Contents

Chapter 1. About Cassandra.................................................................................................................. 11
Chapter 2. What’s new............................................................................................................................ 12
Chapter 3. CQL......................................................................................................................................... 14
Chapter 4. Understanding the architecture........................................................................................... 15
  Architecture in brief............................................................................................................................... 15
  Internode communications (gossip)...................................................................................................... 17
    Failure detection and recovery........................................................................................................... 17
  Data distribution and replication........................................................................................................... 18
    Consistent hashing............................................................................................................................. 18
    Virtual nodes..................................................................................................................................... 19
  Data replication.................................................................................................................................... 20
  Partitioners........................................................................................................................................... 21
    Murmur3Partitioner.............................................................................................................................. 22
    RandomPartitioner.............................................................................................................................. 22
    ByteOrderedPartitioner....................................................................................................................... 22
  Snitches................................................................................................................................................ 23
    Dynamic snitching............................................................................................................................... 23
    SimpleSnitch....................................................................................................................................... 23
    RackInferringSnitch.............................................................................................................................. 23
    PropertyFileSnitch............................................................................................................................... 24
    GossippingPropertyFileSnitch............................................................................................................. 24
    Ec2Snitch........................................................................................................................................... 25
    Ec2MultiRegionSnitch......................................................................................................................... 26
    GoogleCloudSnitch.............................................................................................................................. 27
    CloudstackSnitch................................................................................................................................. 28
  Client requests...................................................................................................................................... 28
Chapter 5. Planning a deployment.......................................................................................................... 29
Chapter 6. Installing................................................................................................................................. 30
  Installing on RHEL-based systems........................................................................................................ 30
  Installing on Debian-based systems...................................................................................................... 31
  Installing from the binary tarball.......................................................................................................... 33
  Recommended settings........................................................................................................................... 35
## Chapter 7. Initializing a cluster

- Initializing a multiple node cluster (single datacenter).......................... 41
- Initializing a multiple node cluster (multiple datacenters)........................ 44

## Chapter 8. Security

- Securing Cassandra................................................................. 48
  - SSL encryption....................................................................... 48
    - Preparing server certificates............................................... 48
    - Client-to-node encryption................................................. 50
    - Using cqlsh with SSL encryption........................................ 50
    - Using nodetool (JMX) with SSL.......................................... 51
    - Node-to-node encryption................................................... 52
  - Internal authentication....................................................... 53
    - Internal authentication.................................................... 53
    - Configuring authentication.............................................. 53
    - Logging in using cqlsh.................................................... 54
  - Internal authorization....................................................... 55
    - Object permissions......................................................... 55
    - Configuring internal authorization.................................... 55
    - Configuring firewall port access....................................... 56
    - Enabling JMX authentication............................................ 56

## Chapter 9. Database internals

- Storage engine........................................................................ 59
- Separate table directories..................................................... 59
- Cassandra storage basics...................................................... 59
  - The write path to compaction............................................ 59
  - How Cassandra stores indexes.......................................... 62
  - About index updates........................................................ 62
- The write path of an update.................................................. 62
- About deletes....................................................................... 63
- About hinted handoff writes................................................ 63
- Reads.................................................................................... 65
  - About reads....................................................................... 65
How off-heap components affect reads................................................................. 66
Reading from a partition......................................................................................... 66
How write patterns affect reads............................................................................. 66
How the row cache affects reads........................................................................... 67
About transactions and concurrency control......................................................... 67
  Atomicity............................................................................................................... 67
  Consistency........................................................................................................... 67
  Isolation................................................................................................................ 68
  Durability.............................................................................................................. 68
  Lightweight transactions....................................................................................... 68
Data consistency....................................................................................................... 69
  About data consistency........................................................................................ 69
  About built-in consistency repair features............................................................. 69
  Configuring data consistency.............................................................................. 69
Read requests.......................................................................................................... 72
Write requests......................................................................................................... 80

Chapter 10. Configuration......................................................................................... 83
cassandra.yaml configuration file........................................................................... 83
Configuring gossip settings.................................................................................... 98
Configuring the heap dump directory..................................................................... 99
Configuring the buffered read size.........................................................................100
Configuring virtual nodes.......................................................................................100
  Enabling virtual nodes on a new cluster.............................................................. 100
  Enabling virtual nodes on an existing production cluster...................................... 100
Using multiple network interfaces.........................................................................101
Configuring logging...............................................................................................103
Commit log archive configuration..........................................................................105
Generating tokens.................................................................................................106
Hadoop support........................................................................................................107

Chapter 11. Operations............................................................................................ 109
Monitoring Cassandra............................................................................................ 109
  Getting statistics and metrics.............................................................................. 109
Using nodetool sjk................................................................................................. 111
Table statistics........................................................................................................ 114
Thread pool and read/write latency statistics......................................................... 114
Chapter 12. Backing up and restoring data .............................................. 148
  About snapshots ..................................................................................... 148
  Taking a snapshot .................................................................................. 148
  Deleting snapshot files .......................................................................... 149
Starting Cassandra as a stand-alone process

json2sstable

sstablerepairedset

sstableexpiredblockers

stop

stopdaemon
toppartitions
tpstats

truncatethints

upgradesstables

version

The cassandra utility

The cassandra-stress tool

Using the Daemon Mode

Interpreting the output of cassandra-stress

SSTable utilities

sstableexpiredblockers

sstablekeys

sstablelevelreset

sstableloader (Cassandra bulk loader)

sstablemetadata

sstableofflinerelevel

stablerepairedset

sstablescrub

sstablesplit

The sstableupgrade tool

sstable2json

json2sstable

Chapter 14. References

Starting and stopping Cassandra

Starting Cassandra as a service

Starting Cassandra as a stand-alone process
Stopping Cassandra as a service ........................................................................................................ 242
Stopping Cassandra as a stand-alone process .................................................................................. 242
Clearing the data as a service ........................................................................................................... 242
Clearing the data as a stand-alone process ...................................................................................... 242
Install locations ...................................................................................................................................... 243
Tarball installation directories ............................................................................................................. 243
Package installation directories ........................................................................................................... 243
Cassandra include file .......................................................................................................................... 244
Cassandra-CLI utility (deprecated) ...................................................................................................... 244
Table attributes .................................................................................................................................... 245

Chapter 15. Moving data to/from other databases .............................................................................. 248
Chapter 16. Troubleshooting ............................................................................................................... 249
Chapter 17. Release notes .................................................................................................................... 250
Chapter 1. About 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 datacenters 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.

The latest version of Apache Cassandra™ 2.1 is 2.1.21.

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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra is a partitioned row store database</td>
<td>Cassandra’s architecture allows any authorized user to connect to any node in any datacenter 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 of drivers so that CQL statements can be passed from client to cluster and back.</td>
</tr>
<tr>
<td>Automatic data distribution</td>
<td>Cassandra provides automatic data distribution across all nodes that participate in a ring 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.</td>
</tr>
<tr>
<td>Built-in and customizable replication</td>
<td>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 datacenter, many datacenters, and multiple cloud availability zones.</td>
</tr>
<tr>
<td>Cassandra supplies linear scalability</td>
<td>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.</td>
</tr>
</tbody>
</table>

![Diagram showing linear scalability](image-url)
New features:

- User-defined types
- Collection indexes
- Better implementation of counters that makes them safer, simpler, and typically faster
- New listsnapshots and reloaddtriggers nodetool commands
- Improved metrics reporting through the use of the metrics-core library

Performance improvements:

- Faster reads and writes than previous releases
- Improved row cache
- Reduced heap used by memtables
- New counters implementation

Compaction and repair improvements:

- Post-compaction read performance
- A configurable percentage of cold SSTables can be ignored
- Incremental node repair

Other notable changes:

- Pre-Cassandra 2.0 SSTables are no longer supported in Cassandra 2.1 and later. To upgrade to Cassandra 2.1 from a previous release that stored Cassandra 1.2.x SSTables, see the Apache Cassandra website.
- The shuffle utility for migrating to virtual nodes (vnodes) and the nodetool taketoken command have been removed. To migrate to vnodes, bootstrap a new data center.
- Cassandra 2.1 bundles and enables JNA. If JNA fails to initialize, you can disable JNA by using the -Dcassandra.boot_without_jna=true option to start Cassandra.
- Incremental replacement of compacted SSTables has been disabled.
- Improved Hadoop support
- Unique table IDs
- Improved logging using logback
- New configuration options for allocating and managing memtable memory
- Improvements to bootstrapping a node that ensure data consistency
- A number of other CQL and cqlsh changes
- Configurable properties have been added to manage counter writes.
- A configurable counter cache reduces lock contention and helps with concurrency.
- CASSANDRA-6504 has been backported to Cassandra 2.0.5 so you can perform a rolling upgrade of a database having counters to Cassandra 2.1.
What's new in Apache Cassandra 2.1

- Default data and log locations have changed for tarball installations and source checkouts. By default, the data file directory, commitlog directory, and saved caches directory are in `$CASSANDRA_HOME/data/data`, `$CASSANDRA_HOME/data/commitlog`, and `$CASSANDRA_HOME/data/saved_caches`, respectively. The log directory now defaults to `$CASSANDRA_HOME/logs`. If not set, `$CASSANDRA_HOME`, defaults to the top-level directory of the installation. Deb and RPM packages continue to use `/var/lib/cassandra` and `/var/log/cassandra` by default.

- Cassandra 2.1 maintains data consistency during bootstrapping. As you bootstrap a new node, Cassandra streams the data for the new node from an existing node that is free from range movement. If data inconsistency issues are present in the cluster, the improvement to bootstrapping handles these issues. Data inconsistency commonly occurs after frequent data deletions and a node going down.

- To inhibit the new Cassandra 2.1 bootstrapping behavior, and make Cassandra 2.0 behavior effective, start the node using the `-Dcassandra.consistent.rangemovement=false` property:

  ```bash
  # Cassandra Package installations: Add the following option to /usr/share/cassandra/cassandra-env.sh file:
  
  JVM_OPTS="$JVM_OPTS -Dcassandra.consistent.rangemovement=false"
  
  # Cassandra Tarball installations: Start Cassandra with this option:
  
  $ bin/cassandra -Dcassandra.consistent.rangemovement=false
  
  To replace a dead node, you also need to specify the address of the node from which Cassandra streams the data.
  
  For a detailed list of changes, see the Cassandra 2.1 change log.
Chapter 3. CQL

Cassandra Query Language (CQL) is the default and 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 section, examples, and command reference. The cqlsh utility for using CQL interactively on the command line is also covered.
Chapter 4. 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 exchanges information across the cluster every second. 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, which resembles a write-back cache. Once 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. Using a process called compaction Cassandra periodically consolidates SSTables, discarding obsolete data and tombstone (an indicator that data was deleted).

Cassandra is a row-oriented database. Cassandra's architecture allows any authorized user to connect to any node in any datacenter and access data using the CQL language. For ease of use, CQL uses a similar syntax to SQL. From the CQL perspective the database consists of tables. Typically, a cluster has one keyspace per application. Developers can access CQL through cqlsh as well as via drivers for application languages.

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. For more information, see Client requests.

Key structures

- **Node**
  Where you store your data. It is the basic infrastructure component of Cassandra.

- **Datacenter**
  A collection of related nodes. A datacenter can be a physical data center or virtual datacenter. Different workloads should use separate datacenters, either physical or virtual. Replication is set by datacenter. 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 datacenters. However, datacenters should never span physical locations.

- **Cluster**
  A cluster contains one or more datacenters. 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.

- **Table**
  A collection of ordered columns fetched by row. A row consists of columns and have a primary key. The first part of the key is a column name.

- **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.
Understanding the architecture

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 which node will receive the first replica of a piece of data, and how to distribute other replicas across other nodes in the cluster. Each row of data is uniquely identified by a primary key, which may be the same as its partition key, but which may also include other clustering columns. A partitioner is a hash function that derives a token from the primary key of a row. The partitioner uses the token value to determine which nodes in the cluster receive the replicas of that row. 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 datacenter. 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 datacenters 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 datacenters 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 datacenter or rack information. Use it for single-datacenter deployments or single-zone in public clouds. The GossipingPropertyFileSnitch is recommended for production. It defines a node's datacenter 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.

```bash
# Package installations: /var/lib/cassandra

# Tarball installations: install_location/data/data

In a production cluster deployment, you can change the commitlog-directory to a different disk drive from the data_file_directories.
Understanding the architecture

- 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.

The cassandra.yaml configuration file [The cassandra.yaml file is the main configuration file for Cassandra.]
Install locations [Install location topics.]

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.

In multiple data-center clusters, the seed list should include at least one node from each datacenter (replication group). More than a single seed node per datacenter is recommended for fault tolerance. Otherwise, gossip has to communicate with another datacenter 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 datacenter).

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 gossipping 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. Once the failure detector marks a node as down, missed writes are stored by other replicas for a period of time providing hinted handoff is enabled. If a node is down for longer than max_hint_window_in_ms (3 hours by default), hints are no longer saved. Nodes that die may have stored undelivered hints. Run a repair after recovering a node that has been down for an extended period. Moreover, you should routinely run nodetool repair on all nodes to ensure they have consistent data.
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 2.0.)

For example, if you have the following data:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>car</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>jim</td>
<td>36</td>
<td>camaro</td>
<td>M</td>
</tr>
<tr>
<td>carol</td>
<td>37</td>
<td>bmw</td>
<td>F</td>
</tr>
<tr>
<td>johnny</td>
<td>12</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>suzy</td>
<td>10</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

Cassandra assigns a hash value to each partition key:

<table>
<thead>
<tr>
<th>Partition key</th>
<th>Murmur3 hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jim</td>
<td>-2245462676723223822</td>
</tr>
<tr>
<td>carol</td>
<td>7723358927203680754</td>
</tr>
<tr>
<td>johnny</td>
<td>-672372954036780375</td>
</tr>
<tr>
<td>suzy</td>
<td>1168604627387940318</td>
</tr>
</tbody>
</table>

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

Hash values in a 4 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:

<table>
<thead>
<tr>
<th>Node</th>
<th>Start range</th>
<th>End range</th>
<th>Partition key</th>
<th>Hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-9223372036854775808</td>
<td>-4611686018427387903</td>
<td>johnny</td>
<td>-6723372854036780875</td>
</tr>
<tr>
<td>B</td>
<td>-4611686018427387904</td>
<td>-1</td>
<td>jim</td>
<td>-2245462676723223822</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>4611686018427387903</td>
<td>suzy</td>
<td>1168604627387940318</td>
</tr>
<tr>
<td>D</td>
<td>4611686018427387904</td>
<td>9223372036854775807</td>
<td>carol</td>
<td>7723358927203680754</td>
</tr>
</tbody>
</table>

Virtual nodes

Vnodes simplify many tasks in Cassandra:

- You no longer have to calculate and assign tokens to each node.
- Rebalancing a cluster is no longer necessary 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.
- Improves the use of heterogeneous machines in a cluster. You can assign a proportional number of vnodes to smaller and larger machines.

For more information, see the article Virtual nodes in Cassandra 1.2 and Enabling virtual nodes on an existing production cluster.

To convert an existing cluster to vnodes, see Enabling virtual nodes on an existing production cluster.

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

Prior to version 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. Starting in version 1.2, Cassandra allows many tokens per node. The new paradigm is called virtual nodes (vnodes). Vnodes allow
Understanding the architecture

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 1: 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, 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 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. 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 datacenter, use the NetworkTopologyStrategy.
- **NetworkTopologyStrategy**: Highly recommended for most deployments because it is much easier to expand to multiple datacenters when required by future expansion.

**SimpleStrategy**
Use only for a single datacenter 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 datacenter location).

**NetworkTopologyStrategy**
Use NetworkTopologyStrategy when you have (or plan to have) your cluster deployed across multiple datacenters. This strategy specifies how many replicas you want in each datacenter. 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 datacenter, 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 datacenter clusters are:

- Two replicas in each datacenter: 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 datacenter: 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 datacenter using consistency level ONE.

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

**Choosing keyspace replication options**
To set the replication strategy for a keyspace, see CREATE KEYSPACE.

When you use NetworkTopologyStrategy, during creation of the keyspace, you use the datacenter names defined for the snitch used by the cluster. To place replicas in the correct location, Cassandra requires a keyspace definition that uses the snitch-configured datacenter names. For example, if the cluster uses the PropertyFileSnitch, create the keyspace using the user-defined datacenter and rack names in the cassandra-topologies.properties file. If the cluster uses the Ec2Snitch, create the keyspace using EC2 datacenter and rack names. If the cluster uses the GoogleCloudSnitch, create the keyspace using GoogleCloud datacenter and rack names.

