nodes own partition key_1, key_2, and key_3?

Note: The partitioner returns a token for the key. Cassandra will return an endpoint whether or not data exists on the identified node for that token.

```
$ nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_1
```

```
127.0.0.2
```

```
$ nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_2
```

```
127.0.0.2
```

```
$ nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_2
```

```
127.0.0.1
```

nodetool getlogginglevels

Get the runtime logging levels.

Synopsis

```
$ nodetool <options> getlogginglevels
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool getsstables

Provides the SSTables that own the partition key.

Synopsis

```
$ nodetool <options> getsstables -- <keyspace> <table> key
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.
- key is the partition key of the SSTables.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool getstreamthroughput

Provides the megabytes per second throughput limit for streaming in the system.

Synopsis

```
$ nodetool <options> getstreamthroughput
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool gettraceprobability

Get the current trace probability.

Synopsis

```
$ nodetool <options> gettraceprobability
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- value is a probability between 0 and 1.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides the current trace probability. To set the trace probability, see [nodetool settraceprobability](#).

nodetool gossipinfo

Provides the gossip information for the cluster.

Synopsis

```
$ nodetool <options> gossipinfo
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool help

Provides nodetool command help.

Synopsis

```
$ nodetool help <command>
```

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Using this command without the The help command provides a synopsis and brief description of each nodetool command.

Examples

Using nodetool help lists all commands and usage information. For example, `nodetool help netstats` provides the following information.

```
NAME
    nodetool netstats - Print network information on provided host
    (connecting node by default)

SYNOPSIS
    nodetool [(-h <host> | --host <host>)] [(-p <port> | --port <port>)]
    [(-pw <password> | --password <password>)]
    [(-u <username> | --username <username>)] netstats

OPTIONS
    -h <host>, --host <host>
        Node hostname or ip address

    -p <port>, --port <port>
        Remote jmx agent port number

    -pw <password>, --password <password>
        Remote jmx agent password

    -u <username>, --username <username>
        Remote jmx agent username
```

nodetool info

Provides node information, such as load and uptime.

Synopsis

```
$ nodetool <options> info ( -T | --tokens )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -T or --tokens means provide all token information.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides node information including the token and on disk storage (load) information, times started (generation), uptime in seconds, and heap memory usage.

nodetool invalidatecountercache

Invalidates the counter cache, and resets the global counter cache parameter, counter_cache_keys_to_save, to the default (not set), which saves all keys..

Synopsis

```
$ nodetool [options] invalidatecountercache
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the cassandra-env.sh file for the host, then you must specify credentials.
- `nodetool invalidatecountercache` operates on a single node in the cluster if -h is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the -h option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using -h.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool invalidatecountercache` command will invalidate the counter cache, and the system will start saving all counter keys.

Examples

```
$ nodetool -u cassandra -pw cassandra invalidatecountercache
```

nodetool invalidatekeycache

Resets the global key cache parameter to the default, which saves all keys.

Synopsis

```
$ nodetool <options> invalidatekeycache
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

By default the `key_cache_keys_to_save` is disabled in the `cassandra.yaml`. This command resets the parameter to the default.

nodetool invalidaterowcache

Resets the global key cache parameter, `row_cache_keys_to_save`, to the default (not set), which saves all keys.

Synopsis

```
$ nodetool <options> invalidaterowcache
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool join

Causes the node to join the ring.

Synopsis

```
$ nodetool <options> join
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Causes the node to join the ring, assuming the node was initially *not* started in the ring using the - [Djoin_ring=false](#) cassandra utility option. The joining node should be properly configured with the desired options for seed list, initial token, and auto-bootstrapping.

nodetool listsnapshots

Lists snapshot names, size on disk, and true size.

Synopsis

```
$ nodetool <options> listsnapshots
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Available in Cassandra 2.1 and later.

Example

```
Snapshot Details:
Snapshot Name    Keyspace    Column Family    True Size    Size on Disk
1387304478196    Keyspace1    Standard1        0 bytes      308.66 MB
1387304417755    Keyspace1    Standard1        0 bytes      107.21 MB
1387305820866    Keyspace1    Standard2        0 bytes      41.69 MB
                  Keyspace1    Standard1        0 bytes      308.66 MB
```

nodetool move

Moves the node on the token ring to a new token.

Synopsis

```
$ nodetool <options> move -- <new token>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- new token is a number in the range -2^{63} to $+2^{63}-1$.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Escape negative tokens using \ . For example: move \-123. This command moves a node from one token value to another. This command is generally used to shift tokens slightly.

nodetool netstats

Provides network information about the host.

Synopsis

```
$ nodetool <options> netstats -H
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password>
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- H converts bytes to a human readable form: kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). (Cassandra 2.1.1)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The default host is the connected host if the user does not include a host name or IP address in the command. The output includes the following information:

- JVM settings
- Mode

The operational mode of the node: JOINING, LEAVING, NORMAL, DECOMMISSIONED, CLIENT
- Read repair statistics
- Attempted

The number of successfully completed [read repair operations](#)
- Mismatch (blocking)

The number of read repair operations since server restart that blocked a query.
- Mismatch (background)

The number of read repair operations since server restart performed in the background.
- Pool name

Information about client read and write requests by thread pool.
- Active, pending, and completed number of commands and responses

Example

Get the network information for a node 10.171.147.128:

```
$ nodetool -h 10.171.147.128 netstats
```

The output is:

```
Mode: NORMAL
```

```

Not sending any streams.
Read Repair Statistics:
Attempted: 0
Mismatch (Blocking): 0
Mismatch (Background): 0
Pool Name           Active   Pending   Completed
Commands            n/a      0         1156
Responses           n/a      0         2750

```

nodetool pausehandoff

Pauses the hints delivery process

Synopsis

```
$ nodetool <options> pausehandoff
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool proxyhistograms

Provides a histogram of network statistics.

Synopsis

```
$ nodetool <options> proxyhistograms
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The output of this command shows the full request latency recorded by the coordinator. The output includes the percentile rank of read and write latency values for inter-node communication. Typically, you use the command to see if requests encounter a slow node.

Examples

This example shows the output from `nodetool proxyhistograms` after running 4,500 insert statements and 45,000 select statements on a three [ccm](#) node-cluster on a local computer.

```
$ nodetool proxyhistograms
```

```
proxy histograms
Percentile      Read Latency      Write Latency      Range Latency
                (micros)        (micros)           (micros)
50%             1502.50         375.00             446.00
75%             1714.75         420.00             498.00
95%            31210.25         507.00             800.20
98%            36365.00         577.36             948.40
99%            36365.00         740.60            1024.39
Min              616.00         230.00             311.00
Max            36365.00        55726.00          59247.00
```

nodetool rangekeysample

Provides the sampled keys held across all keyspaces.

Synopsis

```
$ nodetool <options> rangekeysample
```

Options are:

- (`-h` | `--host`) `<host name>` | `<ip address>`
- (`-p` | `--port`) `<port number>`
- (`-pw` | `--password`) `<password >`
- (`-u` | `--username`) `<user name>`
- (`-pwf` `<passwordFilePath>` | `--password-file` `<passwordFilePath>`)

Synopsis Legend

- Angle brackets (`< >`) mean not literal, a variable
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

nodetool rebuild

Rebuilds data by streaming from other nodes.

Synopsis

```
$ nodetool <options> rebuild -- <src-dc-name>
```

- Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- src-dc-name is the name of the data center from which to select sources for streaming. You can pick any data center.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command operates on multiple nodes in a cluster. Similar to bootstrap. Rebuild (like bootstrap) only streams data from a single source replica per range. Use this command to [bring up a new data center](#) in an existing cluster.

nodetool rebuild_index

Performs a full rebuild of the index for a table

Synopsis

```
$ nodetool <options> rebuild_index -- ( <keyspace>.<table>.<indexName> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.
- indexName is an optional list of index names separated by a space.

The keyspace and table name followed by a list of index names. For example: Standard3.IdxName
Standard3.IdxName1

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Fully rebuilds one or more indexes for a table.

nodetool refresh

Loads newly placed SSTables onto the system without a restart.

Synopsis

```
$ nodetool <options> refresh -- <keyspace> <table>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool reloadtriggers

Reloads trigger classes.

Synopsis

```
$ nodetool <options> reloadtriggers
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Available in Cassandra 2.1 and later.

nodetool removenode

Provides the status of current node removal, forces completion of pending removal, or removes the identified node.

Synopsis

```
$ nodetool <options> removenode -- <status> | <force> | <ID>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- status provides status information.
- force forces completion of the pending removal.
- ID is the host ID, in UUID format.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command removes a node, shows the status of a removal operation, or forces the completion of a pending removal. When the node is down and `nodetool decommission` cannot be used, use `nodetool removenode`. Run this command only on nodes that are down. If the cluster does not use `vnodes`, before running the `nodetool removenode` command, [adjust the tokens](#).

Examples

Determine the UUID of the node to remove by running `nodetool status`. Use the UUID of the node that is down to remove the node.

```
$ nodetool status
```

```
Datacenter: DC1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address                Load          Tokens   Owns (effective)  Host ID
   Rack
UN  192.168.2.101         112.82 KB    256      31.7%              420129fc-0d84-42b0-be41-ef7dd3a8ad06 RAC1
DN  192.168.2.103         91.11 KB     256      33.9%              d0844a21-3698-4883-ab66-9e2fd5150edd RAC1
```

```
UN 192.168.2.102 124.42 KB 256 32.6% 8d5ed9f4-7764-4dbd-
bad8-43fddce94b7c RAC1
```

```
$ nodetool removemode d0844a21-3698-4883-ab66-9e2fd5150edd
```

View the status of the operation to remove the node:

```
$ nodetool removemode status
```

```
RemovalStatus: No token removals in process.
```

Confirm that the node has been removed.

```
$ nodetool removemode status
```

```
Datacenter: DC1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address          Load          Tokens   Owns (effective)  Host ID
--
UN 192.168.2.101    112.82 KB    256      37.7%             420129fc-0d84-42b0-
be41-ef7dd3a8ad06  RAC1
UN 192.168.2.102    124.42 KB    256      38.3%             8d5ed9f4-7764-4dbd-
bad8-43fddce94b7c  RAC1
```

nodetool repair

Repairs one or more tables. See [Manual repair: Anti-entropy repair](#) for more information.

Synopsis

```
$ nodetool [options] repair [args] [--] <keyspace> <tables>
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- keyspace is the keyspace name. The default is all.

- `table` is one or more table names, separated by a space. The default is all.
- For tarball installations, execute the command from the `install_location/bin` directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool repair` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Table: Arguments

Short	Long	Description
<code>-dc <dc></code>	<code>--in-dc <dc></code>	Repair specified data center
<code>-dcpar</code>	<code>--dc-parallel</code>	Repair data centers in parallel
<code>-et <end_token></code>	<code>--end-token <end_token></code>	Token at which repair range ends
<code>-full</code>	<code>--full</code>	Do full repair
<code>-j <job_threads></code>	<code>--job-threads <job_threads></code>	Number of threads to run repair jobs. Default: 1 Maximum: 4
<code>-local</code>	<code>--in-local-dc</code>	Repair only nodes in same data center
<code>-pr</code>	<code>--partitioner-range</code>	Repair only first range returned by partitioner
<code>-seq</code>	<code>--sequential</code>	Sequential repair
<code>-st <start_token></code>	<code>--start-token <start_token></code>	Token at which repair range starts
<code>-tr</code>	<code>--trace</code>	Trace the repair. Traces are logged to <code>system_traces.events</code> .

- `-dc`, or `--in-dc`, followed by `dc_name`, or means restrict repair to nodes in the named data center, which must be the local data center.
- `-dcpar`, or `--dc-parallel`, means repair data centers in parallel.
- `-et`, or `--end-token`, followed by the UUID of a token means stop repairing a range of nodes after repairing this token. Use `-hosts` to specify neighbor nodes.
- `-inc`, or `--incremental` means do an incremental repair.
- `-local`, or `--in-local-dc`, means repair nodes in the same data center only.
- `-par`, or `--parallel`, means do a parallel repair.
- `-pr`, or `--partitioner-range`, means repair only the first range returned by the partitioner.
- `-st`, or `--start-token`, followed by the UUID of a token means start repairing a range of nodes at this token.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (...) means repeatable

- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

Performing an anti-entropy node repair on a [regular basis](#) is important, especially when frequently deleting data. The `nodetool repair` command repairs one or more nodes in a cluster, and includes an option to restrict repair to a set of nodes. Anti-entropy node repair performs the following tasks:

- Ensures that all data on a replica is consistent.
- Repairs inconsistencies on a node that has been down.

Full repair is the default in Cassandra 2.1 and earlier. Incremental repair is the default for Cassandra 2.2 and later. In Cassandra 2.2 and later, when a full repair is run, SSTables are marked as repaired and anti-compacted. Sequential repair is the default in Cassandra 2.1 and earlier. Parallel repair is the default for Cassandra 2.2 and later.

Using options

You can use options to do these other types of repair:

- Sequential or Parallel
- Full or incremental

Use the `-hosts` option to list the good nodes to use for repairing the bad nodes. Use `-h` to name the bad nodes.

Use the `-full` option for a full repair if required. By default, an incremental repair eliminates the need for constant Merkle tree construction by persisting already repaired data and calculating only the Merkle trees for SSTables that have not been repaired. The repair process is likely more performant than the other types of repair even as datasets grow, assuming you run repairs frequently. Before doing an incremental repair for the first time, perform [migration steps](#) first if necessary for tables created before Cassandra 2.2.