**Install locations** [Install location topics.]

**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 key, 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.
Understanding the architecture

Cassandra offers the following partitioners:

- **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 new Cassandra clusters and the right choice for new clusters in almost all cases.

Set the **partitioner** in the `cassandra.yaml` file:

- **Murmur3Partitioner**: `org.apache.cassandra.dht.Murmur3Partitioner`
- **ByteOrderedPartitioner**: `org.apache.cassandra.dht.ByteOrderedPartitioner`

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 and use the method for the type of partitioner you are using.

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

<table>
<thead>
<tr>
<th>Installations</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/cassandra/cassandra.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/</code></td>
</tr>
<tr>
<td></td>
<td><code>conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

**Install locations** [Install location topics.]

**Murmur3Partitioner**

The **Murmur3Partitioner** provides faster hashing and improved performance than the previous default partitioner (RandomPartitioner). You can only use Murmur3Partitioner for new clusters; you cannot change the partitioner in existing clusters. 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. You can use it in later versions of Cassandra, even when using 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.

The cassandra.yaml configuration file [The cassandra.yaml file is the main configuration file for Cassandra.]

Install locations [Install location topics.]

---

**Snitches**

A snitch determines which datacenters 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 datacenters 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 datacenter. Cassandra does its best not to have more than one replica on the same rack (which is not necessarily a physical location).

If you change snitches, you may need to perform additional steps because the snitch affects where replicas are placed. See Switching snitches.

**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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

**SimpleSnitch**

The SimpleSnitch (default) is used only for single-datacenter deployments. It does not recognize datacenter or rack information and can be used only for single-datacenter 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 datacenter, 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).
Understanding the architecture

PropertyFileSnitch

This snitch determines proximity as determined by rack and datacenter. It uses the network details located in the cassandra-topology.properties file. When using this snitch, you can define your datacenter names to be whatever you want. Make sure that the datacenter names correlate to the name of your datacenters 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-topology.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-topology.properties</td>
</tr>
</tbody>
</table>

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

   datacenter and rack names are case-sensitive.

```
# datacenter 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

# datacenter 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 datacenter 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 datacenter and rack used by this snitch:

`dc=DC1`
Understanding the architecture

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 datacenters.

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:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/conf/cassandra-rackdc.properties</code></td>
</tr>
</tbody>
</table>

**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 datacenter name and availability zones are treated as racks within a datacenter. 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 datacenter naming.) Because private IPs are used, this snitch does not work across multiple regions.

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

If you need multiple datacenters, 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:

- `datacenter` 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 datacenters for the region:

- `us-east_1_cassandra`
- `us-east_1_analytics`
- `us-east_1_search`

The datacenter naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. Datacenter names are case sensitive.
Understanding the architecture

**Keyspace strategy options**

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

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

**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 datacenter 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
   ```

   Do not make all nodes seeds, see Internode communications (gossip).

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 datacenter name and availability zones are treated as racks within a datacenter. 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 datacenter naming.)

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

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

<table>
<thead>
<tr>
<th>Region: us-east</th>
<th>Region: us-west</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node and datacenter:</td>
<td>Node and datacenter:</td>
</tr>
<tr>
<td>• node0</td>
<td>• node0</td>
</tr>
<tr>
<td>dc_suffix=_1_cassandra</td>
<td>dc_suffix=_1_cassandra</td>
</tr>
<tr>
<td>• node1</td>
<td>• node1</td>
</tr>
<tr>
<td>dc_suffix=_1_cassandra</td>
<td>dc_suffix=_1_cassandra</td>
</tr>
<tr>
<td>• node2</td>
<td>• node2</td>
</tr>
<tr>
<td>dc_suffix=_2_cassandra</td>
<td>dc_suffix=_2_cassandra</td>
</tr>
<tr>
<td>• node3</td>
<td>• node3</td>
</tr>
<tr>
<td>dc_suffix=_2_cassandra</td>
<td>dc_suffix=_2_cassandra</td>
</tr>
<tr>
<td>• node4</td>
<td>• node4</td>
</tr>
<tr>
<td>dc_suffix=_1_analytics</td>
<td>dc_suffix=_1_analytics</td>
</tr>
<tr>
<td>• node5</td>
<td>• node5</td>
</tr>
<tr>
<td>dc_suffix=_1_search</td>
<td>dc_suffix=_1_search</td>
</tr>
</tbody>
</table>

This results in four us-east data centers:

us-east_1_cassandra
us-east_2_cassandra
us-east_1_analytics
us-east_1_search

This results in four us-west data centers:

us-west_1_cassandra
us-west_2_cassandra
us-west_1_analytics
us-west_1_search

Keyspace strategy options

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

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

The cassandra.yaml configuration file ([The cassandra.yaml file is the main configuration file for Cassandra.])
Install locations ([Install location topics.])

GoogleCloudSnitch

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

The region name is treated as the datacenter name and zones are treated as racks within a datacenter. 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 datacenter naming.) This snitch can work across multiple regions without additional configuration.

If you are using only a single datacenter, you do not need to specify any properties.
If you need multiple datacenters, 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 datacenter in its cassandra-rackdc.properties file:
Understanding the architecture

datacenter 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

datacenter 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.

Client requests
Client read or write requests can be sent to any node in the cluster because all nodes in Cassandra are peers. When a client connects to a node and issues a read or write request, that node serves as the coordinator for that particular client operation.

The job of the coordinator is to act as a proxy between the client application and the nodes (or replicas) that own the data being requested. The coordinator determines which nodes in the ring should get the request based on the cluster configured partitioner and replica placement strategy.
Chapter 5. Planning a cluster deployment

Planning has moved to Planning and testing cluster deployments.
Chapter 6. Installing Apache Cassandra

Installing Apache Cassandra 2.1 on RHEL-based systems

The latest version of Apache Cassandra™ 2.1 is 2.1.21.

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

To install on SUSE, use the Cassandra binary tarball distribution.

Prerequisites:

- Yum Package Management application installed.
- Root or sudo access to the install machine.
- DataStax recommends using the latest version of either OpenJDK 8 or Oracle Java Platform, Standard Edition 8 (JDK).
- Python 2.7 if using cqlsh.

The packaged releases create a cassandra user. When starting Cassandra as a service, the service runs as this user.

In a terminal window:

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

   ```
   $ java -version
   
   DataStax recommends using the latest version of Java 8 on all nodes. See Installing the JDK and Python 2.7.
   ```

2. Add the Apache Cassandra repository to the `/etc/yum.repos.d/cassandra.repo`:

   ```
   [cassandra]
   name=Apache Cassandra
   baseurl=https://www.apache.org/dist/cassandra/redhat/21x/
   gpgcheck=1
   repo_gpgcheck=1
   gpgkey=https://www.apache.org/dist/cassandra/KEYS
   ```

3. Install the packages:

   Apache Cassandra only provides the latest patch version. If you need to install an earlier patch version, contact DataStax Support.

   ```
   $ sudo yum update

   $ sudo yum install cassandra
   ```

   Cassandra is ready for configuration.
4. To change the location of the default directories (/var/lib/cassandra), see the following in /etc/cassandra/conf/cassandra.yaml:
   • data_file_directories
   • commitlog_directory
   • saved_caches_directory

5. To change the location of the log files (/var/log/cassandra), replace the path to the log directory in /usr/sbin/cassandra:

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

What's next:
   • Starting Cassandra as a service
   • Package installation directories
   • Initializing a multiple node cluster (single datacenter)
   • Initializing a multiple node cluster (multiple datacenters)
   • Recommended production settings
   • Key components for configuring Cassandra

Installing Apache Cassandra 2.1 on Debian-based systems

The latest version of Apache Cassandra™ 2.1 is 2.1.21.

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

Prerequisites:
   • APT (Advanced Package Tool) is installed.
   • Root or sudo access to the install machine.
   • Python 2.7 if using cqlsh.
   • DataStax recommends using the latest version of either OpenJDK 8 or Oracle Java Platform, Standard Edition 8 (JDK).

The packaged releases create a cassandra user. When starting Cassandra as a service, the service runs as this user.

In a terminal window:

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

   ```
   $ java -version
   ```

   DataStax recommends using the latest version of Java 8 on all nodes. See Installing the JDK and Python 2.7.
2. Add the Apache Cassandra repository to the `/etc/apt/sources.list.d/cassandra.sources.list`:

```
$ echo "deb http://www.apache.org/dist/cassandra/debian 21x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

3. If using Oracle Java on Debian systems:
   
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
   ```

   This allows installation of the Oracle JVM instead of the OpenJDK JVM.

   b. Save and close the file when you are done adding/editing your sources.

4. Add the Apache Cassandra repository key to your aptitude trusted keys.

```
$ curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
```

5. Install the packages:

   Apache Cassandra only provides the latest patch version. If you need to install an earlier patch version, contact DataStax Support.

```
$ sudo apt-get update
```

If you encounter this error:

```
GPG error: http://dl.bintray.com/apache/cassandra 21x InRelease: The following signatures couldn't be verified because the public key is not available: NO_PUBKEY A278B781FE4B2BDA
```

a. Add the public key A278B781FE4B2BDA as follows:

```
$ sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-key A278B781FE4B2BDA
```

   The key may be different. If this happens, use key listed in the error message. For a full list of Apache contributors public keys, refer to https://www.apache.org/dist/cassandra/KEYS.

b. Update the packages:

```
$ sudo apt-get update
```
6. Install Cassandra:

   $ sudo apt-get install cassandra

7. Install the optional tools:

   $ sudo apt-get install cassandra-tools=2.1.x  # Optional utilities

8. 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/*

Cassandra is ready for configuration.

9. To change the location of the default directories (/var/lib/cassandra), see the following in /etc/cassandra/conf/cassandra.yaml:

   - data_file_directories
   - commitlog_directory
   - saved_caches_directory

10. To change the location of the log files (/var/log/cassandra), replace the path to the log directory in /usr/sbin/cassandra:

    cassandraParms="$cassandra_parms -Dcassandra.logdir=/var/log/cassandra"

What's next:

- Starting Cassandra as a service
- Package installation directories
- Initializing a multiple node cluster (single datacenter)
- Initializing a multiple node cluster (multiple datacenters)
- Recommended production settings
- Key components for configuring Cassandra

Installing Apache Cassandra 2.1 on any Linux-based platform

The latest version of Apache Cassandra™ 2.1 is 2.1.21.

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

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

Prerequisites:
Installing Apache Cassandra

- DataStax recommends using the latest version of either OpenJDK 8 or Oracle Java Platform, Standard Edition 8 (JDK).
- Python 2.7 if using cqlsh.
- If you are using an older RHEL-based Linux distribution, such as CentOS-5, you may see the following error: `GLIBCXX_3.4.9 not found`. You must replace the Snappy compression/decompression library (`snappy-java-1.0.5.jar`) with the `snappy-java-1.0.4.1.jar`.

The binary tarball runs as a stand-alone process.

In a terminal window:

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

   ```
   $ java -version
   ```

   DataStax recommends using the latest version of Java 8 on all nodes. See Installing the JDK and Python 2.7.

2. Download Apache Cassandra:
   - From Download Cassandra.
   - Use curl to download from one of the mirrors. For example:

     ```
     $ curl -OL http://mirrors.sonic.net/apache/cassandra/2.1.21/apache-cassandra-2.1.21-bin.tar.gz
     ```

     Apache Cassandra only provides the latest patch version. If you need to install an earlier patch version, contact DataStax Support.

3. Verify the integrity of the downloaded tarball using one of the methods described here.

4. Extract the tarball to the desired location:

   ```
   $ tar -xzvf apache-cassandra-2.1.21-bin.tar.gz
   ```

   Cassandra is ready for configuration.

5. To change the location of the default directories (`install_location/data`), see the following in `install_location/conf/cassandra.yaml`:
   - `data_file_directories`
   - `commitlog_directory`
   - `saved_caches_directory`

6. To change the location of the log files (`install_location/logs`), set the path to log directory in `bin/cassandra`:

   ```
   cassandra_parms="$cassandra_parms -Dcassandra.logdir=$CASSANDRA_HOME/logs"
   ```

What's next:
- Starting Cassandra as a stand-alone process
- Tarball installation directories
- Initializing a multiple node cluster (single datacenter)
• Initializing a multiple node cluster (multiple datacenters)
• Recommended production settings
• Key components for configuring Cassandra

Recommended production settings

The following sections provide recommendations for optimizing your Apache Cassandra™ installation on Linux:

Use the latest Java Virtual Machine

Use the latest 64-bit version of Oracle Java Platform, Standard Edition 8 (JDK) or OpenJDK 8.

Synchronize clocks

Synchronize the clocks on all nodes, using NTP (Network Time Protocol) or other methods. This is required because Cassandra only overwrites a column if there is another version whose timestamp is more recent.

TCP settings

To handle thousands of concurrent connections used by Cassandra, DataStax recommends these settings to optimize the Linux network stack. Add these settings to /etc/sysctl.conf.

```
net.core.rmem_max = 16777216
    net.core.wmem_max = 16777216
    net.core.rmem_default = 16777216
    net.core.wmem_default = 16777216
    net.core.optmem_max = 40960
    net.ipv4.tcp_rmem = 4096 87380 16777216
    net.ipv4.tcp_wmem = 4096 65536 16777216
```

To set immediately (depending on your distribution):

```
$ sudo sysctl -p /etc/sysctl.conf

$ sudo sysctl -p /etc/sysctl.d/filename.conf
```

Make sure that new settings persist after reboot

Depending on your environment, some of the following settings may not be persisted after reboot. Check with your system administrator to ensure they are viable for your environment.

Optimize SSDs

The default SSD configurations on most Linux distributions are not optimal. Follow these steps to ensure the best 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. Apply the same rotational flag setting 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.
Installing Apache Cassandra

- The deadline scheduler optimizes requests to minimize IO latency. If in doubt, use the deadline scheduler.

4. Set the `readahead` value for the block device to 8 KB.
   
   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
touch /var/lock/subsys/local
   echo 0 > /sys/class/block/sda/queue/rotational
   echo 8 > /sys/class/block/sda/queue/read_ahead_kb
   ```

Use the optimum --setra setting for RAID on SSD

The optimum `readahead` setting for RAID on SSDs (in Amazon EC2) is 8KB, the same as it is for non-RAID SSDs. For details, see Optimizing SSDs.

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.

- 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
```

Set user resource limits

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

Package installations: 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 installations: In RHEL version 6.x, 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
```
For RHEL 6.x-based systems, also set the `nproc` limits in `/etc/security/limits.d/90-nproc.conf`:

```bash
cassandra_user - nproc 32768
```

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

```bash
vm.max_map_count = 1048575
```

For installations on Debian and Ubuntu operating systems, the `pam_limits.so` module is not enabled by default. Edit the `/etc/pam.d/su` file and uncomment this line:

```bash
session    required   pam_limits.so
```

This change to the PAM configuration file ensures that the system reads the files in the `/etc/security/limits.d` directory.

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

```bash
$ 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:

```bash
$ cat /proc/pid/limits
```

### Disable swap

Failure to disable swap entirely 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.

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

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

### Check the Java Hugepages setting

Many modern Linux distributions ship with Transparent Hugepages enabled by default. When Linux uses Transparent Hugepages, the kernel tries to allocate memory in large chunks (usually 2MB), rather than 4K. This can improve performance by reducing the number of pages the CPU must track. However, some applications still allocate memory based on 4K pages. This can cause noticeable performance problems when Linux tries to defrag 2MB pages. For more information, see [Cassandra Java Huge Pages](https://cassandra.apache.org/doc/released/latest/ops/resize-cassandra.html) and this [RedHat bug report](https://bugzilla.redhat.com/show_bug.cgi?id=1032197).
Installing Apache Cassandra

To solve this problem, disable `defrag` for hugepages. Enter:

```
$ echo never | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
```

Set the heap size for optimal Java garbage collection in Cassandra

See Tuning Java resources.

Apply 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/spinning_disk
```

To set `setra`:

```
$ sudo blockdev --setra 128 /dev/spinning_disk
```

The recommended setting for RAID on SSDs is the same as that for SSDs that are not being used in a RAID installation. For details, see Optimizing SSDs.

Install locations

Installing the JDK and Python 2.7

Installation information is located in Installing the JDK.

Installing Apache Cassandra on cloud providers

You can install Cassandra by creating your instances using an AMI for a supported platform and from a trusted source. Then use the appropriate install method for your platform.

Uninstalling Apache Cassandra

This topic provides information on completely removing Apache Cassandra™ from your machine.

If you want to keep the installation or if you don't remove the installation immediately, see Preventing the node from re-joining the cluster.

Select the uninstall method for your type of installation.

Uninstalling Debian- and RHEL-based packages

Use this method when you have installed Apache Cassandra™ using APT or Yum.
1. Stop the Cassandra services:
   $ sudo service cassandra stop

2. Make sure all services are stopped:
   $ ps auwx | grep cassandra

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

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-*"
   
   Debian-based packages:
   $ sudo apt-get purge "cassandra-**"

Uninstalling the binary tarball

Use this method when you have installed Cassandra using the binary tarball.

1. Stop the node:
   $ ps auwx | grep cassandra

2. $ sudo kill <pid>

3. Stop the DataStax Agent if installed:
   $ sudo kill datastax_agent_pid

4. Remove the installation directory.

Preventing the node from re-joining the cluster

The following steps will prevent the node from re-joining the cluster if someone inadvertently starts Cassandra again.

1. Stop Cassandra using one of the above methods.

2. In the cassandra.yaml:
   a. Change the cluster_name to DECOMMISSIONED.
   b. Set the - seeds list to 127.0.0.1.
c. Restart the node.
Chapter 7. Initializing a cluster

Initializing a multiple node cluster (single datacenter)

This topic contains information for deploying a Cassandra cluster with a single data center.

**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, Data replication, 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).
- Determine the snitch and replication strategy. The GossipingPropertyFileSnitch and NetworkTopologyStrategy are recommended for production environments.
- If using multiple datacenters, 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 datacenter 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 datacenter is a grouping of nodes. datacenter is synonymous with replication group, that is, a grouping of nodes configured together for replication purposes.

1. Suppose you install Cassandra on these nodes:

<table>
<thead>
<tr>
<th>Node</th>
<th>IP Address</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>110.82.155.0</td>
<td>(seed1)</td>
</tr>
<tr>
<td>node1</td>
<td>110.82.155.1</td>
<td></td>
</tr>
<tr>
<td>node2</td>
<td>110.82.155.2</td>
<td></td>
</tr>
<tr>
<td>node3</td>
<td>110.82.156.3</td>
<td>(seed2)</td>
</tr>
<tr>
<td>node4</td>
<td>110.82.156.4</td>
<td></td>
</tr>
<tr>
<td>node5</td>
<td>110.82.156.5</td>
<td></td>
</tr>
</tbody>
</table>

   It is a best practice to have more than one seed node per datacenter.

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.

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:
Initializing a cluster

a. Stop Cassandra:

```
$ sudo service cassandra stop
```

b. Clear the data:

```
$ sudo rm -rf /var/lib/cassandra /*
```

Tarball installations:

a. Stop Cassandra:

```
$ ps auwx | grep cassandra
$ sudo kill pid
```

b. Clear the data:

```
$ sudo rm -rf install_location/data/*
```

4. Set the properties in the cassandra.yaml file for each node:

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**: GossipingPropertyFileSnitch (See endpoint_snitch.) If you are changing snitches, see Switching snitches.

  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:

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

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

6. 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.

   If the node has restarted because of automatic restart, you must first stop the node and clear the directories, as described above.

   **Package installations:**

   ```
   $ sudo service cassandra start
   ```

   **Tarball installations:**

   ```
   $ cd install_location
   $ cd 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).

   ![Node Tool Status](image)

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

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/cassandra/cassandra.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>
Initializing a multiple node cluster (multiple datacenters)

This topic contains information for deploying a Cassandra cluster with multiple data centers.

This example describes installing a six node cluster spanning two datacenters. Each node is configured to use the GossipingPropertyFileSnitch (multiple rack aware) and 256 virtual nodes (vnodes).

In Cassandra, the term datacenter is a grouping of nodes. datacenter 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, Data replication, 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).
- Determine the snitch and replication strategy. The GossipingPropertyFileSnitch and NetworkTopologyStrategy are recommended for production environments.
- If using multiple datacenters, 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 datacenter is not possible.
- Other possible configuration settings are described in The cassandra.yaml configuration file and property files such as cassandra-rackdc.properties.

1. Suppose you install Cassandra on these nodes:

<table>
<thead>
<tr>
<th>Node</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>node0</td>
<td>10.168.66.41</td>
</tr>
<tr>
<td>node1</td>
<td>10.176.43.66</td>
</tr>
<tr>
<td>node2</td>
<td>10.168.247.41</td>
</tr>
<tr>
<td>node3</td>
<td>10.176.170.59</td>
</tr>
<tr>
<td>node4</td>
<td>10.169.61.170</td>
</tr>
<tr>
<td>node5</td>
<td>10.169.30.138</td>
</tr>
</tbody>
</table>

   It is a best practice to have more than one seed node per datacenter.

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.

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:
Initializing a cluster

a. Stop Cassandra:

```
$ sudo service cassandra stop
```

b. Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/*
```

Tarball installations:

a. Stop Cassandra:

```
$ ps auwx | grep cassandra
$ sudo kill pid
```

b. Clear the data:

```
$ sudo rm -rf install_location/data/*
```

4. Set the properties in the cassandra.yaml file for each node:

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**: *GossipingPropertyFileSnitch* *(See endpoint_snitch.*) If you are changing snitches, see *Switching snitches.*

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:

```
class_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
```

Include at least one node from each datacenter.
5. In the `cassandra-rackdc.properties` file, assign the datacenter and rack names you determined in the Prerequisites. For example:

**Nodes 0 to 2**

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

**Nodes 3 to 5**

```plaintext
# 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.

   If the node has restarted because of automatic restart, you must first stop the node and clear the directories, as described above.

**Package installations:**

```bash
$ sudo service cassandra start
```

**Tarball installations:**

```bash
$ cd install_location

$ cd bin/cassandra
```

7. To check that the ring is up and running, run:

**Package installations:**

```bash
$ nodetool status
```

**Tarball installations:**

```bash
$ cd install_location

$ bin/nodetool status
```

Each node should be listed and its status and state should be UN (Up Normal).

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

Install locations
Chapter 8. Security

Securing Cassandra

Cassandra provides these security features to the open source community.

- **Client-to-node encryption**
  Cassandra includes an optional, secure form of communication from a client machine to a database cluster. Client to server SSL ensures data in flight is not compromised and is securely transferred back/forth from client machines.

- **Authentication based on internally controlled login accounts/passwords**
  Administrators can create users who can be authenticated to Cassandra database clusters using the CREATE USER command. Internally, Cassandra manages user accounts and access to the database cluster using passwords. User accounts may be altered and dropped using the Cassandra Query Language (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.

- **Generate a private and public key pair on each node of the cluster. Use an alias that identifies the node.**
  - Prompts for the keystores 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.

```bash
$ keytool -genkey -keyalg RSA -alias node0 -validity 36500 -keystore keystore.node0
```

In this example, the `-validity 36500` gives this key pair a validity period of 100 years. The default key pair validity is 90 days.

- 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`.

```bash
$ keytool -genkey -keyalg RSA -alias node0 -validity 36500 -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.

```bash
$ keytool -export -alias cassandra -file node0.cer -keystore truststore.node0
```

- Add the `node0.cer` certificate to the node0 truststore of the node using the `keytool -import` command.

```bash
$ 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.

```bash
$ keytool -importkeystore -srckeystore keystore.node0 -destkeystore node0.p12 -deststoretype PKCS12 -srcstorepass cassandra -deststorepass cassandra openssl pkcs12 -in node0.p12 -nokeys -out node0.cer.pem
openssl pkcs12 -in node0.p12 -nodes -nocerts -out node0.key.pem
```

- 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.

```bash
$ 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.

```bash
$ 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`).

1. Generate the certificate as described in **Client-to-node encryption**.
2. Import the user's certificate into every node's truststore using keytool:

```bash
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.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/cassandra/cassandra.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

**Prerequisites:** All nodes must have all the relevant SSL certificates on all nodes. See [Preparing server certificates](#).

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

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 options 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`:

```yaml
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 trustore file; ex: conf/truststore.node0
  truststore: conf/truststore.node0
  # The password for your truststore file
  truststore_password: cassandra
```

**What's next:** Cassandra must be restarted after making changes to the `cassandra.yaml` file.

### Using cqlsh with SSL encryption

Using a `cqlshrc` file means you don't have to override the `SSL_CERTFILE` environmental variables every time.

Additional settings in the `cqlshrc` file are described in [Creating and using the cqlshrc file](#).
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.

```plaintext
[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
```

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.

```bash
$ 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.

```bash
$ 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.

The location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

1. First, follow steps #1-5 in Enabling JMX authentication.
Security

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.

```java
JVM_OPTS="-Djavax.net.ssl.keyStore=/home/automaton/keystore.node0"
JVM_OPTS="-Djavax.net.ssl.keyStorePassword=cassandra"
JVM_OPTS="-Djavax.net.ssl.trustStore=/home/automaton/truststore.node0"
JVM_OPTS="-Djavax.net.ssl.trustStorePassword=cassandra"
JVM_OPTS="-Dcom.sun.management.jmxremote.ssl.need.client.auth=true"
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.

```java
-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](#).

To enable node-to-node SSL, you must set the `server_encryption_options` in the `cassandra.yaml` file. The location of the `cassandra.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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.

```yaml
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's 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.

```bash
$ 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
- cassandra-cli
- 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 USER**
- **CREATE USER**
- **DROP USER**
- **LIST USERS**

To configure authorization, see Configuring internal authorization.

The location of the cassandra.yaml file depends on the type of installation:
1. Change the authenticator option in the `cassandra.yaml` file to `PasswordAuthenticator`. By default, the authenticator option is set to `AllowAllAuthenticator`.

```yaml
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.

4. Start `cqlsh` using the superuser name and password.

```bash
./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 enter credentials every time you launch `cqlsh`, you can create a `cqlshrc` file in the `.cassandra` directory, which is in your home directory. When present, this file passes default login information to `cqlsh`.

Sample `cqlshrc` files are available in:

- **Package installations**: `/etc/cassandra`
- **Tarball installations**: `install_location/conf`

1. Open a text editor and create a file that specifies a user name and password.

```yaml
[authentication]
username = fred
password = !!bang!!$
```

Additional settings in the `cqlshrc` file are described in Creating and using the `cqlshrc` file.

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 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 because the tables are used by most Cassandra tools:

- system.schema_keyspace
- system.schema_columns
- system.schema_columnfamilies
- 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.

To configure authentication, see Configuring authentication.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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

   ```yaml
   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.

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 bi-directional communication among 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 1: Public ports**

<table>
<thead>
<tr>
<th>Port number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>SSH port</td>
</tr>
</tbody>
</table>

**Table 2: Cassandra inter-node ports**

<table>
<thead>
<tr>
<th>Port number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7000</td>
<td>Cassandra inter-node cluster communication.</td>
</tr>
<tr>
<td>7001</td>
<td>Cassandra SSL inter-node cluster communication.</td>
</tr>
<tr>
<td>7199</td>
<td>Cassandra JMX monitoring port.</td>
</tr>
</tbody>
</table>

**Table 3: Cassandra client ports**

<table>
<thead>
<tr>
<th>Port number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9042</td>
<td>CQL native clients port.</td>
</tr>
<tr>
<td>9160</td>
<td>Cassandra client port (Thrift).</td>
</tr>
</tbody>
</table>

/ en/landing_page/doc/landing_page/planning/planningEC2.html

## 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 you enable JMX authentication, ensure that tools that use JMX, such as nodetool are configured to use authentication.

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:
   
   LOCAL_JMX=no
   
   2. Depending on whether the JDK or JRE is installed:
• Copy the `jmxremote.password.template` from `/jdk_install_location/jre/lib/management/` to `/etc/cassandra/` and rename it to `jmxremote.password`:

```bash
$ cp /jdk_install_dir/lib/management/jmxremote.password.template /etc/cassandra/
```

• Copy the `jmxremote.password.template` from `/jre_install_location/lib/management/` to `/etc/cassandra/` and rename it to `jmxremote.password`:

```bash
$ cp /jre_install_dir/lib/management/jmxremote.password.template /etc/cassandra/
```

3. Change the ownership of `jmxremote.password` to the user you run cassandra with and change permission to read only:

```bash
$ 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:

```bash
monitorRole QED
controlRole R&D
cassandra cassandrapassword
```

This cassandra user and cassandra password is just an example. Specify the user and password for your environment.

5. Add the cassandra user with read and write permission to `/jre_install_location/lib/management/jmxremote.access`:

```bash
monitorRole readonly
cassandra readwrite
controlRole readwrite \create javax.management.monitor.,javax.management.timer. \unregister
```

6. Restart Cassandra.

7. Run nodetool with the cassandra user and password.

```bash
$ nodetool -u cassandra -pw cassandra status
```

If you run nodetool without user and password, you see an error similar to:

```
root@VM1 cassandra# nodetool status
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)
```
Chapter 9. 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 then 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 hard disks (HDD) and solid-state disks (SSD). On HDD, writing randomly involves a higher number of seek operations than sequential writing. The seek penalty incurred can be substantial. Using sequential IO, and thereby avoiding write amplification and disk failure, Cassandra accommodates inexpensive, consumer SSDs extremely well.

Separate table directories

Cassandra provides fine-grained control of table storage on disk, writing tables to disk using separate directories for each table. From the installation directory, data files are stored using this directory and file naming format on default tarball installations:

```
data/data/ks1/cf1-5be396077b811e3ab9dc4b9ac088d/ks1-cf1-hc-1-Data.db
```

On packaged installations, the data files are stored in the same format, but in `/var/lib/cassandra/data` by default. In this example, ks1 represents the keyspace name to distinguish the keyspace for streaming or bulk loading data. A hexadecimal string, 5be396077b811e3ab9dc4b9ac088d in this example, is appended to table names to represent unique table IDs.

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 SSD’s for better performance, and also divvy up tables across all attached storage devices for better I/O balance at the storage layer.

Cassandra storage basics

To manage and access data in Cassandra, it is important to understand how Cassandra stores data. The hinted handoff feature and Cassandra conformance and non-conformance to the ACID (atomic, consistent, isolated, durable) database properties are key concepts in this discussion. 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.

The write path to compaction

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

- Logging data in the commit log
Database internals

- Writing data to the memtable
- Flushing data from the memtable
- Storing data on disk in SSTables
- Compaction

Logging writes and memtable storage

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

Flushing data from the memtable

When memtable contents exceed a configurable threshold, the memtable data, which includes indexes, is put in a queue to be flushed to disk. You can configure the length of the queue by changing `memtable_heap_space_in_mb` or `memtable_offheap_space_in_mb` setting in the `cassandra.yaml`. 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 the `nodetool flush` command. Typically, before restarting nodes, flushing the memtable is recommended to reduce commit log replay time. To flush the data, Cassandra sorts memtables by token and then writes the data to disk sequentially.

Storing data on disk in SSTables

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

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.

For each SSTable, Cassandra creates these structures:

- Partition index
  A list of partition keys and the start position of rows in the data file (on disk)
- Partition summary (in memory)
  A sample of the partition index.
- Bloom filter

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 manages the accumulation of SSTables on disk using compaction.

Cassandra also does not delete in place because the SSTable is 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.
During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. This diagram depicts the compaction process:

Compauction merges the data in each SSTable by partition key, selecting the latest data for storage based on its timestamp. Cassandra can merge the data performantly, without random IO, because rows are sorted by partition key within each SSTable. After evicting tombstones and removing deleted data, columns, and rows, the compaction process consolidates SSTables into a single file. The old SSTable files are deleted as soon as any pending reads finish using the files. Disk space occupied by old SSTables becomes available for reuse.

Cassandra 2.1 improves read performance after compaction by performing an incremental replacement of compacted SSTables. Instead of waiting for the entire compaction to finish and then throwing away the old SSTable (and cache), 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. The dramatic cache miss is gone. Cassandra provides predictable high performance even under heavy load.

**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.

For more information about compaction strategies, see When to Use Leveled Compaction and Leveled Compaction in Apache Cassandra.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>
How Cassandra stores and distributes indexes

Internally, a Cassandra index is a data partition. In the example of a music service, the playlists table includes an artist column and uses a compound partition key: id is the partition key and song_order is the clustering column.