Use the `-dcpair` option to repair data centers in parallel. Unlike [sequential repair](#), parallel repair constructs the Merkle tables for all data centers at the same time. Therefore, no snapshots are required (or generated). Use parallel repair to complete the repair quickly or when you have operational downtime that allows the resources to be completely consumed during the repair.

Performing [partitioner range repairs](#) by using the `-pr` option is generally not recommended.

Example

All `nodetool repair` arguments are optional. The following examples show the following types of repair:

- A sequential repair of all keyspaces on the current node
- A partitioner range repair of the bad partition on current node using the good partitions on 10.2.2.20 or 10.2.2.21
- A start-point-to-end-point repair of all nodes between two nodes on the ring

```
$ nodetool repair -seq
$ nodetool repair -pr -hosts 10.2.2.20 10.2.2.21
$ nodetool -st a9fa31c7-f3c0-44d1-b8e7-a26228867840c -et f5bb146c-d
b51-475ca44f-9facf2f1ad6e
```

To restrict the repair to the local data center, use the `-dc` option followed by the name of the data center. Issue the command from a node in the data center you want to repair. Issuing the command from a data center other than the named one returns an error. Do not use `-pr` with this option to repair only a local data center.

```
$ nodetool repair -dc DC1
```

```
[2014-07-24 21:59:55,326] Nothing to repair for keyspace 'system'
[2014-07-24 21:59:55,617] Starting repair command #2, repairing 490 ranges
    for keyspace system_traces (seq=true, full=true)
[2014-07-24 22:23:14,299] Repair session 323b9490-137e-11e4-88e3-
c972e09793ca
    for range (820981369067266915,822627736366088177] finished
[2014-07-24 22:23:14,320] Repair session 38496a61-137e-11e4-88e3-
c972e09793ca
    for range (2506042417712465541,2515941262699962473] finished
. . .
```

An inspection of the system.log shows repair taking place only on IP addresses in DC1.

```
. . .
INFO [AntiEntropyStage:1] 2014-07-24 22:23:10,708 RepairSession.java:171
- [repair #16499ef0-1381-11e4-88e3-c972e09793ca] Received merkle tree
  for sessions from /192.168.2.101
INFO [RepairJobTask:1] 2014-07-24 22:23:10,740 RepairJob.java:145
- [repair #16499ef0-1381-11e4-88e3-c972e09793ca] requesting merkle trees
  for events (to [/192.168.2.103, /192.168.2.101])
. . .
```

nodetool replaybatchlog

Replay batchlog and wait for finish.

Synopsis

```
$ nodetool <options> replaybatchlog
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

`nodetool replaybatchlog` is intended to force a batchlog replay. It also blocks until the batches have been replayed.

nodetool resetlocalschema

Reset the node's local schema and resynchronizes.

Synopsis

```
$ nodetool [options] resetlocalschema [args]
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the *cassandra-env.sh* file for the host, then you must specify credentials.
- `nodetool resetlocalschema` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Normally, this command is used to rectify schema disagreements on different nodes. It can be useful if table schema changes have generated too many tombstones, on the order of 100,000s.

`nodetool resetlocalschema` drops the schema information of the local node and resynchronizes the schema from another node. To drop the schema, the tool truncates all the system schema tables. The node will temporarily lose metadata about the tables on the node, but will rewrite the information from another node. If the node is experiencing problems with too many tombstones, the truncation of the tables will eliminate the tombstones.

This command is useful when you have one node that is out of sync with the cluster. The system schema tables must have another node from which to fetch the tables. It is not useful when all or many of your nodes are in an incorrect state. If there is only one node in the cluster (replication factor of 1) – it does not perform the operation, because another node from which to fetch the tables does not exist. Run the command on the node experiencing difficulty.

nodetool resumehandoff

Resume hints delivery process.

Synopsis

```
$ nodetool <options> resumehandoff
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool ring

Provides node status and information about the ring.

Synopsis

```
$ nodetool <options> ring ( -r | --resolve-ip ) -- <keyspace>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password>
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -r, or --resolve-ip, means to provide node names instead of IP addresses.
- -- separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Displays node status and information about the ring as determined by the node being queried. This information can give you an idea of the load balance and if any nodes are down. If your cluster is not

properly configured, different nodes may show a different ring. Check that the node appears the same way in the ring. If you use virtual nodes (vnodes), use `nodetool status` for succinct output.

- Address
The node's URL.
- DC (data center)
The data center containing the node.
- Rack
The rack or, in the case of Amazon EC2, the availability zone of the node.
- Status - Up or Down
Indicates whether the node is functioning or not.
- State - N (normal), L (leaving), J (joining), M (moving)
The state of the node in relation to the cluster.
- Load - updates every 90 seconds
The amount of file system data under the cassandra data directory after excluding all content in the snapshots subdirectories. Because all SSTable data files are included, any data that is not cleaned up, such as TTL-expired cell or tombstoned data) is counted.
- Token
The end of the token range up to and including the value listed. For an explanation of token ranges, see [Data Distribution in the Ring](#).
- Owns
The percentage of the data owned by the node per data center times the replication factor. For example, a node can own 33% of the ring, but show 100% if the replication factor is 3.
- Host ID
The network ID of the node.

nodetool scrub

Rebuild SSTables for one or more Cassandra tables.

Synopsis

```
$ nodetool <options> scrub <keyspace> -- ( -ns | --no-snapshot ) ( -s | --skip-corrupted ) ( <table> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- -ns, or --no-snapshot, triggers a snapshot of the scrubbed table first assuming snapshots are not disabled (the default).
- -s, or --skip-corrupted skips corrupted partitions even when scrubbing counter tables. (default false)
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Rebuilds SSTables on a node for the named tables and snapshots data files before rebuilding as a safety measure. If possible use [nodetool upgradesstables](#). While scrub rebuilds SSTables, it also discards data that it deems broken and creates a snapshot, which you have to remove manually. If the -ns option is specified, snapshot creation is disabled. If scrub can't validate the column value against the column definition's data type, it logs the partition key and skips to the next partition. Skipping corrupted partitions in tables having counter columns results in under-counting. By default the scrub operation stops if you attempt to skip such a partition. To force the scrub to skip the partition and continue scrubbing, re-run `nodetool scrub` using the `--skip-corrupted` option.

nodetool setcachecapacity

Set global key and row cache capacities in megabytes.

Synopsis

```
$ nodetool <options> setcachecapacity -- <key-cache-capacity> <row-cache-capacity>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- key-cache-capacity is the maximum size in MB of the key cache in memory.
- row-cache-capacity corresponds to the maximum size in MB of the row cache in memory.
- counter-cache-capacity corresponds to the maximum size in MB of the counter cache in memory.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The key-cache-capacity argument corresponds to the [key_cache_size_in_mb](#) parameter in the `cassandra.yaml`. Each key cache hit saves one seek and each row cache hit saves a minimum of two seeks. Devoting some memory to the key cache is usually a good tradeoff considering the positive effect on the response time. The default value is empty, which means a minimum of five percent of the heap in MB or 100 MB.

The row-cache-capacity argument corresponds to the `row_cache_size_in_mb` parameter in the `cassandra.yaml`. By default, row caching is zero (disabled).

The counter-cache-capacity argument corresponds to the `counter_cache_size_in_mb` in the `cassandra.yaml`. By default, counter caching is a minimum of 2.5% of Heap or 50MB.

nodetool setcachekeystosave

Sets the number of keys saved by each cache for faster post-restart warmup.

Synopsis

```
$ nodetool <options> setcachekeystosave -- <key-cache-keys-to-save> <row-cache-keys-to-save>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- key-cache-keys-to-save is the number of keys from the key cache to save to the saved caches directory.
- row-cache-keys-to-save is the number of keys from the row cache to save to the saved caches directory.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command saves the specified number of key and row caches to the saved caches directory, which you specify in the `cassandra.yaml`. The `key-cache-keys-to-save` argument corresponds to the `key_cache_keys_to_save` in the `cassandra.yaml`, which is disabled by default, meaning all keys will be saved. The `row-cache-keys-to-save` argument corresponds to the `row_cache_keys_to_save` in the `cassandra.yaml`, which is disabled by default.

nodetool setcompactionthreshold

Sets minimum and maximum compaction thresholds for a table.

Synopsis

```
$ nodetool <options> setcompactionthreshold -- <keyspace> <table> <minthreshold> <maxthreshold>
```

- Options are:
 - (-h | --host) <host name> | <ip address>

- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is a table name.
- minthreshold sets the minimum number of SSTables to trigger a minor compaction when using SizeTieredCompactionStrategy or DateTieredCompactionStrategy.
- maxthreshold sets the maximum number of SSTables to allow in a minor compaction when using SizeTieredCompactionStrategy or DateTieredCompactionStrategy.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This parameter controls how many SSTables of a similar size must be present before a minor [compaction](#) is scheduled. The [max_threshold](#) table property sets an upper bound on the number of SSTables that may be compacted in a single minor compaction, as described in <http://wiki.apache.org/cassandra/MemtableSSTable>.

When using LeveledCompactionStrategy, maxthreshold sets the MAX_COMPACTING_L0, which limits the number of L0 SSTables that are compacted concurrently to avoid wasting memory or running out of memory when compacting highly overlapping SSTables.

nodetool setcompactionthroughput

Sets the throughput capacity for compaction in the system, or disables throttling.

Synopsis

```
$ nodetool <options> setcompactionthroughput -- <value_in_mb>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- value_in_mb is the throughput capacity in MB per second for compaction.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable

- Orange (and) means not literal, indicates scope

Description

Set `value_in_mb` to 0 to disable throttling.

nodetool sethintedhandoffthrottlekb

Sets hinted handoff throttle in kb/sec per delivery thread. (Cassandra 2.1.1 and later)

Synopsis

```
$ nodetool <options> sethintedhandoffthrottlekb <value_in_kb/sec>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- `value_in_kb/sec` is the throttle time.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

When a node detects that a node for which it is holding hints has recovered, it begins sending the hints to that node. This setting specifies the maximum sleep interval per delivery thread in kilobytes per second after delivering each hint. The interval shrinks proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread uses the maximum interval; if there are three nodes, each node throttles to half of the maximum interval, because the two nodes are expected to deliver hints simultaneously.

Example

```
$ nodetool sethintedhandoffthrottlekb 2048
```

nodetool setlogginglevel

Set the log level for a service.

Synopsis

```
$ nodetool <options> setlogginglevel -- < class_qualifier > < level >
```

options are:

- (-h | --host) <host name> | <ip address>

- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- class_qualifier is the logger class qualifier, a fully qualified domain name, such as org.apache.cassandra.service.StorageProxy.
- level is the logging level, for example DEBUG.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

You can use this command to set logging levels for services instead of modifying the logback-text.xml file. The following values are valid for the logger class qualifier:

- org.apache.cassandra
- org.apache.cassandra.db
- org.apache.cassandra.service.StorageProxy

The possible log levels are:

- ALL
- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- OFF

If both class qualifier and level arguments to the command are empty or null, the command resets logging to the initial configuration.

Example

This command sets the StorageProxy service to debug level.

```
$ nodetool setlogginglevel org.apache.cassandra.service.StorageProxy DEBUG
```

nodetool setstreamthroughput

Sets the throughput capacity in MB for streaming in the system, or disable throttling.

Synopsis

```
$ nodetool <options> setstreamthroughput -- <value_in_mb>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>

- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- value_in_mb is the throughput capacity in MB per second for streaming.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Set value_in_MB to 0 to disable throttling.

nodetool settraceprobability

Sets the probability for tracing a request.

Synopsis

```
$ nodetool <options> settraceprobability -- <value>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- separates an option and argument that could be mistaken for a option.
- value is a probability between 0 and 1.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Probabilistic tracing is useful to determine the cause of intermittent query performance problems by identifying which queries are responsible. This option traces some or all statements sent to a cluster. Tracing a request usually requires at least 10 rows to be inserted.

A probability of 1.0 will trace everything whereas lesser amounts (for example, 0.10) only sample a certain percentage of statements. Care should be taken on large and active systems, as system-wide tracing will have a performance impact. Unless you are under very light load, tracing all requests (probability 1.0) will probably overwhelm your system. Start with a small fraction, for example, 0.001 and increase only if necessary. The trace information is stored in a system_traces keyspace that holds two tables – sessions and events, which can be easily queried to answer questions, such as what the most time-

consuming query has been since a trace was started. Query the parameters map and thread column in the `system_traces.sessions` and `events` tables for probabilistic tracing information.

To discover the current trace probability setting, use [nodetool gettraceprobability](#).

nodetool snapshot

Take a snapshot of one or more keyspaces, or of a table, to backup data.

Synopsis

```
$ nodetool <options> snapshot
    ( -cf <table> | --column-family <table> )
    (-kc <ktlist> | --kc.list <ktlist> | -kt <ktlist> | --kt-list <ktlist>)
    ( -t <tag> | --tag <tag> )
    -- ( <keyspace> ) | ( <keyspace> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- cf, or --column-family, followed by the name of the table to be backed up.
- kc, --kc.list, -kt, or --kt-list, followed by a list of keyspace.table names to be back up, ktlist.
- t or --tag, followed by the snapshot name.
- separates an option and argument that could be mistaken for a option.
- keyspace is a single keyspace name that is required when using the -cf option
- keyspace_list is one or more optional keyspace names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Use this command to [back up data](#) using a snapshot. See the examples below for various options.