```
CREATE TABLE playlists (
    id uuid,
    song_order int,
    . . .
    artist text,
    PRIMARY KEY (id, song_order ) );
```

As shown in the music service example, to filter the data based on the artist, create an index on artist. Cassandra uses the index to pull out the records in question. An attempt to filter the data before creating the index will fail because the operation would be very inefficient. A sequential scan across the entire playlists dataset would be required. After creating the artist index, Cassandra can filter the data in the playlists table by artist, such as Fu Manchu.

The partition is the unit of replication in Cassandra. In the music service example, partitions are distributed by hashing the playlist id and using the ring to locate the nodes that store the distributed data. Cassandra would generally store playlist information on different nodes, and to find all the songs by Fu Manchu, Cassandra would have to visit different nodes. To avoid these problems, each node indexes its own data.

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

About index updates

As with relational databases, keeping indexes up to date is not free, so unnecessary indexes should be avoided. When a column is updated, the index is updated as well. If the old column value was still 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.

The write path of an update

Cassandra treats each new row as an upsert: if the new row has the same primary key as that of an existing row, Cassandra processes it as an update to the existing row.

During a write, Cassandra adds each new row to the database without checking on whether a duplicate record exists. This policy makes it possible that many versions of the same row may exist in the database. For more details about writes, see The write path to compaction.

Periodically, the rows stored in memory are streamed to disk into structures called SSTables. At certain intervals, Cassandra compacts smaller SSTables into larger SSTables. If Cassandra encounters two or more versions of the same row during this process, Cassandra only writes the most recent version to the new SSTable. After compaction, Cassandra drops the original SSTables, deleting the outdated rows.

Most Cassandra installations store replicas of each row on two or more nodes. Each node performs compaction independently. This means that even though out-of-date versions of a row have been dropped from one node, they may still exist on another node.

This is why Cassandra performs another round of comparisons during a read process. When a client requests data with a particular primary key, Cassandra retrieves many versions of the row from one or more replicas. The version with the most recent timestamp is the only one returned to the client ("last-write-wins").

Some database operations may only write partial updates of a row, so some versions of a row may include some columns, but not all. During a compaction or write, Cassandra assembles a complete version of each row from the partial updates, using the most recent version of each column.
About deletes

The way Cassandra deletes data differs from the way a relational database deletes data. 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, for example, to clear it out faster. 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 \texttt{gc\_grace\_seconds}. After data is marked with a tombstone, the data is automatically removed during the normal compaction process.

Facts about deleted data to keep in mind are:

- Cassandra does not immediately remove data marked for deletion from disk. The deletion occurs during compaction.
- If you use the \texttt{size-tiered} or \texttt{date-tiered} compaction strategy, you can drop data immediately by manually starting the compaction process. Before doing so, understand the documented disadvantages of the process.
- Deleted data can reappear if you do not run node repair routinely.

Why deleted data can reappear

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 \texttt{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 \texttt{gc\_grace\_seconds}, remove the node, wipe it, and bootstrap it again.

About hinted handoff writes

Hinted handoff is a Cassandra feature that optimizes the cluster consistency process and anti-entropy when a replica-owning node is not available, due to network issues or other problems, to accept a replica from a successful write operation. Hinted handoff is not a process that guarantees successful write operations, except when a client application uses a consistency level of \texttt{ANY}. You enable or disable hinted handoff in the \texttt{cassandra.yaml} file.

How hinted handoff works

During a write operation, when hinted handoff is enabled and consistency can be met, the coordinator stores a hint about dead replicas in the local system.hints table under either of these conditions:

- A replica node for the row is known to be down ahead of time.
- A replica node does not respond to the write request.

When the cluster cannot meet the consistency level specified by the client, Cassandra does not store a hint. A hint indicates that a write needs to be replayed to one or more unavailable nodes. The hint consists of:

- The location of the replica that is down
- Version metadata
- The actual data being written

By default, hints are saved for three hours after a replica fails because if the replica is down longer than that, it is likely permanently dead. You can configure this interval of time using the \texttt{max\_hint\_window\_in\_ms} property in the \texttt{cassandra.yaml} file. If the node recovers after the save time has elapsed, run a repair to re-allocate the data written during the down time.
After a node discovers from gossip that a node for which it holds hints has recovered, the node sends the data row corresponding to each hint to the target. Additionally, the node checks every ten minutes for any hints for writes that timed out during an outage too brief for the failure detector to notice through gossip.

For example, in a cluster of two nodes, A and B, having a replication factor (RF) of 1, each row is stored on one node. Suppose node A is down while we write row K to it with consistency level of one. The write fails because reads always reflect the most recent write when:

\[ W + R > \text{replication factor} \]

where W is the number of nodes to block for writes and R is the number of nodes to block for reads. Cassandra does not write a hint to B and call the write good because Cassandra cannot read the data at any consistency level until A comes back up and B forwards the data to A.

In a cluster of three nodes, A (the coordinator), B, and C, each row is stored on two nodes in a keyspace having a replication factor of 2. Suppose node C goes down. The client writes row K to node A. The coordinator, replicates row K to node B, and writes the hint for downed node C to node A.

Cassandra, configured with a consistency level of ONE, calls the write good because Cassandra can read the data on node B. When node C comes back up, node A reacts to the hint by forwarding the data to node C. For more information about how hinted handoff works, see "Modern hinted handoff" by Jonathan Ellis.

**Extreme write availability**

For applications that want Cassandra to accept writes even when all the normal replicas are down, when not even consistency level ONE can be satisfied, Cassandra provides consistency level ANY. ANY guarantees that the write is durable and will be readable after an appropriate replica target becomes available and receives the hint replay.

**Performance**

By design, hinted handoff inherently forces Cassandra to continue performing the same number of writes even when the cluster is operating at reduced capacity. Pushing your cluster to maximum capacity with no allowance for failures is a bad idea.

Hinted handoff is designed to minimize the extra load on the cluster.

All hints for a given replica are stored under a single partition key, so replaying hints is a simple sequential read with minimal performance impact.

If a replica node is overloaded or unavailable, and the failure detector has not yet marked it down, then expect most or all writes to that node to fail after the timeout triggered by `write_request_timeout_in_ms`, which defaults to 10 seconds.

If this happens on many nodes at once this could become substantial memory pressure on the coordinator. So the coordinator tracks how many hints it is currently writing, and if this number gets too high it will temporarily refuse writes (withOverloadedException) whose replicas include the misbehaving nodes.
Removal of hints

When removing a node from the cluster by decommissioning the node or by using the `nodetool removenode` command, Cassandra automatically removes hints targeting the node that no longer exists. Cassandra also removes hints for dropped tables.

Scheduling repair weekly

At first glance, it seems that hinted handoff eliminates the need for 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.

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

<table>
<thead>
<tr>
<th>Installation Type</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td><code>/etc/cassandra/cassandra.yaml</code></td>
</tr>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

Reads

About reads

To satisfy a read, Cassandra must combine results from the active memtable and potentially multiple SSTables. First, Cassandra checks the Bloom filter. Each SSTable has a Bloom filter associated with it that checks the probability of having any data for the requested partition in the SSTable before doing any disk I/O.

If the Bloom filter does not rule out the SSTable, Cassandra checks the partition key cache and takes one of these courses of action:

- If an index entry is found in the cache:
  
  # Cassandra goes to the compression offset map to find the compressed block having the data.
  
  # Fetches the compressed data on disk and returns the result set.

- If an index entry is not found in the cache:

  # Cassandra searches the partition summary to determine the approximate location on disk of the index entry.
  
  # Next, to fetch the index entry, Cassandra hits the disk for the first time, performing a single seek and a sequential read of columns (a range read) in the SSTable if the columns are contiguous.
  
  # Cassandra goes to the compression offset map to find the compressed block having the data.
  
  # Fetches the compressed data on disk and returns the result set.
How off-heap components affect reads

To increase the data handling capacity per node, Cassandra keeps these components off-heap:

- Bloom filter
- Compression offsets map
- Partition summary

Of the components in memory, only the partition key cache is a fixed size. Other components grow as the data set grows.

**Bloom filter**

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 summary**

By default, the partition summary is a sample of the partition index. You configure sample frequency by changing the index_interval property in the table definition, also if you want to trade memory for performance.

**Compression offsets**

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.

**Reading from a partition**

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.

**How write patterns affect reads**

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. For more information about LCS, see the article, Leveled Compaction in Apache Cassandra.
How the row cache affects reads

Typical of any database, reads are fastest when the most in-demand data (or hot working set) fits into memory. Although all modern storage systems rely on some form of caching to allow for fast access to hot data, not all of them degrade gracefully when the cache capacity is exceeded and disk I/O is required. Cassandra's read performance benefits from built-in caching, shown in the following diagram.

The red lines in the SSTables in this diagram are fragments of a row that Cassandra needs to combine to give the user the requested results. Cassandra caches the merged value, not the raw row fragments. That saves some CPU and disk I/O.

The row cache is not write-through. If a write comes in for the row, the cache for it is invalidated and is not cached again until it is read.

For rows that are accessed frequently, Cassandra 2.1 has improved the built-in partition key cache and an optional row cache.

About transactions and concurrency control

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. By default, all operations in a batch are performed atomically. Cassandra uses a batch log to ensure all operations in a batch are applied atomically. There is a performance penalty for batch atomicity when a batch spans multiple partitions. If you do not want to incur this penalty, use the UNLOGGED option. Using UNLOGGED makes the batch operation atomic only within a single partition.

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 timestamps to determine the most recent update to a column. Depending on the version of the Native CQL Protocol, the timestamp is provided by either the client application or the server. 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.

Native CQL Protocol V3 supports client-side timestamps. Be sure to check your client's documentation to ensure that it generates client-side timestamps and that this feature is activated.

Consistency

Cassandra offers two consistency types:
Database internals

- **Tunable consistency**
  
  Availability and consistency can be tuned, and can be strong in the CAP sense--data is made consistent across all the nodes in a distributed database cluster.

- **Linearizable consistency**
  
  In ACID terms, linearizable consistency is a serial (immediate) isolation level for lightweight transactions.

In Cassandra, there are no locking or transactional dependencies when concurrently updating multiple rows or tables. Tuning availability and consistency always gives you partition tolerance. A user can pick and choose on a per operation basis how many nodes must receive a DML command or respond to a SELECT query.

For in-depth information about this new consistency level, see the article, *Lightweight transactions in Cassandra*. To support linearizable consistency, a consistency level of SERIAL has been added to Cassandra. Additions to CQL have been made to support lightweight transactions.

**Isolation**

In early versions of Cassandra, it was possible to see partial updates in a row when one user was updating the row while another user was reading that same row. For example, if one user was writing a row with two thousand columns, another user could potentially read that same row and see some of the columns, but not all of them if the write was still in progress.

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 user 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:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

**Lightweight transactions**

Lightweight transactions with linearizable consistency ensure transaction isolation level similar to the serializable level offered by RDBMS’s. They are also known as compare and set transactions. You use lightweight transactions instead of durable transactions with eventual/tunable consistency for situations the require nodes in the distributed system to agree on changes to data. For example, two users attempting to create a unique user account in the same cluster could overwrite each other’s work. Using a lightweight transaction, the nodes can agree to create only one account.

Cassandra implements lightweight transactions by extending the Paxos consensus protocol, which is based on a quorum-based algorithm. Using this protocol, a distributed system can agree on proposed data additions/modifications without the need for a master database or two-phase commit.

You use extensions in CQL for lightweight transactions. You can use an IF clause in a number of CQL statements, such as INSERT, for lightweight transactions. For example, to ensure that an insert into a new accounts table is unique for a new customer, use the IF NOT EXISTS clause:

```cql
INSERT INTO customer_account (customerID, customer_email) VALUES ('LauraS', 'lauras@gmail.com')
```
DML modifications you make using UPDATE can also make use of the IF clause by comparing one or more columns to various values:

```sql
UPDATE customer_account
SET    customer_email='laurass@gmail.com'
IF     customerID='LauraS';
```

Cassandra 2.1.1 and later support non-equal conditions for lightweight transactions. You can use <, <=, >, >=, ! = and IN operators in WHERE clauses to query lightweight tables. Behind the scenes, Cassandra is making four round trips between a node proposing a lightweight transaction and any needed replicas in the cluster to ensure proper execution so performance is affected. Consequently, reserve lightweight transactions for those situations where they are absolutely necessary; Cassandra’s normal eventual consistency can be used for everything else.

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.

### Data consistency

#### About data consistency

Consistency refers to how up-to-date and synchronized a row of Cassandra data is on all of its replicas. Cassandra extends the concept of eventual consistency by offering tunable consistency. Tunable consistency means for any given read or write operation, the client application decides how consistent the requested data must be.

Even at low consistency levels, Cassandra writes to all replicas of the partition key, even replicas in other datacenters. The consistency level determines only the number of replicas that need to acknowledge the write success to the client application. Typically, a client specifies a consistency level that is less than the replication factor specified by the keyspace. This practice ensures that the coordinating server node reports the write successful even if some replicas are down or otherwise not responsive to the write.

The read consistency level specifies how many replicas must respond to a read request before returning data to the client application. Cassandra checks the specified number of replicas for data to satisfy the read request.

The CQL documentation contains a tutorial comparing consistency levels using cqlsh tracing.

#### About built-in consistency repair features

You can use these built-in repair utilities to ensure that data remains consistent across replicas.

- Read repair
- Hinted handoff
- Anti-entropy node repair

#### Configuring data consistency

Consistency levels in Cassandra can be configured to manage availability versus data accuracy. You can configure consistency on a cluster, datacenter, or individual I/O operation basis. Consistency among participating nodes can be set globally and also controlled on a per-operation basis (for example insert or update) using Cassandra’s drivers and client libraries.

#### Write consistency levels

This table describes the write consistency levels in strongest-to-weakest order.
### Table 4: Write Consistency Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition.</td>
<td>Provides the highest consistency and the lowest availability of any other level.</td>
</tr>
<tr>
<td>EACH_QUORUM</td>
<td>Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in all datacenter.</td>
<td>Used in multiple datacenter clusters to strictly maintain consistency at the same level in each datacenter. For example, choose this level if you want a read to fail when a datacenter is down and the QORUM cannot be reached on that datacenter.</td>
</tr>
<tr>
<td>QORUM</td>
<td>A write must be written to the commit log and memtable on a quorum of replica nodes.</td>
<td>Provides strong consistency if you can tolerate some level of failure.</td>
</tr>
<tr>
<td>LOCAL_QUORUM</td>
<td>Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in the same datacenter as the coordinator. Avoids latency of inter-datacenter communication.</td>
<td>Used in multiple datacenter clusters with a rack-aware replica placement strategy, such as NetworkTopologyStrategy, and a properly configured snitch. Use to maintain consistency locally (within the single datacenter). Can be used with SimpleStrategy.</td>
</tr>
<tr>
<td>ONE</td>
<td>A write must be written to the commit log and memtable of at least one replica node.</td>
<td>Satisfies the needs of most users because consistency requirements are not stringent.</td>
</tr>
<tr>
<td>TWO</td>
<td>A write must be written to the commit log and memtable of at least two replica nodes.</td>
<td>Similar to ONE.</td>
</tr>
<tr>
<td>THREE</td>
<td>A write must be written to the commit log and memtable of at least three replica nodes.</td>
<td>Similar to TWO.</td>
</tr>
<tr>
<td>LOCAL_ONE</td>
<td>A write must be sent to, and successfully acknowledged by, at least one replica node in the local datacenter.</td>
<td>In a multiple datacenter 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 datacenter to prevent automatic connection to online nodes in other datacenters if an offline node goes down.</td>
</tr>
<tr>
<td>ANY</td>
<td>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.</td>
<td>Provides low latency and a guarantee that a write never fails. Delivers the lowest consistency and highest availability.</td>
</tr>
<tr>
<td>SERIAL</td>
<td>Achieves linearizable consistency for lightweight transactions by preventing unconditional updates.</td>
<td>You cannot configure this level as a normal consistency level, configured at the driver level using the consistency level field. You configure this level using the serial consistency field as part of the native protocol operation. See failure scenarios.</td>
</tr>
<tr>
<td>LOCAL_SERIAL</td>
<td>Same as SERIAL but confined to the datacenter. A write must be written conditionally to the commit log and memtable on a quorum of replica nodes in the same datacenter.</td>
<td>Same as SERIAL. Used for disaster recovery. See failure scenarios.</td>
</tr>
</tbody>
</table>

### SERIAL and LOCAL_SERIAL write failure scenarios

If one of three nodes is down, the Paxos commit fails under the following conditions:

- CQL query-configured consistency level of ALL
- Driver-configured serial consistency level of SERIAL
- Replication factor of 3

A WriteTimeout with a WriteType of CAS occurs and further reads do not see the write. If the node goes down in the middle of the operation instead of before the operation started, the write is committed, the value is written to the live nodes, and a WriteTimeout with a WriteType of SIMPLE occurs.

Under the same conditions, if two of the nodes are down at the beginning of the operation, the Paxos commit fails and nothing is committed. If the two nodes go down after the Paxos proposal is accepted, the write is committed to the remaining live nodes and written there, but a WriteTimeout with WriteType SIMPLE is returned.
### Read consistency levels

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

#### Table 5: Read Consistency Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Returns the record after all replicas have responded. The read operation will fail if a replica does not respond.</td>
<td>Provides the highest consistency of all levels and the lowest availability of all levels.</td>
</tr>
<tr>
<td>EACH_QUORUM</td>
<td>Not supported for reads.</td>
<td>Not supported for reads.</td>
</tr>
<tr>
<td>QUORUM</td>
<td>Returns the record after a quorum of replicas has responded from any datacenter.</td>
<td>Ensures strong consistency if you can tolerate some level of failure.</td>
</tr>
<tr>
<td>LOCAL_QUORUM</td>
<td>Returns the record after a quorum of replicas in the current datacenter as the coordinator node has reported. Avoids latency of inter-datacenter communication.</td>
<td>Used in multiple datacenter clusters with a rack-aware replica placement strategy (NetworkTopologyStrategy) and a properly configured snitch. Fails when using SimpleStrategy.</td>
</tr>
<tr>
<td>ONE</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>TWO</td>
<td>Returns the most recent data from two of the closest replicas.</td>
<td>Similar to ONE.</td>
</tr>
<tr>
<td>THREE</td>
<td>Returns the most recent data from three of the closest replicas.</td>
<td>Similar to TWO.</td>
</tr>
<tr>
<td>LOCAL_ONE</td>
<td>Returns a response from the closest replica in the local datacenter.</td>
<td>Same usage as described in the table about write consistency levels.</td>
</tr>
<tr>
<td>SERIAL</td>
<td>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, it will commit the transaction as part of the read. Similar to QUORUM.</td>
<td>To read the latest value of a column after a user has invoked a lightweight transaction to write to the column, use SERIAL. Cassandra then checks the inflight lightweight transaction for updates and, if found, returns the latest data.</td>
</tr>
<tr>
<td>LOCAL_SERIAL</td>
<td>Same as SERIAL, but confined to the datacenter. Similar to LOCAL_QUORUM.</td>
<td>Used to achieve linearizable consistency for lightweight transactions.</td>
</tr>
</tbody>
</table>

#### About the QUORUM levels

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:

\[
\text{quorum} = \left\lfloor \frac{\text{sum_of_replication_factors}}{2} \right\rfloor + 1
\]

The sum of all the replication_factor settings for each datacenter is the sum_of_replication_factors.

\[
\text{sum_of_replication_factors} = \text{datacenter1_RF} + \text{datacenter2_RF} + \ldots + \text{datacentern_RF}
\]

**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 datacenter cluster where each datacenter has a replication factor of 3, a quorum is 4 nodes. The cluster can tolerate 2 replica nodes down.
- In a five datacenter cluster where two datacenters have a replication factor of 3 and three datacenters have a replication factor of 2, a quorum is 7 nodes.

The more datacenters, the higher number of replica nodes need to respond for a successful operation.
Database internals

If consistency is a top priority, you can ensure that a read always reflects the most recent write by using the following formula:

\[(\text{nodes_written} + \text{nodes_read}) > \text{replication_factor}\]

For example, if your application is using the \text{QUORUM} consistency level for both write and read operations and you are using a replication factor of 3, then this ensures that 2 nodes are always written and 2 nodes are always read. The combination of nodes written and read (4) being greater than the replication factor (3) ensures strong read consistency.

Similar to \text{QUORUM}, the \text{LOCAL QUORUM} level is calculated based on the replication factor of the same datacenter as the coordinator node. That is, even if the cluster has more than one datacenter, the quorum is calculated only with local replica nodes.

In \text{EACH QUORUM}, every datacenter in the cluster must reach a quorum based on that datacenter’s replication factor in order for the read or write request to succeed. That is, for every datacenter 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 new cqlsh command, \text{CONSISTENCY}, to set the consistency level for queries from the current cqlsh session. The \text{WITH CONSISTENCY} clause has been removed from CQL commands. You set the consistency level programmatically (at the driver level). For example, call \text{QueryBuilder.insertInto} with a \text{setConsistencyLevel} argument. The consistency level defaults to \text{ONE} for all write and read operations.

Read requests

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

In a direct read request, the coordinator node contacts one replica node. Then the coordinator sends a digest request to a number of replicas determined by the \text{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 involved in a read query.

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 respond 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 for consistency. If they are not consistent, the replica having 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 frequently-read data, \text{read repair} is carried out to update out-of-date replicas. Read repair can be configured per table for non-\text{QUORUM} consistency levels (using \text{read_repair_chance}), and is enabled by default.

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 \text{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.
Recovering from replica node failure with rapid read protection

Examples of read consistency levels

- QUORUM in a single data center
- ONE in a single data center
- QUORUM in two data centers
- LOCAL_QUORUM in two datacenters
- ONE in two data centers
- LOCAL_ONE in two data centers

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

Example: A single datacenter cluster with a consistency level of QUORUM

In a single datacenter 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.
In a single datacenter 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.
Single datacenter cluster with 3 replica nodes and consistency set to ONE

Example: A two datacenter cluster with a consistency level of QUORUM

In a two datacenter 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 datacenter. 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.
Multiple datacenter cluster with 3 replica nodes and consistency set to QUORUM

- Datacenter Alpha
- Datacenter Beta

- Coordinator node
- Chosen node
- Read response
- Read repair

Apache Cassandra 2.1™
Example: A two datacenter cluster with a consistency level of LOCAL_QUORUM

In a multiple datacenter cluster with a replication factor of 3, and a read consistency of LOCAL_QUORUM, 2 replicas in the same datacenter 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.
Multiple datacenter cluster with 3 replica nodes and consistency set to LOCAL_QUORUM

Client

Coordinator node
Chosen node
Read response
Read repair
Example: A two datacenter cluster with a consistency level of **ONE**

In a multiple datacenter cluster with a replication factor of 3, and a read consistency of **ONE**, the closest replica for the given row, regardless of datacenter, 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.

Multiple datacenter cluster with 3 replica nodes and consistency set to **ONE**
Example: A two datacenter cluster with a consistency level of local_one

In a multiple datacenter cluster with a replication factor of 3, and a read consistency of local_one, the closest replica for the given row in the same datacenter 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.

Multiple datacenter cluster with 3 replica nodes and consistency set to local_one

Write requests

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 the write path.

For example, in a single datacenter 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 \texttt{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 \texttt{ONE} means that it is possible that 2 of the 3 replicas can miss the write if they happen to be down at the time the request is made. If a replica misses a write, the row is made consistent later using one of its built-in repair mechanisms: hinted handoff, read repair, or anti-entropy node repair.

\textbf{Multiple datacenter write requests}

In multiple datacenter deployments, Cassandra optimizes write performance by choosing one coordinator node. The coordinator node contacted by the client application forwards the write request to one replica in each of the other datacenters, with a special tag to forward the write to the other local replicas.

If using a write consistency level of \texttt{LOCAL\_ONE} or \texttt{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.
Multiple datacenter cluster with 3 replica nodes and consistency set to QUORUM

Datacenter Alpha

Datacenter Beta

Client

Nodes that make up a quorum

Coordinator node

Write response
Chapter 10. Configuration

The cassandra.yaml configuration file

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

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

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.

Values with 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 datacenter) and Initializing a multiple node cluster (multiple datacenters).

```yaml
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).
```
Configuration

- 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) 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**.

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:

- Package installations: /var/lib/cassandra/commitlog
- Tarball installations: install_location/data/commitlog

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

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

**Commonly used properties**

Properties most frequently used when configuring Cassandra.

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

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_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](blog).

**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-datacenter deployments or single-zone in public clouds. Does not recognize datacenter 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 datacenter 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.

- **PropertyFileSnitch**
  Determines proximity by rack and datacenter, 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 datacenter 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 datacenter, 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).
Configuration

Related information: Snitches

**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:**
  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) 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 datacenter (replication group). More than a single seed node per datacenter is recommended for fault tolerance. Otherwise, gossip has to communicate with another datacenter 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 datacenter).

Related information: Initializing a multiple node cluster (single datacenter) and Initializing a multiple node cluster (multiple datacenters).

**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

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

**memtable_heap_space_in_mb**
(Default: 1/4 of heap size)
Configuration

Specifies the amount of on-heap memory allocated for memtables. Cassandra uses the total of this amount and the value of `memtable_offheap_space_in_mb` to determine when to flush memtables automatically. For details, see `memtable_cleanup_threshold`.

Related information: Tuning the Java heap

**memtable_offheap_space_in_mb**

(Default: \(\frac{1}{4}\) of heap size)

Specifies the total amount of off-heap memory allocated for memtables. Cassandra uses the total of this amount and the value of `memtable_heap_space_in_mb` to determine when to flush memtables automatically. For details, see `memtable_cleanup_threshold`.

Related information: Tuning the Java heap

**concurrent_reads**

(Default: 32)\(^\text{note}\) For workloads with more data than can fit in memory, the bottleneck is reads fetching data from disk. Setting to \((16 \times \text{number} \_\text{of} \_\text{drives})\) 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)\(^\text{note}\) 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 \(8 \times \text{number} \_\text{of} \_\text{cpu} \_\text{cores}\).

**concurrent_counter_writes**

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

**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

**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

**phi_convict_threshold**

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

Related information: Failure detection and recovery

Performance tuning properties

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

**commitlog_sync**

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

- **periodic**: (Default: 10000 milliseconds [10 seconds])
  
  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)\(^\text{note}\)
  
  Used with `commitlog_sync_batch_window_in_ms` (Default: 2 ms), which is the maximum length of time that queries may be batched together.
Configuration

Related information: Durability

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

commitlog_total_space_in_mb
(Default: 32 MB for 32-bit JVMs, 8192 MB for 64-bit JVMs) Total space used for commitlogs. 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

Related information: Configuring Compaction

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) 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.

Increasing concurrent compactors impacts the available disk storage because concurrent compactions happen in parallel, especially for STCS. Ensure that adequate disk space is available when increasing this configuration.

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_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: 1/(memtable_flush_writers + 1)) Ratio used for automatic memtable flush. Cassssandra adds memtable_heap_space_in_mb to memtable_offheap_space_in_mb and multiplies the total by memtable_cleanup_threshold to get a space amount in MB. When the total amount of memory being
used by all non-flushing memtables exceeds this amount, Cassandra flushes the largest memtable to disk.

For example, consider a system in which the total of `memtable_heap_space_in_mb` and `memtable_offheap_space_in_mb` is 1000, and `memtable_cleanup_threshold` is 0.50. The "memtable_cleanup" amount is 500MB. This system has two memtables: Memtable A (150MB) and Memtable B (350MB). When either memtable increases, the total space they use exceeds 500MB. When this happens, Cassandra flushes the Memtable B to disk.

A larger value for `memtable_cleanup_threshold` means larger flushes, less frequent flushes and potentially less compaction activities, but also less concurrent flush activity, which can make it difficult to keep the disks saturated under heavy write load.

This section documents the formula used to calculate the ratio based on the number of `memtable_flush_writers`. The default value in `cassandra.yaml` is 0.11, which works if the node has many disks or if you set the node's `memtable_flush_writers` to 8. As another example, if the node uses a single SSD, the value for `memtable_cleanup_threshold` computes to 0.33, based on the minimum `memtable_flush_writers` value of 2.

- **file_cache_size_in_mb**: (Default: Smaller of 1/4 heap or 512) Total memory to use for SSTable-reading buffers.
- **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.
- **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.
- **stream_throughput_outbound_megabits_per_sec**: (Default: 200 Mbps)\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 datacenters. 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.

**Advanced properties**

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

**auto_bootstrap**
Configuration

(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 datacenter) and Initializing a multiple node cluster (multiple datacenters).

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.

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. See Ec2MultiRegionSnitch.

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.

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.

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
- OrderPreservingPartitioner (deprecated)

Related information: Partitioners

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

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.

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)\(^\text{note}\) 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: `nodetool setcachecapacity`.

**row_cache_keys_to_save**
(Default: disabled - all keys are saved)\(^\text{note}\) 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`.

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.

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

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.
Configuration

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.

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

Related information: About hinted handoff writes

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.

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
- /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
  Compresses all traffic.

- dc
  Compresses traffic between data centers.

- none
  No compression.

inter_dc_tcp_nodelay
(Default: false) Enable or disable tcp_nodelay for inter-datacenter 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) 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.

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) 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.

Settings for configuring and tuning client connections.

broadcast_rpc_address

(Default: unset) 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) 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) Sets the minimum thread pool size for remote procedure calls.

rpc_recv_buff_size_in_bytes

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

rpc_send_buff_size_in_bytes

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

**rpc_server_type**
(Default: **sync**) Cassandra provides three options for the RPC server. 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).
  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.

Settings to handle poorly performing or failing nodes.

**gc_warn_threshold_in_ms**
(Default: **1000**) Any GC pause longer than this interval is logged at the WARN level. (By default, Cassandra logs any GC pause greater than 200 ms at the INFO level.)

Additional information: Configuring logging.

**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.

**hinted_handoff_enabled**
(Default: **true**) Enable or disable hinted handoff. To enable per datacenter, 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 depends on the version:

- Prior to 1.0  
  Writes to a live replica node.

- 1.0 and later  
  Writes to the coordinator node.

Related information: About hinted handoff writes

**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 uses the maximum rate. If there are three, each node throttles to half of the maximum, since the two nodes are expected to deliver hints simultaneously.
When applying this limit, Cassandra computes the hint transmission rate based on the uncompressed hint size, even if `internode_compression` is enabled.

**max_hint_window_in_ms**

(Default: 10800000 milliseconds [3 hours]) Maximum amount of time 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

**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.

**batchlog_replay_throttle_in_kb**

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

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`) 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 `((concurrent_reads + concurrent_writes) * 2)`.

- `default_weight`: (Default: 1) 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`.

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**
Configuration

(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

internode_authenticator

(Default: enabled) Internode authentication backend. It implements org.apache.cassandra.auth.AllowAllInternodeAuthenticator to allows or disallow connections
from peer nodes.

authorizer

(Default: AllowAllAuthorizer) The authorization backend. It implements IAuthenticator 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

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

permissions_update_interval_in_ms

(Default: same value as permissions_validity_in_ms) 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 datacenters (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**: (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

**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.)

- **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

**ssl_storage_port**

(Default: 7001) The SSL port for encrypted communication. Unused unless enabled in encryption_options.

### 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

#### 1. In the cassandra.yaml file, set the following parameters:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cluster_name</strong></td>
<td>Name of the cluster that this node is joining. Must be the same for every node in the cluster.</td>
</tr>
<tr>
<td><strong>listen_address</strong></td>
<td>The IP address or hostname that Cassandra binds to for connecting to other Cassandra nodes.</td>
</tr>
<tr>
<td>(Optional) <strong>broadcast_address</strong></td>
<td>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. The default is the listen_address.</td>
</tr>
</tbody>
</table>
**Seed providers**

A seed 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 datacenter (replication group). More than a single seed node per datacenter is recommended for fault tolerance. Otherwise, gossip has to communicate with another datacenter 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 datacenter).

**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.

**Number of 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 in a subdirectory of the working, root directory when running as a service. If Cassandra does not have write permissions 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

The location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

Base the size of the directory on the value of the Java `-mx` option.

Set the location of the heap dump in the cassandra-env.sh file.

1. Open the cassandra-env.sh file for editing.
2. Scroll down to the comment about the heap dump path:

   ```
   # set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
   if [ "x$CASSANDRA_HEAPDUMP_DIR" != "x" ]; then
     JVM_OPTS="$JVM_OPTS -XX:HeapDumpPath=$CASSANDRA_HEAPDUMP_DIR/cassandra-`date +%s`-pid$.hprof"
   ```

---

The cassandra.yaml configuration file

[The cassandra.yaml file is the main configuration file for Cassandra.]
3. On the line after the comment, set the `CASSANDRA_HEAPDUMP_DIR` to the path you want to use:

```bash
# set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
export CASSANDRA_HEAPDUMP_DIR=path
if [ "x$CASSANDRA_HEAPDUMP_DIR" != "x" ]; then
    JVM_OPTS="$JVM_OPTS -XX:HeapDumpPath=$CASSANDRA_HEAPDUMP_DIR/cassandra-`date +%s`-pid$.hprof"
fi
```

4. Save the `cassandra-env.sh` file and restart.

### Configuring the buffered read size

On read workloads, Cassandra 2.1 reads drastically more data than it emits over the network. This causes problems throughout the system by wasting disk I/O and causing unnecessary garbage collection (GC).

The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/conf/cassandra-env.sh</code></td>
</tr>
</tbody>
</table>

1. Open the `cassandra-env.sh` file for editing and scroll down to the `JVM_OPTS`.

2. Set a value for the buffered read size. By default, the size is 65536.

   ```bash
   JVM_OPTS="$JVM_OPTS -Dcassandra.rar_buffer_size=65536"
   ```

### 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

1. 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.

#### Enabling virtual nodes on an existing production cluster

You cannot directly convert a single-token nodes to a vnode. However, you can configure another datacenter 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.
1. **Add a new datacenter to the cluster.**

2. Once the new datacenter with vnodes enabled is up, switch your clients to use the new datacenter.

3. **Run a full repair with nodetool repair.**

   This step ensures that after you move the client to the new datacenter that any previous writes are added to the new datacenter and that nothing else, such as hints, is dropped when you remove the old datacenter.

4. **Update your schema to no longer reference the old datacenter.**

5. **Remove the old datacenter from the cluster.**

   See *Decommissioning a datacenter.*

### 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
   ```

   Do not make all nodes seeds, see *Internode communications (gossip).*

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

   Be sure to enable encryption and authentication when using public IP's. See *Node-to-node encryption.*

   Another option is to use a custom VPN to have local, inter-region/datacenter IP's.