Cassandra [flushes](#) the node before taking a snapshot, takes the snapshot, and stores the data in the [snapshots directory](#) of each keyspace in the data directory. If you do not specify the name of a snapshot directory using the -t option, Cassandra names the directory using the timestamp of the snapshot, for example 1391460334889. Follow the procedure for [taking a snapshot](#) before upgrading Cassandra. When upgrading, backup all keyspaces. For more information about snapshots, see [Apache documentation](#).

Example: All keyspaces

Take a snapshot of all keyspaces on the node. On Linux, in the Cassandra `bin` directory, for example:

```
$ nodetool snapshot
```

The following message appears:

```
Requested creating snapshot(s) for [all keyspaces] with snapshot name
[1391464041163]
Snapshot directory: 1391464041163
```

Because you did not specify a snapshot name, Cassandra names snapshot directories using the timestamp of the snapshot. If the keyspace contains no data, empty directories are not created.

Example: Single keyspace snapshot

Assuming you created the keyspace `cycling`, take a snapshot of the keyspace and name the snapshot `2015.07.17`.

```
$ nodetool snapshot -t 2015.07.17 cycling
```

The following message appears:

```
Requested creating snapshot(s) for [cycling] with snapshot name [2015.07.17]
Snapshot directory: 2015.07.17
```

Assuming the `cycling` keyspace contains two tables, `cyclist_name` and `upcoming_calendar`, taking a snapshot of the keyspace creates multiple snapshot directories named `2015.07.17`. A number of `.db` files containing the data are located in these directories. For example, from the installation directory:

```
$ cd data/data/cycling/cyclist_name-a882dca02aaf11e58c7b8b496c707234/
snapshots/2015.07.17
$ ls
```

```
la-1-big-CompressionInfo.db  la-1-big-Index.db          la-1-big-TOC.txt
la-1-big-Data.db            la-1-big-Statistics.db     la-1-big-Digest.adler32
la-1-big-Filter.db          la-1-big-Summary.db        manifest.json
```

```
$ cd data/data/cycling/cyclist_name-a882dca02aaf11e58c7b8b496c707234/
snapshots/2015.07.17
```

```
la-1-big-CompressionInfo.db  la-1-big-Index.db          la-1-big-TOC.txt
la-1-big-Data.db            la-1-big-Statistics.db     la-1-big-Digest.adler32
la-1-big-Filter.db          la-1-big-Summary.db        manifest.json
```

Example: Multiple keyspaces snapshot

Assuming you created a keyspace named `mykeyspace` in addition to the `cycling` keyspace, take a snapshot of both keyspaces.

```
$ nodetool snapshot mykeyspace cycling
```

The following message appears:

```
Requested creating snapshot(s) for [mykeyspace, cycling] with snapshot name
[1391460334889]
Snapshot directory: 1391460334889
```

Example: Single table snapshot

Take a snapshot of only the `cyclist_name` table in the `cycling` keyspace.

```
$ nodetool snapshot --table cyclist_name cycling
```

```
Requested creating snapshot(s) for [cycling] with snapshot name
[1391461910600]
Snapshot directory: 1391461910600
```

Cassandra creates the snapshot directory named 1391461910600 that contains the backup data of cyclist_name table in data/data/cycling/cyclist_name-a882dca02aaf11e58c7b8b496c707234/snapshots, for example.

Example: List of different keyspace.tables snapshot

Take a snapshot of several tables in different keyspaces, such as the cyclist_name table in the cycling keyspace and the sample_times table in the test keyspace. The keyspace.table list should be comma-delimited with no spaces.

```
$ nodetool snapshot -kt cycling.cyclist_name,test.sample_times
```

```
Requested creating snapshot(s) for [cycling.cyclist_name,test.sample_times]
with snapshot name [1431045288401]
Snapshot directory: 1431045288401
```

nodetool status

Provide information about the cluster, such as the state, load, and IDs.

Synopsis

```
$ nodetool <options> status ( -r | --resolve-ip ) -- <keyspace>
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- r, or --resolve-ip, means to provide node names instead of IP addresses.
- separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The status command provides the following information:

- Status - U (up) or D (down)
Indicates whether the node is functioning or not.
- State - N (normal), L (leaving), J (joining), M (moving)

The state of the node in relation to the cluster.

- Address

The node's URL.

- Load - updates every 90 seconds

The amount of file system data under the cassandra data directory after excluding all content in the snapshots subdirectories. Because all SSTable data files are included, any data that is not cleaned up, such as TTL-expired cell or tombstoned data) is counted.

- Tokens

The number of tokens set for the node.

- Owns

The percentage of the data owned by the node per data center times the replication factor. For example, a node can own 33% of the ring, but show 100% if the replication factor is 3.

Attention: If your cluster uses keyspaces having different replication strategies or replication factors, specify a keyspace when you run `nodetool status` to get meaningful ownership information.

- Host ID

The network ID of the node.

- Rack

The rack or, in the case of Amazon EC2, the availability zone of the node.

Example

This example shows the output from running `nodetool status`.

```
$ nodetool status mykeyspace
```

```
Datacenter: datacenter1
```

```
=====
```

```
Status=Up/Down
```

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

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

```
  Rack
```

```
UN 127.0.0.1  47.66 KB    1         33.3%   aa1b7c1-6049-4a08-ad3e-3697a0e30e10
```

```
  rack1
```

```
UN 127.0.0.2  47.67 KB    1         33.3%   1848c369-4306-4874-afdf-5c1e95b8732e
```

```
  rack1
```

```
UN 127.0.0.3  47.67 KB    1         33.3%   49578bf1-728f-438d-b1c1-d8dd644b6f7f
```

```
  rack1
```

nodetool statusbackup

Synopsis

```
$ nodetool <options> statusbackup
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

Provides the status of backup.

nodetool statusbinary

Provide the status of native transport.

Synopsis

```
$ nodetool <options> statusbinary
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides the status of the binary protocol, also known as the native transport.

nodetool statusgossip

Synopsis

```
$ nodetool <options> statusgossip
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides the status of gossip.

nodetool statushandoff

Synopsis

```
$ nodetool <options> statushandoff
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>
- (-pwf <passwordFilePath | --password-file <passwordFilePath>)

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

Provides the status of hinted handoff.

nodetool statusthrift

Provide the status of the Thrift server.

Synopsis

```
$ nodetool <options> statusthrift
```

Options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>

- (`-pw | --password`) `<password>`
- (`-u | --username`) `<user name>`
- (`-pwf <passwordFilePath> | --password-file <passwordFilePath>`)

Synopsis Legend

- Angle brackets (`<>`) mean not literal, a variable
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

nodetool stop

Stops the compaction process.

Synopsis

```
$ nodetool <options> stop -- <compaction_type>
```

Table: Options

Short	Long	Description
<code>-h</code>	<code>--host</code>	Hostname or IP address
<code>-id</code>	<code>--compaction-id</code>	Compaction ID
<code>-p</code>	<code>--port</code>	Port number
<code>-pwf</code>	<code>--password-file</code>	Password file path
<code>-pw</code>	<code>--password</code>	Password
<code>-u</code>	<code>--username</code>	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool stop` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.
- Valid compaction types: `COMPACTION`, `VALIDATION`, `CLEANUP`, `SCRUB`, `INDEX_BUILD`

Synopsis Legend

- Angle brackets (`<>`) mean not literal, a variable
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

Description

Stops all compaction operations from continuing to run. This command is typically used to stop a compaction that has a negative impact on the performance of a node. After the compaction stops, Cassandra continues with the remaining operations in the queue. Eventually, Cassandra restarts the compaction.

In Cassandra 2.2 and later, a single compaction operation can be stopped with the `-id` option. Run `nodetool compactionstats` to find the compaction ID.

nodetool stopdaemon

Stops the cassandra daemon.

Synopsis

```
$ nodetool <options> stopdaemon
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool tablehistograms

Provides statistics about a table that could be used to plot a frequency function.

Synopsis

- Options are:
 - (`-h` | `--host`) <host name> | <ip address>
 - (`-p` | `--port`) <port number>
 - (`-pw` | `--password`) <password >
 - (`-u` | `--username`) <user name>
 - (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)
- `--` separates an option from an argument that could be mistaken for a option.
- `keyspace` is the name of a keyspace.
- `table` is the name of a table.
- `<keyspace>.<table>` or `<keyspace> <table>` can be used to designate the table.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional

- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The `nodetool tablehistograms` command provides statistics about a table, including read/write latency, partition size, column count, and number of SSTables. The report covers all operations since the last time `nodetool tablehistograms` was run in the current session. The use of the `metrics-core` library in Cassandra 2.1 and later makes the output more informative and easier to understand.

Example

For example, to get statistics about the `libout` table in the `libdata` keyspace on Linux, use this command:

```
$ install_location nodetool tablehistograms libdata libout
```

Output is:

libdata/libout histograms				
Percentile	SSTables	Write Latency	Read Latency	Partition Size
	Cell Count	(micros)	(micros)	(bytes)
50%	0.00	39.50	36.00	1597
	42			
75%	0.00	49.00	55.00	1597
	42			
95%	0.00	95.00	82.00	8239
	258			
98%	0.00	126.84	110.42	17084
	446			
99%	0.00	155.13	123.71	24601
	770			
Min	0.00	3.00	3.00	1110
	36			
Max	0.00	50772.00	314.00	126934
	3973			

The output shows the percentile rank of read and write latency values, the partition size, and the cell count for the table.

nodetool tablestats

Provides statistics about tables.

Synopsis

```
$ nodetool <options> tablestats -i -- (<keyspace>.<table> ... ) -H
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- -- separates an option from an argument that could be mistaken for a option.
- i ignores the following tables, providing only information about other Cassandra tables

- `keyspace.table` is one or more keyspace and table names in dot notation.
- `H` converts bytes to a human readable form: kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). (Cassandra 2.1.1)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The `nodetool tablestats` command provides statistics about one or more tables. You use dot notation to specify one or more keyspace and table names. If you do not specify a keyspace and table, Cassandra provides statistics about all tables. If you use the `-i` option, Cassandra provides statistics about all tables except the given ones. The use of the metrics-core library in Cassandra 2.1 and later makes the output more informative and easier to understand.

This table describes the `nodetool tablestats` output.

Table: `nodetool tablestats` output

Name of statistic	Example value	Brief description	Related information
Keyspace	libdata	Name of the keyspace	Keyspace and table
Read count	11207	Number of requests to read tables in the libdata keyspace since startup	
Read latency	0.047. . . ms	Latency reading the tables in the libdata keyspace	
Write count	17598	Number of requests to update tables in the libdata keyspace since startup	
Write latency	0.053. . . ms	Latency writing tables in the libdata keyspace	
Pending tasks	0	Tasks in the queue for reads, writes,	

Name of statistic	Example value	Brief description	Related information
		and cluster operations of tables in the keyspace	
Table	libout	Name of the Cassandra table	
SSTable count	3	Number of SSTables containing data from the table	How to use the SSTable counts metric
Space used (live), bytes:	9592399	Space used by the table (depends on operating system)	
Space used (total), bytes:	9592399	Same as above	Same as above
Space used by snapshots (total), bytes:	0	Same occupied by backup data	
SSTable compression ratio	0.367. . .	Fraction of data-representation size resulting from compression	Types of compression option)
Memtable cell count	1022550	Number of cells (storage engine rows x columns) of data in the memtable	Cassandra memtable structure in memory
Memtable data size, bytes	32028148	Size of the memtable data	Same as above
Memtable switch count	3	Number of times a full memtable was swapped for an empty one (Increases each time the memtable	How memtables are measured article

Name of statistic	Example value	Brief description	Related information
		for a table is flushed to disk)	
Local read count	11207	Number of local read requests for the libout table since startup	
Local read latency	0.048 ms	Round trip time in milliseconds to complete a request to read the libout table	Factors that affect read latency
Local write count	17598	Number of local requests to update the libout the table since startup	
Local write latency	0.054 ms	Round trip time in milliseconds to complete an update to the libout table	Factors that affect write latency
Pending tasks	0	Number of read, write, and cluster operations that are pending	
Bloom filter false positives	0	Number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers	Tuning bloom filters

Name of statistic	Example value	Brief description	Related information
Bloom filter false ratio	0.00000	Fraction of all bloom filter checks resulting in a false positive	Same as above
Bloom filter space used, bytes	11688	Size of bytes of the bloom filter data	Same as above
Compacted partition minimum bytes	1110	Lower size limit for the partition being compacted in memory	Used to calculate what the approximate row cache size should be. Multiply the reported row cache size, which is the number of rows in the cache, by the compacted row mean size for every table and sum them.
Compacted partition maximum bytes	126934	Upper size limit for compacted table rows.	Configurable in the cassandra.yaml in_memory_compaction_limit_in_mb
Compacted partition mean bytes	2730	The average size of compacted table rows	
Average live cells per slice (last five minutes)	0.0	Average of cells scanned by single key queries during the last five minutes	
Average tombstones per slice (last five minutes)	0.0	Average of tombstones scanned by single key queries during the last five minutes	

Examples

This example shows an excerpt of the output of the command after flushing a table of library data to disk.