#### Configuring the snitch for multiple networks

External communication between the datacenters can only happen when using the `broadcast_address` (public IP).

The *GossipingPropertyFileSnitch* is recommended for production. The *cassandra-rackdc.properties* file defines the datacenters used by this snitch.

For each node in the network, specify its datacenter in *cassandra-rackdc.properties* file.

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

<table>
<thead>
<tr>
<th>Network A</th>
<th>Network B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node and datacenter:</strong></td>
<td><strong>Node and datacenter:</strong></td>
</tr>
<tr>
<td>• node0</td>
<td>• node0</td>
</tr>
<tr>
<td>dc=DC_A_cassandra</td>
<td>dc=DC_A_cassandra</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
<tr>
<td>• node1</td>
<td>• node1</td>
</tr>
<tr>
<td>dc=DC_A_cassandra</td>
<td>dc=DC_A_cassandra</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
<tr>
<td>• node2</td>
<td>• node2</td>
</tr>
<tr>
<td>dc=DC_B_cassandra</td>
<td>dc=DC_B_cassandra</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
<tr>
<td>• node3</td>
<td>• node3</td>
</tr>
<tr>
<td>dc=DC_B_cassandra</td>
<td>dc=DC_B_cassandra</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
<tr>
<td>• node4</td>
<td>• node4</td>
</tr>
<tr>
<td>dc=DC_A_analytics</td>
<td>dc=DC_A_analytics</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
<tr>
<td>• node5</td>
<td>• node5</td>
</tr>
<tr>
<td>dc=DC_A_search</td>
<td>dc=DC_A_search</td>
</tr>
<tr>
<td>rack=RAC1</td>
<td>rack=RAC1</td>
</tr>
</tbody>
</table>

**Configuring the snitch for cross-region communication in cloud implementations**

Be sure to use the appropriate snitch for your implementation. If your deploying on Amazon EC2, see the instructions in `Ec2MultiRegionSnitch`.

In cloud deployments, the region name is treated as the datacenter name and availability zones are treated as racks within a datacenter. 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 datacenter naming.)

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

For each node, specify its datacenter in the `cassandra-rackdc.properties`. The **dc_suffix** option defines the datacenters used by the snitch. Any other lines are ignored.
The location of the `cassandra.yaml` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

**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`. The `logback.xml` location for different types of installations is listed in the "File locations" section. For example, on tarball and source installations, `logback.xml` is located in the `install_location/conf` directory.

The XML configuration files look something like this:

```xml
<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>
  </appender>
  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
```
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

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.

```bash
$ nodetool getlogginglevels Logger Name Log Level ROOT INFO com.thinkaurelius.thrift ERROR
```

To add debug logging to a class permanently using the logback framework, use `nodetool setlogginglevel` to check you have the right class before you set it in the logback.xml file in `install_location/conf`. Modify to include the following line or similar at the end of the file:

```xml
<logger name="org.apache.cassandra.gms.FailureDetector" level="DEBUG"/>
```

Restart the node to invoke the change.

**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, which is located in the following directories:

- Cassandra package installations: `/etc/cassandra`
- Cassandra tarball installations: `install_location/conf`

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.

- **Archive a commitlog segment:**

<table>
<thead>
<tr>
<th>Command</th>
<th>archive_command=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td><code>%path</code></td>
<td>Fully qualified path of the segment to archive.</td>
</tr>
<tr>
<td><code>%name</code></td>
<td>Name of the commit log.</td>
</tr>
<tr>
<td>Example</td>
<td><code>archive_command=/bin/ln %path /backup/%name</code></td>
</tr>
</tbody>
</table>

- **Restore an archived commit log:**

<table>
<thead>
<tr>
<th>Command</th>
<th>restore_command=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td><code>%from</code></td>
<td>Fully qualified path of the an archived commitlog segment from the <code>restore_directories</code>.</td>
</tr>
<tr>
<td><code>%to</code></td>
<td>Name of live commit log directory.</td>
</tr>
<tr>
<td>Example</td>
<td><code>restore_command=cp -f %from %to</code></td>
</tr>
</tbody>
</table>

- **Set the restore directory location:**
### Configuration

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>restore_directories= restore_directory_location</th>
</tr>
</thead>
</table>

- Restore mutations created up to and including the specified timestamp:

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>restore_point_in_time= &lt;timestamp&gt; (YYYY:MM:DD HH:MM:SS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>restore_point_in_time=2013:12:11 17:00:00</td>
<td></td>
</tr>
</tbody>
</table>

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 datacenters in Cassandra 1.2 and later

- Single datacenter deployments: Calculate tokens by dividing the hash range by the number of nodes in the cluster.
- Multiple datacenter deployments: Calculate the tokens for each datacenter so that the hash range is evenly divided for the nodes in each datacenter.

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
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
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
- Apache Pig

You can use Cassandra 2.1 with Hadoop 2.x or 1.x with some restrictions.

- Isolate Cassandra and Hadoop nodes in separate datacenters.
- Before starting the datacenters 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, 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 datacenter 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 optional clients, MapReduce and Pig. The data is written to Cassandra and results sent back to the client.

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 introduced in Cassandra 1.1

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.
Configuration

Cassandra 2.1.1 supports using the CQLOutputFormat with Apache Pig.

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 datacenter for a cluster, even a one-node cluster, automatically. You assign nodes that run the same type of workload to the same datacenter. The separate, virtual datacenters 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 datacenter. 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/ conf/cassandra.yaml</td>
</tr>
</tbody>
</table>
Monitoring Cassandra

Getting statistics and metrics

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
- nodetool sjk
- JConsole

Using the same tools, you can perform certain administrative commands and operations such as flushing caches or doing a node repair.

Using nodetool

The nodetool utility is a command-line interface for monitoring a cluster and performing routine database operations. It is typically run from an operational node.

The nodetool utility provides commands for viewing detailed metrics for tables, server metrics, and compaction statistics:

- nodetool cfstats 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 tpsstats provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool.

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 (micros)  Write Latency (micros)  Range Latency (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
```
For a summary of the ring and its current state of general health, use `nodetool status`. For example:

```
$ nodetool status
```

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%   aaa1b7c1-6049-4a08-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
```

### Using JConsole

JConsole is a JMX-compliant tool for monitoring Java applications. JConsole is included with Sun JDK 5.0 and later. JConsole consumes the JMX metrics and operations exposed by Apache 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 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 database metrics and operations, the most important area of JConsole is the MBeans tab. This tab lists the following 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.

- `org.apache.cassandra.metrics`
  Includes metrics on CQL, clients, keyspaces, read repair, storage, 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:

![Display of MBeans in JConsole](image)

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.

**Using nodetool sjk**

Nodetool includes the open source Swiss Java Knife as a built-in command, nodetool sjk, for troubleshooting. This section focuses on using the **mx** subcommand to get and set JMX MBean values.

The `nodetool sjk mxdump` command tries to print all exposed MBeans to the console. The command can fail if it encounters an improperly formatted MBean. To interact with a specific list of MBeans, use the `nodetool sjk mx` command.

**Listing MBean names**

To get values from the command line, first get the complete name of the MBean.
Operations

Following is an example of how to use `jmxterm` to dump the full list of JMX MBeans from a database node:

```
echo "beans" | java -jar jmxterm-1.0.0-uber.jar -l localhost:7199 -n 2>&1
```

Only perform this operation once because the list can be large or search MBeans with particular names in the output by redirecting the result to another filter. For example to find MBeans that contain the word `commitlog`:

```
echo "beans" | java -jar /tmp/jmxterm-1.0.0-uber.jar -l localhost:7199 -n 2>&1 | grep -i commitlog
```

The results provide the exact MBean strings to use in `nodetool sjk mx` command.

```
com.datastax.bdp.advrep.v2.metrics:name=CommitLogMessagesRead,type=ReplicationLog
com.datastax.bdp.advrep.v2.metrics:name=CommitLogsDeleted,type=ReplicationLog
com.datastax.bdp.advrep.v2.metrics:name=CommitLogsToConsume,type=ReplicationLog
org.apache.cassandra.db:type=Commitlog
org.apache.cassandra.metrics:name=CompletedTasks,type=CommitLog
org.apache.cassandra.metrics:name=PendingTasks,type=CommitLog
org.apache.cassandra.metrics:name=TotalCommitLogSize,type=CommitLog
org.apache.cassandra.metrics:name=WaitingOnCommit,type=CommitLog
org.apache.cassandra.metrics:name=WaitingOnSegmentAllocation,type=CommitLog
```

### Getting the MBean options and values

Use the MBean information option to show available values. For example, to show the options available for `TotalCommitLogSize` from the previous results, use the complete name:

```
$ nodetool sjk mx -b "org.apache.cassandra.metrics:name=TotalCommitLogSize,type=CommitLog" -mi
```

```
org.apache.cassandra.metrics:type=CommitLog,name=TotalCommitLogSize
org.apache.cassandra.metrics.CassandraMetricsRegistry$JmxGauge
  - Information on the management interface of the MBean
  (A) Value : java.lang.Object
    - Attribute exposed for management
  (O) objectName() : javax.management.ObjectName
    - Operation exposed for management
```

The MBean has a simple value. To read the single number, run the following command:

```
$ nodetool sjk mx -b "org.apache.cassandra.metrics:name=TotalCommitLogSize,type=CommitLog" -f Value -mg
```

```
org.apache.cassandra.metrics:type=CommitLog,name=TotalCommitLogSize
67108864
```

SJK is case-sensitive, enter the names exactly.
To check on a more complex metric such as latency, find out how many values are available by using the `-mi` option:

```
$ nodetool sjk mx -b "org.apache.cassandra.metrics:name=Latency,scope=Read,type=ClientRequest" -mi
```

```
org.apache.cassandra.metrics:type=ClientRequest,scope=Read,name=Latency
org.apache.cassandra.metrics.CassandraMetricsRegistry$JmxTimer
- Information on the management interface of the MBean
  (A) Max : double
  (A) Min : double
  (A) Mean : double
  (A) StdDev : double
  (A) 95thPercentile : double
  (A) 99thPercentile : double
  (A) 99thPercentile : double
  (A) OneMinuteRate : double
  (A) FifteenMinuteRate : double
  (A) FiveMinuteRate : double
  (A) MeanRate : double
  (A) RateUnit : String
  (A) Count : long
  (A) values() : long[]
- Operation exposed for management
- Operation exposed for management
```

Then to read out a single value from this metric, use the following command:

```
$ nodetool sjk mx -b "org.apache.cassandra.metrics:name=Latency,scope=Read,type=ClientRequest" -f 99thPercentile -mg
```

```
org.apache.cassandra.metrics:type=ClientRequest,scope=Read,name=Latency
```
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 6: Compaction Metrics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemtableDataSize</td>
<td>The total size consumed by this table's data (not including metadata).</td>
</tr>
<tr>
<td>MemtableColumnsCount</td>
<td>Returns the total number of columns present in the memtable (across all keys).</td>
</tr>
<tr>
<td>MemtableSwitchCount</td>
<td>How many times the memtable has been flushed out.</td>
</tr>
<tr>
<td>RecentReadLatencyMicros</td>
<td>The average read latency since the last call to this bean.</td>
</tr>
<tr>
<td>RecentWriterLatencyMicros</td>
<td>The average write latency since the last call to this bean.</td>
</tr>
<tr>
<td>LiveSSTableCount</td>
<td>The number of live SSTables for this table.</td>
</tr>
</tbody>
</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 SSTable for a given table does not become too great.

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 7: Compaction Metrics

<table>
<thead>
<tr>
<th>Thread Pool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE_SERVICE_STAGE</td>
<td>Shows anti-entropy tasks.</td>
</tr>
<tr>
<td>CONSISTENCY-MANAGER</td>
<td>Handles the background consistency checks if they were triggered from the client's consistency level.</td>
</tr>
<tr>
<td>FLUSH-SORTER-POOL</td>
<td>Sorts flushes that have been submitted.</td>
</tr>
<tr>
<td>FLUSH-WRITER-POOL</td>
<td>Writes the sorted flushes.</td>
</tr>
<tr>
<td>GOSSIP_STAGE</td>
<td>Activity of the Gossip protocol on the ring.</td>
</tr>
<tr>
<td>LB-OPERATIONS</td>
<td>The number of load balancing operations.</td>
</tr>
<tr>
<td>LB-TARGET</td>
<td>Used by nodes leaving the ring.</td>
</tr>
<tr>
<td>MEMTABLE-POST-FLUSHER</td>
<td>Memtable flushes that are waiting to be written to the commit log.</td>
</tr>
<tr>
<td>MESSAGE-STREAMING-POOL</td>
<td>Streaming operations. Usually triggered by bootstrapping or decommissioning nodes.</td>
</tr>
<tr>
<td>MIGRATION_STAGE</td>
<td>Tasks resulting from the call of system_&quot;*&quot; methods in the API that have modified the schema.</td>
</tr>
<tr>
<td>MISC_STAGE</td>
<td></td>
</tr>
<tr>
<td>MUTATION_STAGE</td>
<td>API calls that are modifying data.</td>
</tr>
<tr>
<td>READ_STAGE</td>
<td>API calls that have read data.</td>
</tr>
<tr>
<td>RESPONSE_STAGE</td>
<td>Response tasks from other nodes to message streaming from this node.</td>
</tr>
</tbody>
</table>
Read/Write latency metrics
Cassandra tracks latency (averages and totals) of read, write, and slicing operations at the server level through StorageProxyMBean.

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 8: Compaction Metrics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompletedTasks</td>
<td>Number of completed compactions since the last start of this Cassandra instance</td>
</tr>
<tr>
<td>PendingTasks</td>
<td>Number of estimated tasks remaining to perform</td>
</tr>
<tr>
<td>ColumnFamilyInProgress</td>
<td>The table currently being compacted. This attribute is null if no compactions are in progress.</td>
</tr>
<tr>
<td>BytesTotalInProgress</td>
<td>Total number of data bytes (index and filter are not included) being compacted. This attribute is null if no compactions are in progress.</td>
</tr>
<tr>
<td>BytesCompacted</td>
<td>The progress of the current compaction. This attribute is null if no compactions are in progress.</td>
</tr>
</tbody>
</table>

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 cfstats 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.
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's in the database
eactly. The integrated cache solves 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
time) 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.