```
$ nodetool tablestats libdata.libout
Keyspace: libdata
  Read Count: 11207
  Read Latency: 0.047931114482020164 ms.
  Write Count: 17598
  Write Latency: 0.053502954881236506 ms.
  Pending Flushes: 0
  Table: libout
```

```

SSTable count: 3
Space used (live), bytes: 9088955
Space used (total), bytes: 9088955
Space used by snapshots (total), bytes: 0
SSTable Compression Ratio: 0.36751363892150946
Memtable cell count: 0
Memtable data size, bytes: 0
Memtable switch count: 3
Local read count: 11207
Local read latency: 0.048 ms
Local write count: 17598
Local write latency: 0.054 ms
Pending flushes: 0
Bloom filter false positives: 0
Bloom filter false ratio: 0.00000
Bloom filter space used, bytes: 11688
Compacted partition minimum bytes: 1110
Compacted partition maximum bytes: 126934
Compacted partition mean bytes: 2730
Average live cells per slice (last five minutes): 0.0
Average tombstones per slice (last five minutes): 0.0

```

Using the human-readable option

Using the human-readable -H option provides output in easier-to-read units than bytes. For example:

```

$ nodetool tablestats demodb.nhanes -H
Keyspace: demodb
Read Count: 0
Read Latency: NaN ms.
Write Count: 20050
Write Latency: 0.08548014962593516 ms.
Pending Flushes: 0
Table: nhanes
SSTable count: 1
Space used (live): 13.75 MB
Space used (total): 13.75 MB
Space used by snapshots (total): 0 bytes
SSTable Compression Ratio: 0.3064650643762481
Memtable cell count: 0
Memtable data size: 0 bytes
Memtable switch count: 1
Local read count: 0
Local read latency: NaN ms
Local write count: 20050
Local write latency: 0.085 ms
Pending flushes: 0
Bloom filter false positives: 0
Bloom filter false ratio: 0.00000
Bloom filter space used: 23.73 KB
Compacted partition minimum bytes: 1.87 KB
Compacted partition maximum bytes: 2.69 KB
Compacted partition mean bytes: 2.26 KB
Average live cells per slice (last five minutes): 0.0
Maximum live cells per slice (last five minutes): 0.0
Average tombstones per slice (last five minutes): 0.0
Maximum tombstones per slice (last five minutes): 0.0
-----

```

nodetool toppartitions

Synopsis

```
$ nodetool <options> toppartitions <keyspace> <table> <duration> ...
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- a <samplers> comma separated list of samplers to use (default: all)
- k <topCount> number of the top partitions to list (default: 10)
- s <size> capacity of stream summary, closer to the actual cardinality of partitions will yield more accurate results (default: 256)
- keyspace is a keyspace name
- cfname is a column family name
- duration in milliseconds

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool toppartitions` command samples and prints the most active partitions during the duration specified. A keyspace and column family must be specified, as well as a duration in milliseconds.

Examples

Sample the most active partitions for the table `test.users` for 1,000 milliseconds

```
nodetool toppartitions test users 1000
```

Output is produced, similar to the following:

```

READS Sampler:
  Cardinality: ~0 (256 capacity)
  Top 10 partitions:
    Nothing recorded during sampling period...

WRITES Sampler:
  Cardinality: ~1 (256 capacity)
  Top 10 partitions:
    Partition      Count      +/-
    10             1          0
    11             1          0
    12             1          0
    13             1          0
    14             1          0
    15             1          0
    16             1          0
    17             1          0
    18             1          0
    19             1          0

```

nodetool tpstats

Provides usage statistics of thread pools.

Synopsis

```
$ nodetool <options> tpstats
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>
- (-pwf <passwordFilePath> | --password-file <passwordFilePath>)

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Cassandra is based on a Staged Event Driven Architecture (SEDA). Different tasks are separated into stages that are connected by a messaging service. Stages have a queue and thread pool. Some stages skip the messaging service and queue tasks immediately on a different stage if it exists on the same node. The queues can back up if executing at the next stage is too busy. Having a queue get backed up can cause performance bottlenecks. `nodetool tpstats` provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool. A high number of pending tasks for any pool can indicate performance problems, as described in <http://wiki.apache.org/cassandra/Operations#Monitoring>.

Run the `nodetool tpstats` command on a local node for get thread pool statistics.

This table describes key indicators:

Table: nodetool tpstats output

Name of statistic	Task	Related information
AntiEntropyStage	Repair consistency	Nodetool repair
CacheCleanupExecutor	Clean the cache	
CommitlogArchiver	Archives commitlog	
CompactionExecutor	Runs compaction	
CounterMutationStage	Stage counter changes	Will back up if the write rate exceeds the mutation rate. A high pending count will be seen if consistency level is set to ONE and there is a high counter increment workload.
GossipStage	Handle gossip rounds every second	Out of sync schemas can cause issues. nodetool resetlocalschema may need to be used.
HintedHandoff	Send missed mutations to other nodes	Usually symptom of a problem elsewhere. Use nodetool disablehandoff and run repair .
InternalResponseStage	Respond to non-client initiated messages, including bootstrapping and schema checking	
MemtableFlushWorker	Writes memtable contents to disk	Will back up if the queue is overrunning the disk I/O capabilities. Sorting can also cause issues if the queue has a high load associated with a small number of flushes. Cause can be huge rows with large column names or inserting too many values into a CQL collection. For disk issues, add nodes or tune configuration.
MemtablePostFlushOp	Operations after flushing the memtable	Discard commit log files and flush secondary indexes.

Name of statistic	Task	Related information
MemtableReclaimMemory	Makes unused memory available	
MigrationStage	Make schema changes	
MiscStage	Miscellaneous operations	Snapshotting, replicating data after node remove completed.
MutationStage	Local writes	A high number of pending write requests indicates a problem handling them. Adding a node, tuning hardware and configuration, or updating data models will improve handling.
PendingRangeCalculator	Calculate pending ranges per bootstraps and departed nodes	Developer notes
ReadRepairStage	A digest query and update of replicas of a key	Fast providing good connectivity between replicas exists. If pending grows too large, attempt to lower the rate for high-read tables by altering the table to use a smaller read_repair_chance value, like 0.11.
ReadStage	Local reads	Performing a local read. Also includes deserializing data from row cache. Pending values can cause increased read latency. Generally resolved by adding nodes or tuning the system.
RequestResponseStage	Handle responses from other nodes	
ValidationExecutor	Validates schema	

Table: Droppable Messages

Message Type	Stage	Notes
BINARY	n/a	This is deprecated and no longer has any use
_TRACE	n/a (special)	Used for recording traces (nodetool settraceprobability) Has a special executor (1 thread, 1000 queue depth) that throws away messages on insertion instead of within the execute
MUTATION	MutationStage	If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted

		handoff and read repairs to do the mutation if it succeeded.
COUNTER_MUTATION	MutationStage	If a write message is processed after its timeout (write_request_timeout_in_ms) it either sent a failure to the client or it met its requested consistency level and will relay on hinted handoff and read repairs to do the mutation if it succeeded.
READ_REPAIR	MutationStage	Times out after write_request_timeout_in_ms
READ	ReadStage	Times out after read_request_timeout_in_ms. No point in servicing reads after that point since it would of returned error to client
RANGE_SLICE	ReadStage	Times out after range_request_timeout_in_ms.
PAGED_RANGE	ReadStage	Times out after request_timeout_in_ms.
REQUEST_RESPONSE	RequestResponseStage	Times out after request_timeout_in_ms. Response was completed and sent back but not before the timeout

Example

Run the command every two seconds.

```
$ nodetool -h labcluster tpstats
```

Example output is:

Pool Name	Active	Pending	Completed	Blocked	All
time blocked					
CounterMutationStage	0	0	0	0	
0					
ReadStage	0	0	103	0	
0					
RequestResponseStage	0	0	0	0	
0					
MutationStage	0	0	13234794	0	
0					
ReadRepairStage	0	0	0	0	
0					
GossipStage	0	0	0	0	
0					
CacheCleanupExecutor	0	0	0	0	
0					
AntiEntropyStage	0	0	0	0	
0					
MigrationStage	0	0	11	0	
0					

ValidationExecutor	0	0	0	0
0				
CommitLogArchiver	0	0	0	0
0				
MiscStage	0	0	0	0
0				
MemtableFlushWriter	0	0	126	0
0				
MemtableReclaimMemory	0	0	126	0
0				
PendingRangeCalculator	0	0	1	0
0				
MemtablePostFlush	0	0	1468	0
0				
CompactionExecutor	0	0	254	0
0				
InternalResponseStage	0	0	1	0
0				
HintedHandoff	0	0	0	
Message type	Dropped			
RANGE_SLICE	0			
READ_REPAIR	0			
PAGED_RANGE	0			
BINARY	0			
READ	0			
MUTATION	180			
_TRACE	0			
REQUEST_RESPONSE	0			
COUNTER_MUTATION	0			

nodetool truncatehints

Truncates all hints on the local node, or truncates hints for the one or more endpoints.

Synopsis

```
$ nodetool <options> truncatehints -- ( <endpoint> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath> | --password-file <passwordFilePath>)
- separates an option and argument that could be mistaken for a option.
- endpoint is one or more endpoint IP addresses or host names which hints are to be deleted.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

nodetool upgradestables

Rewrites SSTables for tables that are not running the current version of Cassandra.

Synopsis

```
$ nodetool <options> upgradestables
( -a | --include-all-sstables )
-- <keyspace> ( <table> ... )
```

- Options are:
 - (-h | --host) <host name> | <ip address>
 - (-p | --port) <port number>
 - (-pw | --password) <password >
 - (-u | --username) <user name>
 - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- a or --include-all-sstables, followed by the snapshot name.
- separates an option and argument that could be mistaken for a option.
- keyspace a keyspace name.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Rewrites SSTables on a node that are incompatible with the current version. Use this command when upgrading your server or changing compression options.

nodetool verify

Verify (check data checksum for) one or more tables.

Synopsis

```
$ nodetool [options] verify [(-e | --extended-verify)] [--] [<keyspace>
<tables>...]
```

Table: Options

Short	Long	Description
-h	--host	Hostname or IP address
-p	--port	Port number
-pwf	--password-file	Password file path
-pw	--password	Password
-u	--username	User name

Note:

- For tarball installations, execute the command from the *install_location/bin* directory.
- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- `nodetool verify` operates on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the target; otherwise, for remote invocation, identify the target node, or nodes, using `-h`.

Synopsis Legend

In the synopsis section of each statement, formatting has the following meaning:

- Uppercase means literal
- Lowercase means not literal
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

A semicolon that terminates CQL statements is not included in the synopsis.

Description

The `nodetool verify` command checks the data checksum for one or more specified tables. An optional argument, `-e` or `--extended-verify`, will verify each cell data, whereas without the option, only the SSTable checksums are verified.

Examples

```
$ nodetool -u cassandra -pw cassandra verify cycling cyclist_name
```

nodetool version

Provides the version number of Cassandra running on the specified node.

Synopsis

```
$ nodetool <options> version
```

Options are:

- (`-h` | `--host`) <host name> | <ip address>
- (`-p` | `--port`) <port number>
- (`-pw` | `--password`) <password >
- (`-u` | `--username`) <user name>
- (`-pwf` <passwordFilePath> | `--password-file` <passwordFilePath>)

Synopsis Legend

- Angle brackets (`< >`) mean not literal, a variable
- Italics mean optional
- The pipe (`|`) symbol means OR or AND/OR
- Ellipsis (`...`) means repeatable
- Orange (`and`) means not literal, indicates scope

The cassandra utility

In Cassandra 3.2 and later, start-up parameters can be specified in the `jvm.options` file (package or tarball installations) or run from the command line in tarball installations.

Note: If you are using Cassandra 3.1, see the [Cassandra 3.0 documentation](#).

You can also use the `jvm.options` file to pass additional options, such as maximum and minimum heap size, to the Java virtual machine rather than setting them in the environment.

Usage

Add the following to the `jvm.options` file:

```
JVM_OPTS="$JVM_OPTS -D[PARAMETER] "
```

For Tarball installations, you can run this tool from the command line:

```
$ cassandra [OPTIONS]
```

Examples:

- Command line: `bin/cassandra -Dcassandra.load_ring_state=false`
- `jvm.options`: `JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"`

The [Example section](#) contains more examples.

Command line only options

Option	Description
-f	Start the cassandra process in foreground. The default is to start as background process.
-h	Help.
-p <i>filename</i>	Log the process ID in the named file. Useful for stopping Cassandra by killing its PID.
-v	Print the version and exit.

Start-up parameters

The `-D` option specifies the start-up parameters in both the command line and `cassandra-env.sh` file.

cassandra.auto_bootstrap=false

Facilitates setting [auto_bootstrap](#) to false on initial set-up of the cluster. The next time you start the cluster, you do not need to change the `cassandra.yaml` file on each node to revert to true.

cassandra.available_processors=number_of_processors

In a multi-instance deployment, multiple Cassandra instances will independently assume that all CPU processors are available to it. This setting allows you to specify a smaller set of processors.

cassandra.boot_without_jna=true

If JNA fails to initialize, Cassandra fails to boot. Use this command to boot Cassandra without JNA.

cassandra.config=directory

The directory location of the `cassandra.yaml` file. The default location depends on the type of installation.

cassandra.initial_token=token

Use when virtual nodes (vnodes) are not used. Sets the initial partitioner token for a node the first time the node is started. (Default: disabled)

Note: Vnodes are highly recommended as they automatically select tokens.

cassandra.join_ring=true/false

Set to false to start Cassandra on a node but not have the node join the cluster. (Default: true) You can use [nodetool join](#) and a JMX call to join the ring afterwards.

cassandra.load_ring_state=true/false

Set to false to clear all gossip state for the node on restart. (Default: true)

cassandra.metricsReporterConfigFile=file

Enable pluggable metrics reporter. See [Pluggable metrics reporting in Cassandra 2.0.2](#).

cassandra.native_transport_port=port

Set the port on which the CQL native transport listens for clients. (Default: 9042)

cassandra.partitioner=partitioner

Set the partitioner. (Default: `org.apache.cassandra.dht.Murmur3Partitioner`)

cassandra.replace_address=listen_address or broadcast_address of dead node

To replace a node that has died, restart a new node in its place specifying the [listen_address](#) or [broadcast_address](#) that the new node is assuming. The new node must not have any data in its data directory, that is, it must be in the same state as before bootstrapping.

Note: The broadcast_address defaults to the listen_address except when using the [Ec2MultiRegionSnitch](#).

cassandra.replayList=table

Allow restoring specific tables from an archived commit log.

cassandra.ring_delay_ms=ms

Defines the amount of time a node waits to hear from other nodes before formally joining the ring. (Default: 1000ms)

cassandra.rpc_port=port

Set the port for the Thrift RPC service, which is used for client connections. (Default: 9160).

cassandra.ssl_storage_port=port

Set the SSL port for encrypted communication. (Default: 7001)

cassandra.start_native_transport=true/false

Enable or disable the native transport server. See [start_native_transport](#) in `cassandra.yaml`. (Default: true)

cassandra.start_rpc=true/false

Enable or disable the Thrift RPC server. (Default: true)

cassandra.storage_port=port

Set the port for inter-node communication. (Default: 7000)

cassandra.triggers_dir=directory

Set the default location for the triggers JARs.

cassandra.write_survey=true

For testing new compaction and compression strategies. It allows you to experiment with different strategies and benchmark write performance differences without affecting the production workload. See [Testing compaction and compression](#) on page 157.

consistent.rangemovement=true

True makes bootstrapping behavior effective.

Example

Clear gossip state when starting a node:

- Command line: `bin/cassandra -Dcassandra.load_ring_state=false`
- `jvm.options`: `JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"`

Start Cassandra on a node and do not join the cluster:

- Command line: `bin/cassandra -Dcassandra.join_ring=false`
- `jvm.options`: `JVM_OPTS="$JVM_OPTS -Dcassandra.join_ring=false"`

Replacing a dead node:

- Command line: `bin/cassandra -Dcassandra.replace_address=10.91.176.160`
- `jvm.options`: `JVM_OPTS="$JVM_OPTS -Dcassandra.replace_address=10.91.176.160"`

The cassandra-stress tool

The `cassandra-stress` tool is a Java-based stress testing utility for basic benchmarking and load testing a Cassandra cluster.

Data modeling choices can greatly affect application performance. Significant load testing over several trials is the best method for discovering issues with a particular data model. The `cassandra-stress` tool is an effective tool for populating a cluster and stress testing CQL tables and queries. Use the `cassandra-stress` to:

- Quickly determine how a schema performs.
- Understand how your database scales.
- Optimize your data model and settings.
- Determine production capacity.

The `cassandra-stress` tool also supports a YAML-based profile for defining specific schema with potential compaction strategies, cache settings, and types. Sample files are located in:

- Package installations: `/usr/share/docs/cassandra/examples`
- Tarball installations: `install_location/tools/`

The YAML file includes user-defined keyspace, tables, and schema. The YAML file can be used for both read, write, and mixed workloads.

When started without a YAML file, `cassandra-stress` creates a keyspace `keyspace1` and tables `standard1` or `counter1`, depending on what type of table tested. These are automatically created the first time you run a stress test and reused on subsequent runs. The keyspace `keyspace1` can be dropped using [DROP KEYSPACE](#). You cannot change the default keyspace and tables names without using a YAML file.

Usage:

- Package installations:

```
$ /usr/bin/cassandra-stress command [options]
```

- Tarball installations:

```
$ cd install_location/tools
$ bin/cassandra-stress command [options]
```

On tarball installations, you can use these commands and options with or without the [cassandra-stress daemon](#) running.

Command	Description
read	Multiple concurrent reads. The cluster must first be populated by a write test.
write	Multiple concurrent writes against the cluster.
mixed	Interleave basic commands with configurable ratio and distribution. The cluster must first be populated by a write test.
counter_write	Multiple concurrent updates of counters.
counter_read	Multiple concurrent reads of counters. The cluster must first be populated by a counter_write test.
user	Interleave user provided queries with configurable ratio and distribution.
help	Display help for a command or option. Display help for an option: <code>cassandra-stress help [options]</code> For example: <code>cassandra-stress help -schema</code>
print	Inspect the output of a distribution definition.
legacy	Legacy support mode.

Important: Additional sub options are available for each option in the following table. Format:

```
$ cassandra-stress help option
```

Option	Description
- pop	Population distribution and intra-partition visit order.
Usage	<pre>\$ -pop seq=? [no-wrap] [read-lookback=DIST(?)] [contents=?] or \$ -pop [dist=DIST(?)] [contents=?]</pre>
- insert	Insert specific options relating to various methods for batching and splitting partition updates.
Usage	<pre>\$ -col [n=DIST(?)] [slice] [super=?] [comparator=?] [timestamp=?] [size=DIST(?)]</pre>
- col	Column details, such as size and count distribution, data generator, names, and comparator.
Usage	<pre>\$ -col [n=DIST(?)] [slice] [super=?] [comparator=?] [timestamp=?] [size=DIST(?)]</pre>
- rate	Thread count, rate limit, or automatic mode (default is auto).
Usage	<pre>\$ -rate threads=? [limit=?] or \$ -rate [threads>=?] [threads<=?] [auto]</pre>
- mode	Thrift or CQL with options.

Option	Description
Usage	<pre>\$ -mode thrift [smart] [user=?] [password=?] or \$ -mode native [unprepared] cql3 [compression=?] [port=?] [user=?] [password=?] or \$ -mode simplenative [prepared] cql3 [port=?]</pre>
- errors	How to handle errors when encountered during stress.
Usage	<pre>\$ -errors [retries=?] [ignore]</pre>
- sample	Specify the number of samples to collect for measuring latency.
Usage	<pre>\$ -sample [history=?] [live=?] [report=?]</pre>
- schema	Replication settings, compression, compaction, and so on.
Usage	<pre>\$ -schema [replication(?)] [keyspace=?] [compaction(?)] [compression=?]</pre>
- node	Nodes to connect to.
Usage	<pre>\$ -node [whitelist] [file=?] []</pre>
- log	Where to log progress and the interval to use.
Usage	<pre>\$ -log [level=?] [no-summary] [file=?] [interval=?]</pre>
- transport	Custom transport factories.
Usage	<pre>\$ -transport [factory=?] [truststore=?] [truststore-password=?] [ssl- protocol=?] [ssl-alg=?] [store-type=?] [ssl-ciphers=?]</pre>
- port	Specify port for connecting Cassandra nodes. Port can be specified for Cassandra native protocol, Thrift protocol or a JMX port for retrieving statistics.
Usage	<pre>\$ -port [native=?] [thrift=?] [jmx=?]</pre>
- sendto	Specify a stress server to send this command to.
Usage	<pre>\$ -sendToDaemon <host></pre>
- graph	Graph results of cassandra-stress tests. Multiple tests can be graphed together.
Usage	<pre>\$ -graph file=? [title=?] [revision=?]</pre>

Additional command-line options can modify how `cassandra-stress` runs:

Command	Description
<code>profile=?</code>	Designate the YAML file to use with <code>cassandra-stress</code> .
<code>ops(?)</code>	Specify what operations (inserts and/or queries) to run and the number of each.
<code>clustering=DIS</code>	Distribution clustering runs of operations of the same kind.
<code>err<?</code>	Specify a standard error of the mean; when this value is reached, <code>cassandra-stress</code> will end. Default is 0.02.
<code>n>?</code>	Specify a minimum number of iterations to run before accepting uncertainly convergence.
<code>n<?</code>	Specify a maximum number of iterations to run before accepting uncertainly convergence.
<code>n=?</code>	Specify the number of operations to run.
<code>duration=?</code>	Specify the time to run, in seconds, minutes or hours.
<code>no-warmup</code>	Do not warmup the process, do a cold start.
<code>truncate=?</code>	Truncate the table created during <code>cassandra-stress</code> . Options are never, once, or always. Default is never.
<code>cl=?</code>	Set the consistency level to use during <code>cassandra-stress</code> . Options are ONE, QUORUM, LOCAL_QUORUM, EACH_QUORUM, ALL, and ANY. Default is LOCAL_ONE.

Simple read and write examples

```
$ cassandra-stress write n=1000000
```

Insert (write) one million rows.

```
$ cassandra-stress read n=200000
```

Read two hundred thousand rows.

```
$ cassandra-stress read duration=3m
```

Read rows for a duration of 3 minutes.

```
$ cassandra-stress read n=200000 no-warmup
```

Read 200,000 rows without a warmup of 50,000 rows first.

View schema help

```
$ cassandra-stress help -schema
```

```
replication([strategy=?] [factor=?] [<option 1..N>=?]):      Define
  the replication strategy and any parameters
  strategy=? (default=org.apache.cassandra.locator.SimpleStrategy) The
  replication strategy to use
  factor=? (default=1)                                         The
  number of replicas
  keyspace=? (default=keyspace1)                               The
  keyspace name to use
compaction([strategy=?] [<option 1..N>=?]):                  Define
  the compaction strategy and any parameters
```



```
strategy=?
compaction strategy to use
compression=?
Specify the compression to use for SSTable, default:no compression
```

Populate the database

Generally it is easier to let `cassandra-stress` create the basic schema and then modify it in [CQL](#):

```
#Load one row with default schema
$ cassandra-stress write n=1 cl=one -mode native cql3 -log file=~/  
create_schema.log

#Modify schema in CQL
$ cqlsh

#Run a real write workload
$ cassandra-stress write n=1000000 cl=one -mode native cql3 -schema  
keyspace="keyspace1" -log file=~/load_1M_rows.log
```

Changing the replication strategy

Changes the replication strategy to `NetworkTopologyStrategy`.

```
$ cassandra-stress write n=500000 no-warmup -node existing0 -schema  
"replication(strategy=NetworkTopologyStrategy, existing=2)"
```

Running a mixed workload

When running a mixed workload, you must escape parentheses, greater-than and less-than signs, and other such things. This example invokes a workload that is one-quarter writes and three-quarters reads.

```
$ cassandra-stress mixed ratio\(write=1,read=3\) n=100000 cl=ONE -pop  
dist=UNIFORM\(1..1000000\) -schema keyspace="keyspace1" -mode native cql3 -  
rate threads\>=16 threads\<=256 -log file=~mixed_autorate_50r50w_1M.log
```

Notice the following in this example:

1. The `ratio` requires backslash-escaped parenthesis.
2. The value of `n` is different than in write phase. During the write phase, `n` records are written. However in the read phase, if `n` is too large, it is inconvenient to read *all* the records for simple testing. Generally, `n` does not need be large when validating the persistent storage systems of a cluster.

The `-pop dist=UNIFORM\(1..1000000\)` portion says that of the `n=100,000` operations, select the keys uniformly distributed between 1 and 1,000,000. Use this when you want to specify more data per node than what fits in DRAM.

3. In the `rate` section, the greater-than and less-than signs are escaped. If not escaped, the shell will attempt to use them for IO redirection. Specifically, the shell will try to read from a non-existent file called `=256` and create a file called `=16`. The `rate` section tells `cassandra-stress` to automatically attempt different numbers of client threads and not test less than 16 or more than 256 client threads.

Standard mixed read/write workload keyspace for a single node

```
CREATE KEYSPACE "keyspace1" WITH replication = {  
  'class': 'SimpleStrategy',
```

```

    'replication_factor': '1'
  };
  USE "keyspace1";
  CREATE TABLE "standard1" (
    key blob,
    "C0" blob,
    "C1" blob,
    "C2" blob,
    "C3" blob,
    "C4" blob,
    PRIMARY KEY (key)
  ) WITH
    bloom_filter_fp_chance=0.010000 AND
    caching='KEYS_ONLY' AND
    comment='' AND
    dclocal_read_repair_chance=0.000000 AND
    gc_grace_seconds=864000 AND
    index_interval=128 AND
    read_repair_chance=0.100000 AND
    replicate_on_write='true' AND
    default_time_to_live=0 AND
    speculative_retry='99.0PERCENTILE' AND
    memtable_flush_period_in_ms=0 AND
    compaction={'class': 'SizeTieredCompactionStrategy'} AND
    compression={'class': 'LZ4Compressor'};

```

Splitting up a load over multiple cassandra-stress instances on different nodes

This example is useful for loading into large clusters, where a single `cassandra-stress` load generator node cannot saturate the cluster. In this example, `$NODES` is a variable whose value is a comma delimited list of IP addresses such as `10.0.0.1,10.0.0.2`, and so on.

```

#On Node1
$$ cassandra-stress write n=1000000 cl=one -mode native cql3 -schema
  keyspace="keyspace1" -pop seq=1..1000000 -log file=~/node1_load.log -node
  $NODES

#On Node2
$ cassandra-stress write n=1000000 cl=one -mode native cql3 -schema
  keyspace="keyspace1" -pop seq=1000001..2000000 -log file=~/node2_load.log -
  node $NODES

```

Using a YAML file to run cassandra-stress

This example uses a YAML file for the keyspace and table definitions, as well as query definition. The keyspace name and definition are the first entries in the YAML file:

```

keyspace: stresscql
#
# The CQL for creating a keyspace (optional if it already exists)
#
keyspace_definition: |
  CREATE KEYSPACE stresscql WITH replication = {'class': 'SimpleStrategy',
    'repl
    ication_factor': 1};

```

The table name and definition are the next entries in the YAML file. The table definition is created using CQL:

```
table: typestest

#
# The CQL for creating a table you wish to stress (optional if it already
# exists
#
table_definition: |
    CREATE TABLE typestest (
        name text,
        choice boolean,
        date timestamp,
        address inet,
        dbl double,
        lval bigint,
        ival int,
        uid timeuuid,
        value blob,
        PRIMARY KEY((name,choice), date, address, dbl, lval, ival, uid)
    )
    WITH compaction = { 'class': 'LeveledCompactionStrategy' }
# AND compression = { 'sstable_compression' : '' }
# AND comment='A table of many types to test wide rows'
```

The population distribution can be defined for any column in the table. A number of distribution options are available. In this example, `name` is set to create a uniform distribution over 10 values and `lval` is set to a Gaussian distribution. The `date` field is set to have a uniform distribution between 20 and 40 for the entire Cassandra cluster.:

```
columnspec:
- name: name
  size: uniform(1..10)
  population: uniform(1..10)      # range of unique values to select for
  name (default is 100B)
- name: date
  cluster: uniform(20..40)
- name: lval
  population: gaussian(1..1000)
  cluster: uniform(1..4)
```

The query that will be run against the defined table, `simple1` for read operations, is defined at the end of the YAML file.

```
queries:
  simple1:
    cql: select * from typestest where name = ? and choice = ? LIMIT 100
    fields: samerow      # samerow or multirow (select arguments from the
    same row, or randomly
    from all rows in the partition)
```

The command specifies the YAML file, `cqlstress-example.yaml`, that is used to run the tests:

```
$ cassandra-stress user profile=tools/cqlstress-example.yaml ops\(simple1=1\)
no-warmup cl=QUORUM
```

Note: Use escaping backslashes for denoting the `ops` value.

The `simple1` operation will be completed once, no warmup is specified, and the consistency level is set to `QUORUM`.

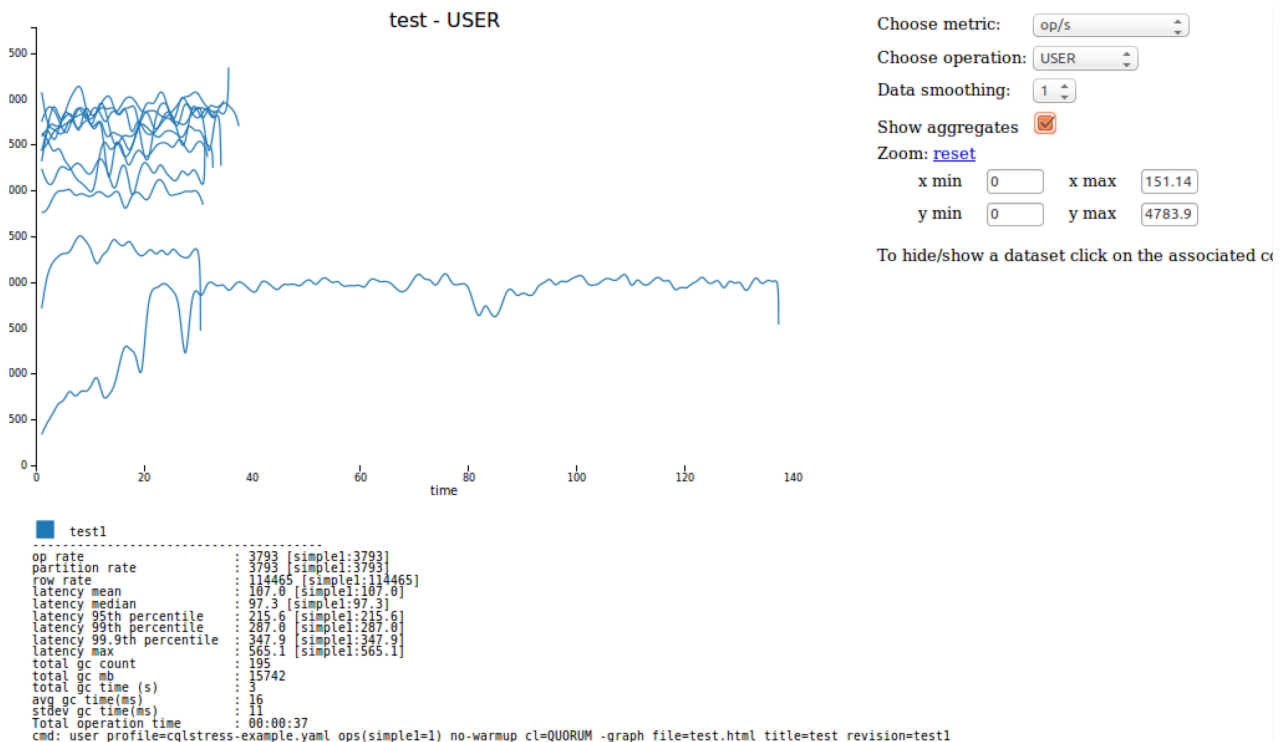
For a complete description on using these YAML files for `cassandra-stress`, see [Improved Cassandra 2.1 Stress Tool: Benchmark Any Schema – Part 1](#).

Using the -graph option

In Cassandra 3.2 and later, the `-graph` option provides visual feedback for `cassandra-stress` tests. A file must be named to build the resulting HTML file. A `title` and `revision` are optional, but `revision` must be used if multiple stress tests are graphed on the same output.

```
$ cassandra-stress user profile=tools/cqlstress-example.yaml ops\ (insert=1\ ) -graph file=test.html title=test revision=test1
```

An interactive graph can be displayed with a web browser:



Using the Daemon Mode

The daemon is only available in tarball installations. Run the daemon from:

```
$ install_location/tools/bin/cassandra-stressd start|stop|status [-h <host>]
```

During stress testing, you can keep the daemon running and send it commands using the `-sendto` option.

Contrast using `cassandra-stress` with and without daemon

- Insert 10,000,000 rows across two nodes:

```
$ tools/bin/cassandra-stress -node 192.168.1.101,192.168.1.102 n=10000000
```

- Insert 10,000,000 rows across two nodes using the daemon mode:

```
$ tools/bin/cassandra-stress -node 192.168.1.101,192.168.1.102 n=10000000 -
sendto 54.0.0.1
```

Interpreting the output of cassandra-stress

Each line reports data for the interval between the last elapsed time and current elapsed time.

```
Created keyspaces. Sleeping 1s for propagation.
Warming up WRITE with 50000 iterations...
INFO 23:57:05 Using data-center name 'datacenter1' for
  DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct
  datacenter name with DCAwareRoundRobinPolicy constructor)
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: localhost/127.0.0.1; Rack: rack1
INFO 23:57:05 New Cassandra host localhost/127.0.0.1:9042 added
Sleeping 2s...
WARNING: uncertainty mode (err<) results in uneven workload between thread
runs, so should be used for high level analysis only
Running with 4 threadCount
Running WRITE with 4 threads until stderr of mean < 0.02
total ops , adj row/s,    op/s,    pk/s,    row/s,    mean,    med,    .95,
    .99,    .999,    max,    time,    stderr, gc: #,    max ms,    sum ms,    sdv
ms,      mb
2552      ,      2553,    2553,    2553,    2553,    1.5,    1.4,    2.5,
6.0,      12.6,    18.0,    1.0,    0.00000,    0,    0,    0,
0,      0
5173      ,      2634,    2613,    2613,    2613,    1.5,    1.5,    1.8,
2.6,      8.6,    9.2,    2.0,    0.00000,    0,    0,    0,
0,      0
...

Results:
op rate           : 3954
partition rate    : 3954
row rate          : 3954
latency mean      : 1.0
latency median    : 0.8
latency 95th percentile : 1.5
latency 99th percentile : 1.8
latency 99.9th percentile : 2.2
latency max       : 73.6
total gc count     : 25
total gc mb        : 1826
total gc time (s)  : 1
avg gc time(ms)    : 37
stdev gc time(ms)  : 10
Total operation time : 00:00:59
Sleeping for 15s
Running with 4 threadCount
```

Table: Output of cassandra-stress

Data	Description
total ops	Running total number of operations during the run.
op/s	Number of operations per second performed during the run.
pk/s	Number of partition operations per second performed during the run.

Data	Description
row/s	Number of row operations per second performed during the run.
mean	Average latency in milliseconds for each operation during that run.
med	Median latency in milliseconds for each operation during that run.
.95	95% of the time the latency was less than the number displayed in the column.
.99	99% of the time the latency was less than the number displayed in the column.
.999	99.9% of the time the latency was less than the number displayed in the column.
max	Maximum latency in milliseconds.
time	Total operation time.
stderr	Standard error of the mean. It is a measure of confidence in the average throughput number; the smaller the number, the more accurate the measure of the cluster's performance.
gc: #	Number of garbage collections.
max ms	Longest garbage collection in milliseconds.
sum ms	Total of garbage collection in milliseconds.
sdv ms	Standard deviation in milliseconds.
mb	Size of the garbage collection in megabytes.

SSTable utilities

sstableexpiredblockers

During compaction, Cassandra can drop entire SSTables if they contain only expired tombstones and if it is guaranteed to not cover any data in other SSTables. This diagnostic tool outputs all SSTables that are blocking other SSTables from being dropped.

Usage:

- Package installations: `$ sstableexpiredblockers [--dry-run] keyspace table`
- Tarball installations:

```
$ cd install_location/tools
$ bin/sstableexpiredblockers [--dry-run] keyspace table
```

Procedure

Choose a keyspace and table to check for any SSTables that are blocking the specified table from dropping.

```
$ sstableexpiredblockers cycling cyclist_name
```

sstablekeys

The `sstablekeys` utility dumps table keys.

Usage:

- Package installations: `$ sstablekeys sstable_name`
- Tarball installations:

```
$ cd install_location/tools
$ bin/sstablekeys sstable_name
```

Procedure

1. If data has not been previously flushed to disk, manually flush it. For example:

```
$ nodetool flush cycling cyclist_name
```

2. To list the keys in an SSTable, find the name of the SSTable file.

The file is located in the `data` directory and has a `.db` extension.

- Package installations: `/var/lib/cassandra/data`
- Tarball installations: `install_location/data/data`
- Windows installations:
- Windows installations:

3. Look at keys in the SSTable data. For example, use `sstablekeys` followed by the path to the data. Use the path to data for your Cassandra installation:

```
## Package installations
$ sstablekeys /var/lib/cassandra/data/cycling/cyclist_name-
a882dca02aaf11e58c7b8b496c707234/1a-1-big-Data.db

## Tarball installations
$ sstablekeys install_location/data/data/cycling/cyclist_name-
a882dca02aaf11e58c7b8b496c707234/1a-1-big-Data.db
```

The output appears, for example:

```
e7ae5cf3-d358-4d99-b900-85902fda9bb0
5b6962dd-3f90-4c93-8f61-eabfa4a803e2
220844bf-4860-49d6-9a4b-6b5d3a79cbfb
6ab09bec-e68e-48d9-a5f8-97e6fb4c9b47
e7cd5752-bc0d-4157-a80f-7523add8dbcd
```

sstablelevelreset

Reset level to 0 on a given set of SSTables.

Usage:

- Package installations: `$ sstablelevelreset [--really-reset] keyspace table`

- Tarball installations:

```
$ cd install_location/tools
$ bin/sstablelevelreset [--really-reset] keyspace table
```

Procedure

Choose a keyspace and table to reset to level 0.

```
$ C:\> %CASSANDRA_HOME%\tools\bin\sstablelevelreset --really-reset cycling
cyclist_name
```

sstableloader (Cassandra bulk loader)

The Cassandra bulk loader, also called the sstableloader, provides the ability to:

- Bulk load external data into a cluster.
- Load existing SSTables into another cluster with a different number of nodes or replication strategy.
- Restore snapshots.

The sstableloader streams a set of SSTable data files to a live cluster; it does not simply copy the set of SSTables to every node, but transfers the relevant part of the data to each node, conforming to the replication strategy of the cluster. The table into which the data is loaded does not need to be empty.

If tables are repaired in a different cluster, after being loaded, the tables are not repaired.

Prerequisites

Because sstableloader uses Cassandra gossip, make sure of the following:

- The `cassandra.yaml` configuration file is in the classpath and properly configured.
- At least one node in the cluster is configured as seed.
- In the `cassandra.yaml` file, the following properties are properly configured for the cluster that you are importing into:
 - `cluster_name`
 - `listen_address`
 - `storage_port`
 - `rpc_address`
 - `rpc_port`

When using sstableloader to load external data, you must first generate SSTables.

If using DataStax Enterprise, you can use [Sqoop](#) to migrate external data to Cassandra.

Generating SSTables

SSTableWriter is the API to create raw Cassandra data files locally for bulk load into your cluster. The Cassandra source code includes the `CQLSSTableWriter` implementation for creating SSTable files from external data without needing to understand the details of how those map to the underlying storage engine. Import the `org.apache.cassandra.io.sstable.CQLSSTableWriter` class, and define the schema for the data you want to import, a writer for the schema, and a prepared insert statement, as shown in [Cassandra 2.0.1](#), [2.0.2](#), and [a quick peek at 2.0.3](#).

Using sstableloader

Before loading the data, you must define the schema of the tables with [CQL](#) or Thrift.

To get the best throughput from SSTable loading, you can use multiple instances of sstableloader to stream across multiple machines. No hard limit exists on the number of SSTables that sstableloader can run at the same time, so you can add additional loaders until you see no further improvement.

If you use `sstableloader` on the same machine as the Cassandra node, you can't use the same network interface as the Cassandra node. However, you can use the JMX **StorageService** > **bulkload()** call from that node. This method takes the absolute path to the directory where the SSTables are located, and loads them just as `sstableloader` does. However, because the node is both source and destination for the streaming, it increases the load on that node. This means that you should load data from machines that are not Cassandra nodes when loading into a live cluster.

Usage:

Package installations:

```
$ sstableloader [options] path_to_keyspace
```

Tarball installations:

```
$ cd install_location/bin
$ sstableloader [options] path_to_keyspace
```

The `sstableloader` bulk loads the SSTables found in the keyspace directory to the configured target cluster, where the parent directories of the directory path are used as the target keyspace/table name.

1. Go to the location of the SSTables:

Package installations:

```
$ cd /var/lib/cassandra/data/Keyspace1/Standard1/
```

Tarball installations:

```
$ cd install_location/data/data/Keyspace1/Standard1/
```

2. To view the contents of the keyspace:

```
$ ls
Keyspace1-Standard1-jb-60-CRC.db
Keyspace1-Standard1-jb-60-Data.db
...
Keyspace1-Standard1-jb-60-TOC.txt
```

3. To bulk load the files, specify the path to `Keyspace1/Standard1/` in the target cluster:

```
$ sstableloader -d 110.82.155.1 /var/lib/cassandra/data/Keyspace1/Standard1/
## Package installation

$ install_location/bin/sstableloader -d 110.82.155.1 /var/lib/cassandra/
data/data/Keyspace1/Standard1/ ## Tarball installation
```

This bulk loads all files.

Table: sstableloader options

Short option	Long option	Description
-alg	--ssl-alg <ALGORITHM>	Client SSL algorithm (default: SunX509).
-ap	--auth-provider <auth provider class name>	Allows the use of a third party auth provider. Can be combined with -u <username> and -pw <password> if the auth provider supports plain text credentials.
-ciphers	--ssl-ciphers <CIPHER-SUITES>	Client SSL. Comma-separated list of encryption suites.
-cph	--connections-per-host <connectionsPerHost>	Number of concurrent connections-per-host.

Short option	Long option	Description
-d	--nodes <initial_hosts>	Required. Connect to a list of (comma separated) hosts for initial cluster information.
-f	--conf-path <path_to_config_file>	Path to the <code>cassandra.yaml</code> path for streaming throughput and client/server SSL.
-h	--help	Display help.
-i	--ignore <NODES>	Do not stream to this comma separated list of nodes.
-ks	--keystore <KEYSTORE>	Client SSL. Full path to the keystore.
-kspw	--keystore-password <KEYSTORE-PASSWORD>	Client SSL. Password for the keystore.
	--no-progress	Do not display progress.
-p	--port <rpc port>	RPC port (default: 9160 [Thrift]).
-prtcl	--ssl-protocol <PROTOCOL>	Client SSL. Connections protocol to use (default: TLS).
-pw	--password <password>	Password for Cassandra authentication.
-st	--store-type <STORE- TYPE>	Client SSL. Type of store.
-t	--throttle <throttle>	Throttle speed in Mbits (default: unlimited).
-tf	--transport-factory <transport factory>	Fully-qualified <code>ITransportFactory</code> class name for creating a connection to Cassandra.
-ts	--truststore <TRUSTSTORE>	Client SSL. Full path to truststore.
-tspw	--truststore-password <TRUSTSTORE- PASSWORD>	Client SSL. Password of the truststore.
-u	--username <username>	User name for Cassandra authentication.
-v	--verbose	Verbose output.

The following `cassandra.yaml` options can be over-ridden from the command line:

Option in <code>cassandra.yaml</code>	Command line example
<code>stream_throughput_outbound_megabits_per_sec</code>	<code>--throttle 300</code>
<code>server_encryption_options</code>	<code>--ssl-protocol none</code>
<code>client_encryption_options</code>	<code>--keystore-password MyPassword</code>

sstablemetadata

The `sstablemetadata` utility prints metadata about a specified SSTable. The utility displays metadata that includes:

- sstable name
- partitioner
- RepairedAt timestamp (for incremental repairs only)

- sstable level (for Leveled Compaction only)
- number of tombstones and Dropped timestamps (in epoch time)
- number of cells and size (in bytes) per row

Such data can be useful for troubleshooting wide rows or performance degrading tombstones.

Procedure

Run the `sstablemetadata` command. Multiple SSTable filenames can be included in the command.

```
$ bin/sstablemetadata <sstable_name filenames>
```

```
tools/bin/sstablemetadata data/data/autogeneratedtest/
transaction_by_retailer-f27e4d5078dc11e59d629d03f52e8a2b/ma-203-big-Data.db
SSTable: data/data/autogeneratedtest/transaction_by_retailer-
f27e4d5078dc11e59d629d03f52e8a2b/ma-203-big
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Bloom Filter FP chance: 0.010000
Minimum timestamp: 1445871871053006
Maximum timestamp: 1445871953354005
SSTable max local deletion time: 2147483647
Compression ratio: -1.0
Estimated droppable tombstones: 0.0
SSTable Level: 0
Repaired at: 0
ReplayPosition(segmentId=1445871179392, position=18397674)
Estimated tombstone drop times:
2147483647: 7816721
```

Count	Row Size	Cell Count
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
10	0	0
12	0	710611
14	0	0
17	0	0
20	0	0
24	0	0
29	0	0
35	0	0
42	0	0
50	0	0
60	0	0
72	0	0
86	0	0
103	0	0
124	0	0
149	0	0
179	0	0
215	0	0
258	0	0
310	81	0
372	710530	0
446	0	0
535	0	0
642	0	0
770	0	0

924	0	0
1109	0	0
1331	0	0
1597	0	0
1916	0	0
2299	0	0
2759	0	0
3311	0	0
3973	0	0
4768	0	0
5722	0	0
6866	0	0
8239	0	0
9887	0	0
11864	0	0
14237	0	0
17084	0	0
20501	0	0
24601	0	0
29521	0	0
35425	0	0
42510	0	0
51012	0	0
61214	0	0
73457	0	0
88148	0	0
105778	0	0
126934	0	0
152321	0	0
182785	0	0
219342	0	0
263210	0	0
315852	0	0
379022	0	0
454826	0	0
545791	0	0
654949	0	0
785939	0	0
943127	0	0
1131752	0	0
1358102	0	0
1629722	0	0
1955666	0	0
2346799	0	0
2816159	0	0
3379391	0	0
4055269	0	0
4866323	0	0
5839588	0	0
7007506	0	0
8409007	0	0
10090808	0	0
12108970	0	0
14530764	0	0
17436917	0	0
20924300	0	0
25109160	0	0
30130992	0	0
36157190	0	0
43388628	0	0
52066354	0	0
62479625	0	0
74975550	0	0
89970660	0	0

```

107964792      0      0
129557750      0      0
155469300      0      0
186563160      0      0
223875792      0      0
268650950      0      0
322381140      0      0
386857368      0      0
464228842      0      0
557074610      0      0
668489532      0      0
802187438      0      0
962624926      0      0
1155149911     0      0
1386179893     0      0
1663415872     0      0
1996099046     0      0
2395318855     0      0
2874382626     0
3449259151     0
4139110981     0
4966933177     0
5960319812     0
7152383774     0
8582860529     0
10299432635    0
12359319162    0
14831182994    0
17797419593    0
21356903512    0
25628284214    0
30753941057    0
36904729268    0
44285675122    0
53142810146    0
63771372175    0
76525646610    0
91830775932    0
110196931118   0
132236317342   0
158683580810   0
190420296972   0
228504356366   0
274205227639   0
329046273167   0
394855527800   0
473826633360   0
568591960032   0
682310352038   0
818772422446   0
982526906935   0
1179032288322   0
1414838745986   0
Estimated cardinality: 722835

```

sstableofflinerelevel

This tool is intended to run in an offline fashion. When using the `LevelledCompactionStrategy`, it is possible for the number of SSTables in level L0 to become excessively large, resulting in read latency degrading. This is often the case when atypical write load is experienced (eg. bulk import of data, node

bootstrapping). This tool will relevel the SSTables in an optimal fashion. The `--dry-run` flag can be used to run in test mode and examine the tools results.

Usage:

- Package installations: `$ sstableofflinerelevel [--dry-run] keyspace table`
- Tarball installations:

```
$ cd install_location/tools
$ bin/sstableofflinerelevel [--dry-run] keyspace table
```

Procedure

Choose a keyspace and table to relevel.

```
$ sstableofflinerelevel cycling cyclist_name
```

sstablerepairedset

This tool is intended to mark specific SSTables as repaired or unrepaired. It is used to set the `repairedAt` status on a given set of SSTables. This metadata facilitates incremental repairs. It can take in the path to an individual sstable or the path to a file containing a list of SSTables paths.

Warning: This command should only be run with Cassandra stopped.

Usage:

- Package installations: `$ sstablerepairedset [--is-repaired | --is-unrepaired] [-f sstable-list | sstables]`
- Tarball installations:

```
$ cd install_location/tools
$ bin/sstablerepairedset [--is-repaired | --is-unrepaired] [-f sstable-list
| sstables]
```

Procedure

- Choose SSTables to mark as repaired.

```
$ sstablerepairedset --is-repaired /var/lib/cassandra/data/cycling/
cyclist_name-a882dca02aaf11e58c7b8b496c707234/1a-1-big-Data.db
```

- Use a file to list the SSTable to mark as unrepaired.

```
$ sstablerepairedset --is-unrepaired -f repairSetSSTables.txt
```

An example file includes the path to the `Data.db` files:

```
/home/user/apache-cassandra-3.0.0/data/data/test/
cyclist_by_country-82246fc065ff11e5a4c58b496c707234/ma-1-big-Data.db
/home/user/apache-cassandra-3.0.0/data/data/test/
cyclist_by_birthday-8248246065ff11e5a4c58b496c707234/ma-1-big-Data.db
/home/user/apache-cassandra-3.0.0/data/data/test/
cyclist_by_birthday-8248246065ff11e5a4c58b496c707234/ma-2-big-Data.db
/home/user/apache-cassandra-3.0.0/data/data/test/
cyclist_by_age-8201305065ff11e5a4c58b496c707234/ma-1-big-Data.db
/home/user/apache-cassandra-3.0.0/data/data/test/
cyclist_by_age-8201305065ff11e5a4c58b496c707234/ma-2-big-Data.db
```

A command to use to list all the `Data.db` files in a keyspace is the following:

```
find '/home/user/apache-cassandra-3.0.0/data/data/test/' -iname "*Data.db*"
```

sstablescrib

The `sstablescrib` utility is an offline version of `nodetool scrub`. It attempts to remove the corrupted parts while preserving non-corrupted data. Because `sstablescrib` runs offline, it can correct errors that `nodetool scrub` cannot. If an SSTable cannot be read due to corruption, it will be left on disk.

If scrubbing results in dropping rows, new SSTables become unrepaired. However, if no bad rows are detected, the SSTable keeps its original `repairedAt` field, which denotes the time of the repair.

Procedure

1. Before using `sstablescrib`, try rebuilding the tables using `nodetool scrub`.

If `nodetool scrub` does not fix the problem, use this utility.

2. [Shut down the node](#).

3. Run the utility:

- Package installations:

```
$ sstablescrib [options] keyspace table
```

- Tarball installations:

```
$ cd install_location
$ bin/sstablescrib [options] keyspace table
```

Table: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.
-m	--manifest-check	Only check and repair the leveled manifest, without actually scrubbing the SSTables.
-s	--skip-corrupted	Skip corrupt rows in counter tables.
-v	--verbose	Verbose output.

sstablesplit

Use this tool to split SSTables files into multiple SSTables of a maximum designated size. For example, if `SizeTieredCompactionStrategy` was used for a major compaction and results in a excessively large SSTable, it's a good idea to split the table because won't get compacted again until the next huge compaction.

Cassandra must be stopped to use this tool:

- Package installations:

```
$ sudo service cassandra stop
```

- Tarball installations:

```
$ ps aux | grep cassandra
$ sudo kill pid
```

Usage:

- Package installations: `$ sstablesplit [options] <filename> [<filename>]*`
- Tarball installations:

```
$ cd install_location/tools/bin
sstablesplit [options] <filename> [<filename>]*
```

Example:

```
$ sstablesplit -s 40 /var/lib/cassandra/data/data/Keyspace1/Standard1/*
```

Table: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.
	--no-snapshot	Do not snapshot the SSTables before splitting.
-s	--size <size>	Maximum size in MB for the output SSTables (default: 50).
-v	--verbose	Verbose output.

sstableupgrade

This tool rewrites the SSTables in the specified table to match the currently installed version of Cassandra.

If restoring with [sstableloader](#), you must upgrade your snapshots before restoring for any snapshot taken in a major version older than the major version that Cassandra is currently running.

Usage:

- Package installations:

```
$ sstableupgrade [options] keyspace table [snapshot]
```

- Tarball installations:

```
$ cd install_location
$ bin/sstableupgrade [options] keyspace table [snapshot]
```

The snapshot option only upgrades the specified snapshot.

Table: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.

sstableutil

The sstableutil will list the SSTable files for a provided table.

Usage:

- Package installations: `$ sstableutil [--cleanup | --debug | --help | --opslog | --type <arg> | --verbose] keypace | table`
- Tarball installations:

```
$ cd install_location$ bin/sstableutil [--cleanup | --debug | --help | --opslog | --type <arg> | --verbose] keypace | table
```

Note: Arguments for `--type` option are: `all`, `tmp`, or `final`.

Procedure

Choose a table for which to list SSTables files.

```
$ sstableutil --all cycling cyclist_name
```

sstableverify

The `sstableverify` utility will verify the SSTable for a provided table and look for errors or data corruption.

Usage:

- Package installations: `$ sstableverify [--debug | --extended | --help | --verbose] keypace | table`
- Tarball installations:

```
$ cd install_location$ bin/sstableverify [--debug | --extended | --help | --verbose] keypace | table
```

Procedure

Choose a table to verify.

```
$ sstableverify --verbose cycling cyclist_name
```

Troubleshooting

Peculiar Linux kernel performance problem on NUMA systems

Problems due to `zone_reclaim_mode`.

The Linux kernel can be inconsistent in enabling/disabling `zone_reclaim_mode`. This can result in odd performance problems:

- Random huge CPU spikes resulting in large increases in latency and throughput.
- Programs hanging indefinitely apparently doing nothing.
- Symptoms appearing and disappearing suddenly.
- After a reboot, the symptoms generally do not show again for some time.

To ensure that `zone_reclaim_mode` is disabled:

```
$ echo 0 > /proc/sys/vm/zone_reclaim_mode
```

Reads are getting slower while writes are still fast

The cluster's IO capacity is not enough to handle the write load it is receiving.

Check the SSTable counts in [tablestats](#). If the count is continually growing, the cluster's IO capacity is not enough to handle the write load it is receiving. Reads have slowed down because the data is fragmented across many SSTables and compaction is continually running trying to reduce them. Adding more IO capacity, either via more machines in the cluster, or faster drives such as SSDs, will be necessary to solve this.

If the SSTable count is relatively low (32 or less) then the amount of file cache available per machine compared to the amount of data per machine needs to be considered, as well as the application's read pattern. The amount of file cache can be formulated as $(TotalMemory - JVMHeapSize)$ and if the amount of data is greater and the read pattern is approximately random, an equal ratio of reads to the cache:data ratio will need to seek the disk. With spinning media, this is a slow operation. You may be able to mitigate many of the seeks by using a key cache of 100%, and a small amount of row cache (10000-20000) if you have some hot rows and they are not extremely large.

Nodes seem to freeze after some period of time

Some portion of the JVM is being swapped out by the operating system (OS).

Check your `system.log` for messages from the `GCInspector`. If the `GCInspector` is indicating that either the `ParNew` or `ConcurrentMarkSweep` collectors took longer than 15 seconds, there is a high probability that some portion of the JVM is being swapped out by the OS.

DataStax strongly recommends that you disable swap entirely (`sudo swapoff --all`). Because Cassandra has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks. To make this change permanent, remove all swap file entries from `/etc/fstab`.

If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

If the `GCInspector` isn't reporting very long GC times, but is reporting moderate times frequently (`ConcurrentMarkSweep` taking a few seconds very often) then it is likely that the JVM is experiencing extreme GC pressure and will eventually OOM. See [Nodes are dying with OOM errors](#) on page 256.

Nodes are dying with OOM errors

Nodes are dying with `OutOfMemory` exceptions.

Check for these typical causes:

Row cache is too large, or is caching large rows

Row cache is generally a high-end optimization. Try disabling it and see if the OOM problems continue.

The memtable sizes are too large for the amount of heap allocated to the JVM

You can expect $N + 2$ memtables resident in memory, where N is the number of tables. Adding another 1GB on top of that for Cassandra itself is a good estimate of total heap usage.

If none of these seem to apply to your situation, try loading the heap dump in [MAT](#) and see which class is consuming the bulk of the heap for clues.

Nodetool or JMX connections failing on remote nodes

Nodetool commands can be run locally but not on other nodes in the cluster.

If you can run nodetool commands locally but not on other nodes in the ring, you may have a common JMX connection problem that is resolved by adding an entry like the following in `cassandra-env.sh` on each node:

```
JVM_OPTS = "$JVM_OPTS -Djava.rmi.server.hostname=public name"
```

If you still cannot run nodetool commands remotely after making this configuration change, do a full evaluation of your firewall and network security. The nodetool utility communicates through JMX on port 7199.

Handling schema disagreements

Check for and resolve schema disagreements.

In the event that a schema disagreement occurs, check for and resolve schema disagreements as follows:

Procedure

1. Run the `nodetool describcluster` command.

```
$ nodetool describcluster
```

If any node is UNREACHABLE, you see output something like this:

```
$ nodetool describcluster
Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
  UNREACHABLE: 1176b7ac-8993-395d-85fd-41b89ef49fbb: [10.202.205.203]
                9b861925-1a19-057c-ff70-779273e95aa6: [10.80.207.102]
                8613985e-c49e-b8f7-57ae-6439e879bb2a: [10.116.138.23]
```

2. Restart unreachable nodes.
3. Repeat steps 1 and 2 until `nodetool describcluster` shows that all nodes have the same schema version number#only one schema version appears in the output.

View of ring differs between some nodes

Indicates that the ring is in a bad state.

This situation can happen when not using virtual nodes (vnodes) and there are token conflicts (for instance, when bootstrapping two nodes simultaneously with automatic token selection.) Unfortunately, the only way to resolve this is to do a full cluster restart. A rolling restart is insufficient since gossip from nodes with the bad state will repopulate it on newly booted nodes.

Java reports an error saying there are too many open files

Java may not have open enough file descriptors.

Cassandra generally needs more than the default (1024) amount of file descriptors. To increase the number of file descriptors, change the security limits on your Cassandra nodes as described in the [Recommended Settings](#) section of [Insufficient user resource limits errors](#) on page 258.

Another, much less likely possibility, is a file descriptor leak in Cassandra. Run `lsuf -n | grep java` to check that the number of file descriptors opened by Java is reasonable and reports the error if the number is greater than a few thousand.

Insufficient user resource limits errors

Insufficient resource limits may result in a number of errors in Cassandra.

Cassandra errors

Insufficient as (address space) or memlock setting

```
ERROR [SSTableBatchOpen:1 ] 2012-07-25 15:46:02,913
AbstractCassandraDaemon.java (line 139)
Fatal exception in thread Thread [SSTableBatchOpen:1,5,main ]
java.io.IOException: java.io.IOException: Map failed at ...
```

Insufficient memlock settings

```
WARN [main ] 2011-06-15 09:58:56,861 CLibrary.java (line 118) Unable to lock
JVM memory (ENOMEM).
This can result in part of the JVM being swapped out, especially with mmaped
I/O enabled.
Increase RLIMIT_MEMLOCK or run Cassandra as root.
```

Insufficient nofiles setting

```
WARN 05:13:43,644 Transport error occurred during acceptance of message.
org.apache.thrift.transport.TTransportException: java.net.SocketException:
Too many open files ...
```

Insufficient nofiles setting

```
ERROR [MutationStage:11 ] 2012-04-30 09:46:08,102 AbstractCassandraDaemon.java
(line 139)
Fatal exception in thread Thread [MutationStage:11,5,main ]
java.lang.OutOfMemoryError: unable to create new native thread
```

Recommended settings

You can view the current limits using the `ulimit -a` command. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Packaged installs: Ensure that the following settings are included in the `/etc/security/limits.d/cassandra.conf` file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

Tarball installs: Ensure that the following settings are included in the `/etc/security/limits.conf` file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

If you run Cassandra as root, some Linux distributions such as Ubuntu, require setting the limits for root explicitly instead of using `<cassandra_user>`:

```
root - memlock unlimited
root - nofile 100000
root - nproc 32768
root - as unlimited
```

For CentOS, RHEL, OEL systems, also set the nproc limits in `/etc/security/limits.d/90-nproc.conf`:

```
<cassandra_user> - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 131072
```

To make the changes take effect, reboot the server or run the following command:

```
$ sudo sysctl -p
```

To confirm the limits are applied to the Cassandra process, run the following command where *pid* is the process ID of the currently running Cassandra process:

```
$ cat /proc/<pid>/limits
```

Cannot initialize class org.xerial.snappy.Snappy

An error may occur when Snappy compression/decompression is enabled although its library is available from the classpath.

```
java.util.concurrent.ExecutionException: java.lang.NoClassDefFoundError:
    Could not initialize class org.xerial.snappy.Snappy
...
Caused by: java.lang.NoClassDefFoundError: Could not initialize class
    org.xerial.snappy.Snappy
    at
    org.apache.cassandra.io.compress.SnappyCompressor.initialCompressedBufferLength
        (SnappyCompressor.java:39)
```

The native library `snappy-1.0.4.1-libsnapappyjava.so` for Snappy compression is included in the `snappy-java-1.0.4.1.jar` file. When the JVM initializes the JAR, the library is added to the default temp directory. If the default temp directory is mounted with a `noexec` option, it results in the above exception.

One solution is to specify a different temp directory that has already been mounted without the `noexec` option, as follows:

- If you use the DSE/Cassandra command `$_BIN/dse cassandra` or `$_BIN/cassandra`, simply append the command line:
 - DSE: `bin/dse cassandra -t -Dorg.xerial.snappy.tmpdir=/path/to/newtmp`
 - Cassandra: `bin/cassandra -Dorg.xerial.snappy.tmpdir=/path/to/newtmp`
- If starting from a package using `service dse start` or `service cassandra start`, add a system environment variable `JVM_OPTS` with the value:

```
JVM_OPTS=-Dorg.xerial.snappy.tmpdir=/path/to/newtmp
```

The default `cassandra-env.sh` looks for the variable and appends to it when starting the JVM.

Firewall idle connection timeout causing nodes to lose communication during low traffic times

During low traffic intervals, a firewall configured with an idle connection timeout can close connections to local nodes and nodes in other data centers. The default idle connection timeout is usually 60 minutes and configurable by the network administrator.

Procedure

To prevent connections between nodes from timing out, set the TCP keep alive variables:

1. Get a list of available kernel variables:

```
$ sysctl -A | grep net.ipv4
```

The following variables should exist:

- *net.ipv4.tcp_keepalive_time*
Time of connection inactivity after which the first keep alive request is sent.
- *net.ipv4.tcp_keepalive_probes*
Number of keep alive requests retransmitted before the connection is considered broken.
- *net.ipv4.tcp_keepalive_intvl*
Time interval between keep alive probes.

2. To change these settings:

```
$ sudo sysctl -w net.ipv4.tcp_keepalive_time=60  
net.ipv4.tcp_keepalive_probes=3 net.ipv4.tcp_keepalive_intvl=10
```

This sample command changes TCP keepalive timeout to 60 seconds with 3 probes, 10 seconds gap between each. This setting detects dead TCP connections after 90 seconds (60 + 10 + 10 + 10). There is no need to be concerned about the additional traffic as it's negligible and permanently leaving these settings shouldn't be an issue.

DataStax Distribution of Apache Cassandra 3.x release notes

Note: Cassandra is now releasing on a tick-tock schedule. For more information, see [Cassandra 2.2](#), [3.0](#), and [beyond](#).

The latest version of DataStax Distribution of Apache Cassandra 3.x is 3.2.

The [CHANGES.txt](#) describes the changes in detail. You can view all version changes by branch or tag in the drop-down list on the changes page.







New features, improvements, and notable changes are described in [What's new?](#).

Tips for using DataStax documentation




Navigating the documents

To navigate, use the Contents pane or search. Additional controls are:

Toolbar icons

	Go back through the topics as listed in the Contents pane.
	Go forward through the topics as listed in the Contents pane.
	See doc tweets and provide feedback.
	Display PDF version.
	Send email to DataStax docs.
	Print page.

Contents, bookmarking, and legend icons

	Opens Contents items. Also expands and collapses text, such as Synopsis and Nodetool legends.
	Closes the Contents items.
	Appears on headings for bookmarking. Right-click to get the link.

Searching documents

Search is designed for each product guide. For example, if searching in DataStax Enterprise 4.8, the results include topics from DataStax Enterprise 4.8, Cassandra 2.1, and CQL 3.1. The results are displayed in tabs:

×

All
DataStax Enterprise
Cassandra
CQL

About 6,800 results (0.38 seconds)
Sort by: Relevance ▾

[Initializing a multiple node cluster \(single data center\)](https://docs.datastax.com/en/cassandra/2.1/.../initializeSingleDS.html)
docs.datastax.com/en/cassandra/2.1/.../initializeSingleDS.html
A deployment scenario for a Cassandra **cluster** with a single data center. | Version 2.1.
Labeled Cassandra

Other resources

You can find more information and help at:

- [Documentation home page](#)
- [DataStax Academy](#)
- [Datasheets](#)
- [Webinars](#)
- [Whitepapers](#)
- [DataStax Developer Blogs](#)
- [Support](#)