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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

1. 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,
)
```
How caching works

When both row cache and partition key cache are configured, the row cache returns results whenever possible. In the event of a row cache miss, the partition key cache might still provide a hit that makes the disk seek much more efficient. This diagram depicts two read operations on a table with both caches already populated.

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
datacenter : 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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

**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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

## Configuring compaction

As discussed in the Compaction 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 can 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): Available in Cassandra 2.0.11 and 2.1.1 and later. 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, DateTieredCompactionStrategy: Notes from the Field, and DateTieredCompactionStrategy: Compaction for Time Series Data.

When using DTCS disabling read repair 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`.

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

| Package installations                  | /etc/cassandra/cassandra.yaml |
Operations

Tarball installations

<table>
<thead>
<tr>
<th>install_location/resources/cassandra/</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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 } |
```

You can monitor the results of your configuration using compaction metrics, see Compaction metrics.

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.

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.
1. Disable compression, using CQL to set the compression parameters to an empty string.

```cql
CREATE TABLE DogTypes {
    block_id uuid,
    species text,
    alias text,
    population varint,
    PRIMARY KEY (block_id)
} 
WITH compression = { 'sstable_compression' : '' };
```

2. Enable compression on an existing table, using `ALTER TABLE` to set the compression algorithm `sstable_compression` to `LZ4Compressor` (Cassandra 1.2.2 and later), `SnappyCompressor`, or `DeflateCompressor`.

```cql
CREATE TABLE DogTypes {
    block_id uuid,
    species text,
    alias text,
    population varint,
    PRIMARY KEY (block_id)
} 
WITH compression = { 'sstable_compression' : 'LZ4Compressor' };
```

3. Change compression on an existing table, using `ALTER TABLE` and setting the compression algorithm `sstable_compression` to `DeflateCompressor`.

```cql
ALTER TABLE CatTypes
    WITH compression = { 'sstable_compression' : 'DeflateCompressor',
                      'chunk_length_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.

1. Enable write survey mode by starting a Cassandra node using the `write_survey` option.

```bash
$ bin/cassandra - Dcassandra.write_survey=true
```

This example shows how to start a tarball installation of Cassandra.
Tuning Java resources

Tuning the Java Virtual Machine (JVM) can improve performance or reduce high memory consumption.

Topics in this page:

• About garbage collection
• Determining the heap size
• How Cassandra uses memory - Read first for a better understanding of the settings and recommendations in this topic.
• Adjusting JVM parameters for other Cassandra services
• Other JMX options

About garbage collection

Garbage collection is the process by which Java removes data that is no longer needed from memory. To achieve the best performance, it is important to find the right heap size settings.

One situation that you definitely want to minimize is a garbage collection pause, also known as a stop-the-world event. A pause occurs when a region of memory is full and the JVM needs to make space to continue. During a pause all operations are suspended. Because a pause affects networking, the node can appear as down to other nodes in the cluster. Additionally, any Select and Insert statements will wait, which increases read and write latencies. Any pause of more than a second, or multiple pauses within a second that add to a large fraction of that second, should be avoided. The basic cause of the problem is the rate of data stored in memory outpaces the rate at which data can be removed. For specific symptoms and causes, contact DataStax Support.

Determining the heap size

You might be tempted to set the Java heap to consume the majority of the computer's RAM. However, this can interfere with the operation of the OS page cache. Recent operating systems maintain the OS page cache for frequently accessed data and are very good at keeping this data in memory. Properly tuning the OS page cache usually results in better performance than increasing the Cassandra row cache.

Cassandra automatically calculates the maximum heap size (MAX_HEAP_SIZE) based on this formula:

\[
\text{max}(\text{min}(1/2 \text{ ram}, 1024\text{MB}), \text{min}(1/4 \text{ ram}, 8\text{GB}))
\]

For production use, you may wish to adjust heap size for your environment using the following guidelines:

• Heap size is usually between $\frac{1}{4}$ and $\frac{1}{2}$ of system memory.
• Do not devote all memory to heap because it is also used for offheap cache and file system cache.
• Always enable GC logging when adjusting GC.
• Adjust settings gradually and test each incremental change.
• Enable parallel processing for GC.
• Cassandra's GCInspector class logs information about any garbage collection that takes longer than 200 ms. Garbage collections that occur frequently and take a moderate length of time (seconds) to complete, indicate excessive garbage collection pressure on the JVM. In addition to adjusting the garbage collection options, other remedies include adding nodes, and lowering cache sizes.

For more tuning tips, see Secret HotSpot option improving GC pauses on large heaps.

MAX_HEAP_SIZE

The recommended maximum heap size depends on which GC is used:
Hardware setup | Recommended MAX_HEAP_SIZE
--- | ---
Older computers | Typically 8 GB.
CMS for newer computers (8+ cores) with up to 256 GB RAM | No more 14 GB.

The easiest way to determine the optimum heap size for your environment is:

1. Set the maximum heap size in the cassandra-env.sh file to a high arbitrary value on a single node. For example:

   MAX_HEAP_SIZE="12G"

2. Enable GC logging.

3. Check the logs to view the heap used by that node and use that value for setting the heap size in the cluster:

   This method decreases performance for the test node, but generally does not significantly reduce cluster performance.

If you don’t see improved performance, contact the DataStax Services team for additional help.

**HEAP_NEWSIZE**

This setting determines the amount of heap memory allocated to newer objects or **young generation**. Cassandra calculates the default value for this property (in MB) as the lesser of:

- 100 times the number of cores
- ¼ of MAX_HEAP_SIZE

As a starting point, set HEAP_NEWSIZE to 100 MB per physical CPU core. For example, for a modern 8-core+ machine:

HEAP_NEWSIZE="800M"

A larger HEAP_NEWSIZE leads to longer GC pause times. For a smaller HEAP_NEWSIZE, GC pauses are shorter but usually more expensive.

**How Cassandra uses memory**

Cassandra performs the following major operations within JVM heap:

- To perform reads, Cassandra maintains the following components in heap memory:
  
  # Bloom filters
  # Partition summary
  # Partition key cache
  # Compression offsets
  # SSTable index summary

  This metadata resides in memory and is proportional to total data

- Cassandra gathers replicas for a read or for anti-entropy repair and compares the replicas in heap memory.

- Data written to Cassandra is first stored in memtables in heap memory. Memtables are flushed to SSTables on disk.

To improve performance, Cassandra also uses off-heap memory as follows:

- Page cache. Cassandra uses additional memory as page cache when reading files on disk.
Operations

- The Bloom filter and compression offset maps reside off-heap.
- Cassandra can store cached rows in native memory, outside the Java heap. This reduces JVM heap requirements, which helps keep the heap size in the sweet spot for JVM garbage collection performance.

**Adjusting JVM parameters for other Cassandra services**

- **Solr**: 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.

  ```
  # Per-thread stack size.
  JVM_OPTS="$JVM_OPTS -Xss256k"
  ```

  Also, decreasing the memtable space to make room for Solr caches can improve performance. Modify the memtable space by changing the `memtable_heap_space_in_mb` and `memtable_offheap_space_in_mb` properties in the `cassandra.yaml` file.

- **MapReduce**: Because MapReduce runs outside the JVM, changes to the JVM do not affect Analytics/Hadoop operations directly.

**Other JMX options**

Cassandra exposes other statistics and management operations via Java Management Extensions (JMX). JConsole and the nodetool utility are JMX-compliant management tools.

Configure Cassandra for JMX management by editing these properties in `cassandra-env.sh`.

- `com.sun.management.jmxremote.port`: sets the port on which Cassandra listens from JMX connections.
- `com.sun.management.jmxremote.ssl`: enables or disables SSL for JMX.
- `com.sun.management.jmxremote.authenticate`: enables or disables 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.

By default, you can interact with Cassandra using JMX on port 7199 without authentication.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/</code></td>
</tr>
<tr>
<td></td>
<td><code>conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/conf/cassandra-env.sh</code></td>
</tr>
</tbody>
</table>

**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.

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:
      
      • Cassandra Package installations:
        $ sudo service cassandra stop
      
      • Cassandra Tarball installations:
        $ sudo service cassandra stop

4. Clear the data from the peers directory:

   $ sudo rm -r /var/lib/cassandra/data/system/peers/*

   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"

     # Cassandra Package installations: /usr/share/cassandra/cassandra-env.sh
     # Cassandra Tarball installations: install_location/conf/cassandra-env.sh

6. Bring the cluster online one node at a time, starting with the seed nodes.
Operations

- Cassandra Package installations:
  
  ```
sudo service cassandra start  #Starts Cassandra
  ```

- Cassandra Tarball installations:
  
  ```
cd install_location
  ```

What's next: Remove the line you added in the cassandra-env.sh file.

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.

Repair overview

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. The hint is stored in the local system keyspace in the table `system.hints`. 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. All hints for a given replica are stored under a single partition key, so replaying hints is a simple sequential read with minimal performance impact. 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.

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 removenode 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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

**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.

In read repair, Cassandra sends a digest request to each replica not directly involved in the read. Cassandra compares all replicas and writes the most recent version to any replica node that does not have it. If the query's consistency level is above ONE, Cassandra performs this process on all replica nodes in the foreground before the data is returned to the client. Read repair repairs any node queried by the read. This means that for a consistency level of ONE, no data is repaired because no comparison takes place. For QUORUM, only the nodes that the query touches are repaired, not all nodes.

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. Build a Merkle tree for each replica
2. Compare the Merkle trees to discover differences

Merkle trees are binary hash trees whose leaves are hashes of the individual key values. The leaf of a Cassandra Merkle tree is the hash of a row value. Each Parent node higher in the tree is a hash of its respective children. Because higher nodes in the Merkle tree represent data further down the tree, Casandra can check each branch independently without requiring the coordinator node to download the entire data set. For anti-entropy repair Cassandra uses 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. Cassandra works with smaller Merkle trees because they require less storage memory and can be transferred more quickly to other nodes 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 the initiating node detects a difference, it directs the differing nodes to exchange data for the conflicting range(s). The new data is written to SSTables. The comparison begins with the top node of the Merkle tree. If Cassandra detects no difference between corresponding tree nodes, the process goes on to compares the left leaves (child nodes), then the right leaves. A difference between corresponding leaves indicates inconsistencies between the data in each replica for the data range that corresponds to that leaf. Cassandra replaces all data that corresponds to the leaves below the differing leaf with the newest version of the 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. For details, see Repair in Cassandra.

You can run the nodetool repair command on a specified node or on all nodes. The node that initiates the repair becomes the coordinator node for the operation. The coordinator node finds peer nodes with matching ranges of data and performs a major, or validation, compaction on each peer node. The validation compaction builds a Merkle tree and returns the tree to the initiating node. The initiating mode processes the Merkle trees as described.

**Full vs Incremental repair**

The process described above represents what occurs for a full repair of a node's data: Cassandra compares all SSTables for that node and makes necessary repairs. Cassandra 2.1 and later support incremental repair. An incremental repair persists data that has already been repaired, and only builds Merkle trees for unrepaired SSTables. This more efficient process depends on new metadata that marks the rows in an SSTable as repaired or unrepaired.
Operations

If you run incremental repairs frequently, the repair process works with much smaller Merkle trees. The incremental repair process works with Merkle trees as described above. Once the process had reconciled the data and built new SSTables, 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. If it does, the process just updates the SSTable’s `repairedAt` field.

Anti-compaction is handled differently, depending on the compaction strategy assigned to the data.

- Size-tiered compaction 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 performs size-tiered compaction on unrepaired data. After repair completes, Cassandra moves data from the set of unrepaired SSTables to L0.

Incremental repair works equally well with any compaction scheme.

Full repair is the default in Cassandra 2.1 and earlier.

**Parallel vs Sequential**

Sequential repair takes action on one node after another. Parallel repair repairs 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 Cassandra removes them. The coordinator node compares the Merkle trees for one replica after the other, and makes required repairs from the snapshots. For example, for a table in a keyspace with a Replication factor RF=3 and replicas A, B and C, the `repair` command takes a snapshot of each replica immediately and then repairs each replica from the snapshots sequentially (using snapshot A to repair replica B, then snapshot A to repair replica C, then snapshot B to repair replica C).

Parallel repair works on nodes A, B, and C all at once. During parallel repair, the `dynamic snitch` processes queries for this table 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.

Do not run repair using both the Sequential and the incremental options in Cassandra 2.1.

**Partitioner range ( -pr)**

Within a cluster, Cassandra stores a particular range of data on multiple nodes. If you run `nodetool repair` on one node at a time, Cassandra may repair the same range of data several times (depending on the replication factor used in the keyspace). Using the partitioner range option, `nodetool repair` only repairs a specified range.
range of data once, rather than repeating the repair operation needlessly. This decreases the strain on network
resources, although nodetool repair still builds Merkle trees for each replica.

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.

The partitioner range option is recommended for routine maintenance. Do not use it to repair a downed node.

**Local (-local, --in-local-dc) vs datacenter (dc, --in-dc) vs Cluster-wide**

Consider carefully before using nodetool repair across datacenters, instead of within a local datacenter. When
you run repair on a node using -local or --in-local-dc, the command runs only on nodes within the same
datacenter as the node that runs it. Otherwise, the command runs repair processes on all nodes that contain
replicas, even those in different datacenters. If run over multiple datacenters, nodetool repair increases
network traffic between datacenters tremendously, and can cause cluster issues. If the local option is too limited,
consider using the -dc or --in-dc options, limiting repairs to a specific datacenter. This does not repair replicas
on nodes in other datacenters, but it can decrease network traffic while repairing more nodes than the local
options.

The nodetool repair -pr option is good for repairs across multiple datacenters, as the number of replicas
in multiple datacenters can increase substantially. For example, if you start nodetool repair over two
datacenters, DC1 and DC2, each with a replication factor of 3, repair must build Merkle tables for 6 nodes. The
number of Merkle Tree increases linearly for additional datacenters.

**Additional notes for -local repairs:**

- The repair tool does not support the use of -local with the -pr option unless the datacenter's nodes have
  all the data for all ranges.

- Also, the tool does not support the use of -local with -inc (incremental repair).

For Cassandra 2.2 and later, a recommended option for repairs across datacenters: use the -dcpar or --dc-
parallel to repair datacenters in parallel.

**Endpoint range vs Subrange repair (-st, --start-token, -et --end-token)**

A repair operation runs on all partition ranges on a node, or an endpoint range, unless you use the -st and -et
(or --start-token and --end-token) options to run subrange repairs. When you specify a start token and end
token, nodetool repair works between these tokens, repairing only those partition ranges.

Subrange repair is not a good strategy because it requires generated token ranges. However, if you know which
partition has an error, you can target that partition range precisely for repair. This approach can relieve the
problem known as overstreaming, which ties up resources by sending repairs to a range over and over.

You can use subrange repair with Java to reduce overstreaming further. Send a Java describe_splits call to
ask for a split containing 32k partitions can be iterated throughout the entire range incrementally or in parallel.
Once the tokens are generated for the split, you can pass them to nodetool repair -st <start_token> -et <end_token>. Add the -local option to limit the repair to the local datacenter. This reduces cross datacenter
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.
  
  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.
Operations

- To update data on a node that has been down.
- To recover missing data or corrupted SSTables. To do this, you must run non-incremental repair.

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.
  
  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.
  
  Before you can run incremental repair, you must set the repaired state of each SSTable. For instructions, see Migrating to incremental repairs.

- 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 setcompactionthroughput) before running a repair.

Guidelines for running repair on a downed node:

- Do not use partitioner range, -pr.

Migrating to incremental repairs

Repairing SSTables using anti-entropy repair is a necessary part of Cassandra maintenance. A full repair of all SSTables on a node takes a lot of time and is resource-intensive. You can manage repairs with less service disruption using incremental repairs. Incremental repair consumes less time and resources because it skips SSTables that are already marked as repaired.

Incremental repair works equally well with any compaction scheme — Size-Tiered Compaction (STCS), Date-Tiered Compaction (DTCS) or Leveled Compaction (LCS).

However, Cassandra's default is full repair: a new SSTable is created without metadata that identifies its repaired state. Before you can start using incremental repairs, you must add this marker to each SSTable on each node in the cluster. Follow these instructions to migrate the cluster to incremental repair gradually, one node at a time.

Overview of the procedure

To migrate one Cassandra node to incremental repair:

1. Disable autocompaction on the node.
2. Run the default full, sequential repair.
3. Stop the node.
4. Mark as repaired all the SSTables that existed before you disabled compaction.

5. Restart Cassandra on the node.

6. Re-enable autocompaction on the node.

**Prerequisites:** Listing SSTables

Before you run a full repair on the node, list its SSTables. The existing SSTables may not be changed by the repair process, and the incremental repair process you run later will not process these SSTables unless you mark each one as repaired (see Step 4 below).

You can find the node’s SSTables in one of the following locations:

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

This directory contains a subdirectory for each keyspace. Each of these subdirectories contains a set of files for each SSTable. The name of the file that contains the SSTable data has the following format:

```
<version_code>-<generation>-<format>-Data.db
```

You can mark multiple SSTables as a batch by running `sstablerepairedset` with a text file of filenames — see Step 4.

**Migrating the node to incremental repair**

In RHEL and Debian installations, you must install the tools packages before you can follow these steps.

1. **Disable autocompaction on the node**
   
   From the `install_directory`:
   
   ```
   $ bin/nodetool disableautocompaction
   ```
   
   Running this command without parameters disables autocompaction for all keyspaces. For details, see `nodetool disableautocompaction`.

2. **Run the default full, sequential repair**
   
   From the `install_directory`:
   
   ```
   $ bin/nodetool repair
   ```
   
   Running this command without parameters starts a full sequential repair of all SSTables on the node. This may take a substantial amount of time. For details, see `nodetool repair`.

3. **Stop the node.**

4. **Mark as repaired all the SSTables that were created before you disabled compaction.**
Operations

Use `sstablerepairedset`. To mark a single SSTable `SSTable-example-Data.db`:

```
sudo bin/sstablerepairedset --is-repaired SSTable-example-Data.db
```

To do this as a batch process using a text file of SSTable names:

```
sudo bin/sstablerepairedset --is-repaired -f SSTable-names.txt
```

The value of the `repaired` metadata is the timestamp of the last repair. The `sstablerepairedset` command applies the current date/time. To check the value of the `repaired` metadata for an SSTable, use:

```
$ bin/sstablemetadata example-keyspace-SSTable-example-Data.db | grep "Repaired at"
```

5. Restart the node.

6. Re-enable autocompaction on the node.
   From the `install_directory`:

   ```
   $ bin/nodetool enableautocompaction
   ```

   Running this command without parameters enables autocompaction for all keyspaces and tables. For details, see `nodetool enableautocompaction`.

What's next:
After you have migrated all nodes, you will be able to run incremental repairs using `nodetool repair` with the `-inc` parameter. For details, see [https://www.datastax.com/blog/2014/02/more-efficient-repairs-21](https://www.datastax.com/blog/2014/02/more-efficient-repairs-21).

[https://www.datastax.com/blog/2014/07/repair-cassandra](https://www.datastax.com/blog/2014/07/repair-cassandra)
[https://www.datastax.com/blog/2014/02/more-efficient-repairs-21](https://www.datastax.com/blog/2014/02/more-efficient-repairs-21)
[https://www.datastax.com/blog/2014/07/anticompaction-cassandra-21](https://www.datastax.com/blog/2014/07/anticompaction-cassandra-21)

Adding or removing nodes, datacenters, 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`.

Simultaneously adding more than one new node violates LOCAL_QUORUM constraints. Data may stream from any replica in order to put data onto the new nodes, including other new nodes. Adding two or more nodes at the same time is possible but not recommended; it may introduce consistency issues. To assess the risks to your environment, see JIRA issues `CASSANDRA-2434` and `CASSANDRA-7069`.

If you do not use vnodes, see `Adding or replacing single-token nodes`.

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster.
1. Install Cassandra on the new nodes, but do not start Cassandra.
   If you used the Debian install, Cassandra starts automatically. 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 - This property is not listed in the default cassandra.yaml configuration file, but it might have been added and set to false by other operations. If it is not defined in cassandra.yaml, Cassandra uses true as a default value. For this operation, search for this property in the cassandra.yaml file. If it is present, set it to true or delete it.
   • 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.
   • seeds - Determines which nodes the new node contacts to learn about the cluster and establish the gossip process. Make sure that the -seeds list includes the address of at least one node in the existing cluster.

   This new node will not bootstrap if it is listed as a seed node. Make sure the new node's address is not listed in the -seeds list. For more information about seed nodes, see Internode communications (gossip).
   To add the new node as a seed node, complete these steps, then go on to Promoting a new node to a seed node.

   • Check the cassandra.yaml file and cassandra-topology.properties or cassandra-rackdc.properties files in other nodes in the cluster for any non-default settings, and make sure to replicate these settings on the new node.

   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:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-topology.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-topology.properties</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-rackdc.properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-rackdc.properties</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

Simultaneously bootstrapping more than one new node from the same rack, violates LOCAL_QUORUM constraints. Data may stream from any replica in order to put data onto the new nodes, including other new nodes. Adding two or more nodes at the same time is possible but not
Operations

recommended; it may introduce consistency issues. To assess the risks to your environment, see JIRA issues CASSANDRA-2434 and CASSANDRA-7069.

If you are adding two or more nodes, configure each node as in the previous steps. Then go to Starting multiple new nodes for additional steps you must take.

3. Start the single node:
   • Package installations: start Cassandra as a service
   • Tarball installations: start Cassandra as a process

4. Use nodetool status to verify that the node is fully bootstrapped and all other nodes are up (UN) and not in any other state.

5. 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.

What’s next:

Starting multiple new nodes
If you have added more than one node:
   • Make sure you start each node with consistent.rangemovement property turned off:

   **Package installations**
   On each of the nodes you are bootstrapping, add the following option to the /usr/share/cassandra/cassandra-env.sh file:

   ```
   JVM_OPTS="$JVM_OPTS -Dcassandra.consistent.rangemovement=false
   Then start Cassandra as a service.
   **Tarball installations**
   Start Cassandra on each of the nodes you are bootstrapping with this option:

   `$ bin/cassandra -Dcassandra.consistent.rangemovement=false`

   • Allow two minutes between node startups.
   • After each new node has bootstrapped, turn consistent range movement back on for each one:

   **Package installations**
   Stop Cassandra and remove the line you added to /usr/share/cassandra/cassandra-env.sh in the previous step:

   ```
   JVM_OPTS="$JVM_OPTS -Dcassandra.consistent.rangemovement=false
   Then restart Cassandra.
   **Tarball installations**
   Stop Cassandra, then restart with this option:

   `$ bin/cassandra -Dcassandra.consistent.rangemovement=false`

   • After restarting the nodes, go to back to step 4 above to verify the new nodes.

What’s next:
Promoting a node as a seed node
A seed node does not bootstrap, so a new node can't be configured as one immediately. After you have bootstrapped new nodes in the cluster, follow these steps for each one you want to promote as a seed node.

Do not promote every node in a cluster as a seed node.

1. Stop Cassandra on the node you want to promote.
2. Open the node's cassandra.yaml file and add the node's address to the seed_provider list.
3. Make this change on all other nodes in the cluster.
4. Start Cassandra as a service or a stand-alone process.

The cassandra.yaml configuration file [The cassandra.yaml file is the main configuration file for Cassandra.]
Starting Cassandra as a service [Start the Cassandra Java server process for packaged installations.]
Starting Cassandra as a stand-alone process [Start the Cassandra Java server process for tarball installations.]
Stopping Cassandra as a service [Stopping the Cassandra Java server process on packaged installations.]
Stopping Cassandra as a stand-alone process [Stop the Cassandra Java server process on tarball installations.]
About the nodetool utility [A command line interface for managing a cluster.]
Install locations [Install location topics.]

Adding a datacenter to a cluster
Steps for adding a datacenter to an existing cluster.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster.

1. Ensure that you are using NetworkTopologyStrategy for all of your keystages.
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 datacenter. 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 datacenters).
      Do not make all nodes seeds, see Internode communications (gossip).
   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: 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).
Operations

- 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
  ```java
  sethostConfig.setAutoDiscoverHosts(false);
  ```
  If you are using Astyanax, use
  ```java
  ConnectionPoolConfigurationImpl.setLocalDatacenter("<data center name>");
  ```
  to ensure you are connecting to the specified datacenter.

- 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
  ```java
  JavaDriverConfigBuilder.withLoadBalancingPolicy()
  ```
  ```java
  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.


7. After all nodes are running in the cluster:
   a. Change the keyspace properties to specify the desired replication factor for the new datacenter.
      For example, set strategy options to DC1:2, DC2:2.
      For more information, see ALTER KEYSPACE.
   b. Run `nodetool rebuild` specifying the existing datacenter on all nodes in the new datacenter:
      ```bash
      nodetool rebuild -- name_of_existing_data_center
      ```
      Otherwise, requests to the new datacenter with LOCAL_ONE or ONE consistency levels may fail if the existing datacenters 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.
      If you don't specify the existing datacenter 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 datacenter if restarted.

Install locations [Install location topics.]
The cassandra.yaml configuration file [The cassandra.yaml file is the main configuration file for Cassandra.]
Starting Cassandra as a service [Start the Cassandra Java server process for packaged installations.]
Starting Cassandra as a stand-alone process [Start the Cassandra Java server process for tarball installations.]

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

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster.

1. Confirm that the node is dead using `nodetool status`:

   The nodetool command shows a down status for the dead node (DN):

   ![nodetool status output]

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 you used the Debian/Ubuntu install, Cassandra starts automatically and you must and 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` - This property is not listed in the default `cassandra.yaml` configuration file, but it might have been added and set to `false` by other operations. If it is not defined in `cassandra.yaml`, Cassandra uses `true` as a default value. For this operation, search for this property in the `cassandra.yaml` file. If it is present, set it to `true` or delete it.
   - `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.
   - `seeds` - Determines which nodes the new node contacts to learn about the cluster and establish the gossip process. Make sure that the `-seeds` list includes the address of at least one node in the existing cluster.

   This new node will not bootstrap if it is listed as a seed node. Make sure the new node's address is not listed in the `-seeds` list. For more information about seed nodes, see **Internode communications (gossip)**.

   To add the new node as a seed node, complete these steps, then go on to **Promoting a new node to a seed node**.

5. Check the `cassandra.yaml` file and `cassandra-topology.properties` or `cassandra-rackdc.properties` files in other nodes in the cluster for any non-default settings, and make sure to replicate these settings on the new node.
Operations

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:

   - Cassandra Package installations: Add the following option to `/usr/share/cassandra/cassandra-env.sh` file:

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

   - Cassandra 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.

7. If using the `cassandra-topology.properties` wait at least 72 hours and then remove the old node's IP address from the file.

   This ensures that old node information is removed from gossip. If removed from the property file too soon, problems may result. Use `nodetool gossipinfo` to check the gossip status. The node is still in gossip until LEFT status disappears.

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

   | Package installations | /etc/cassandra/cassandra-topology.properties |
   | Tarball installations  | install_location/conf/cassandra-topology.properties |

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

   | Package installations | /etc/cassandra/cassandra-rackdc.properties |
   | Tarball installations  | install_location/conf/cassandra-rackdc.properties |

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

   | Package installations | /etc/cassandra/cassandra.yaml |
   | Tarball installations  | install_location/resources/cassandra/conf/cassandra.yaml |

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 can replace a running node in two ways:

- Adding a node and then decommissioning the old node
- Using nodetool to replace a running node

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.

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

| Package installations | /etc/cassandra/cassandra.yaml |
Operations

**Adding a node and then decommissioning the old node**

You must prepare and start the replacement node, integrate it into the cluster, and then decommission the old node.

Be sure to use the same version of Cassandra on all nodes in the cluster.

1. Prepare and start the replacement node, as described in Adding nodes to an existing cluster. If not using vnodes, see Adding or replacing single-token nodes.

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 original 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.

5. Run `nodetool cleanup` on all the other nodes in the same datacenter.

**Using nodetool to replace a running node**

This method allows you to replace a running node while avoiding streaming the data twice or running cleanup.

If using a consistency level of ONE, you risk losing data because the node might contain the only copy of a record. Be absolutely sure that no application uses consistency level ONE.

1. Stop Cassandra on the node to be replaced.

2. Follow the instructions for replacing a dead node using the old node’s IP address for `-Dcassandra.replace_address`.

3. Ensure that consistency level ONE is not used on this node.

**Removing a node** [Reduce the size of a datacenter.]

**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 datacenter.
  - 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.
Operations

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 datacenter
Steps to properly remove a datacenter so no information is lost.

1. Make sure no clients are still writing to any nodes in the datacenter.
2. Run a full repair with `nodetool repair`. This ensures that all data is propagated from the datacenter being decommissioned.
3. Change all keyspaces so they no longer reference the datacenter being removed.
4. Run `nodetool decommission` on every node in the datacenter 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.

If you are not using virtual nodes (vnodes), you must rebalance the cluster.

1. Check whether the node is up or down using `nodetool status`:

   The nodetool command shows the status of the node (UN=up, DN=down):

   ![nodetool status output]

2. 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.

3. If the node is down:
   - If the cluster uses vnodes, remove the node using the `nodetool removenode` command.
   - If the cluster does not use vnodes, before running the `nodetool removenode` command, adjust your tokens to evenly distribute the data across the remaining nodes to avoid creating a hot spot.

4. If the node does not stop streaming data to other nodes because gossip has stale state data for the node, assassinate the node.

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 datacenters 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 datacenter and you change to 5 nodes in 1 datacenter and 1 rack using a network snitch such as the GossipingPropertyFileSnitch.

- **Topology changes**
  # Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 2 datacenters using the PropertyFileSnitch.
  If splitting from one data center to two, you need to change the schema for the keyspace that are splitting. Additionally, the datacenter names must change accordingly.

  # Changing the name of a datacenter is a topology change.

1. Create a properties file with datacenter and rack information.
   - **cassandra-rackdc.properties**
     - GossipingPropertyFileSnitch, 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 | /etc/cassandra/cassandra-topology.properties |
   | Tarball installations  | install_location/conf/cassandra-topology.properties |

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

   | Package installations | /etc/cassandra/cassandra-rackdc.properties |
   | Tarball installations  | install_location/conf/cassandra-rackdc.properties |

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

   | Package installations | /etc/cassandra/cassandra.yaml |
   | Tarball installations  | install_location/resources/cassandra/conf/cassandra.yaml |

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.
Operations

The cassandra.yaml configuration file

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

Snitches

A snitch determines which datacenters and racks nodes belong to.

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.

- 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.

```
ALTER KEYSpace cycling
WITH REPLICATION = {
   'class' : 'NetworkTopologyStrategy',
   'DC1' : 3,
   'DC2' : 2
};
```

Datacenter names are case sensitive.

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.

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.
   
   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 keysaces 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) (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.

8. Remove the nodes in the old cluster.

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 datacenter. 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.

When configuring the new node in Cassandra2.1 or later, you must supply a value for the initial_token property. If you leave this property with no value, Cassandra assigns this node a random token range. Adding nodes with random token range assignments results in a badly unbalanced ring.

Adding Nodes to a Cluster

1. Install Cassandra on the new nodes, but do not start them.

2. Calculate the tokens for the nodes based on the expansion strategy you are using the Token Generating Tool.

3. Set the cassandra.yaml for the new nodes.

4. Set the initial_token according to your token calculations.

5. Start Cassandra on each new node. Allow two minutes between node initializations. You can monitor the startup and data streaming process using nodetool netstats.

6. 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.
Operations

7. 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.

   Cleanup may be safely postponed for low-usage hours.

Adding a datacenter 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 datacenter, see Generating tokens.
   - 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 datacenter. Typically one to three seeds are used per datacenter.

3. Update either the `cassandra-topology.properties (PropertyFileSnitch)` or `cassandra-rackdc.properties (GossipingPropertyFileSnitch)` on all servers to include the new nodes. You do not need to restart.

   The location of the `conf` directory depends on the type of installation:
   - Cassandra Package installations: `/etc/cassandra/conf`
   - Cassandra Tarball installations: `install_location/conf`

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

   ```java
   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 datacenters.

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 datacenter.

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:

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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

Apache Cassandra 2.1 ™
Chapter 12. 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 a memtable is flushed to disk and an SSTable is created, a hard link is copied into a /backups subdirectory of the data directory (provided JNA is enabled). Compacted SSTables will not create hard links in /backups because snapshot_before_compaction creates a new set of hardlinks before every compaction that can be used to recreate any SSTables compacted.

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.

Cassandra can only restore data from a snapshot when the table schema exists. It is recommended that you also backup the schema.

1. Run the nodetool snapshot command, specifying the hostname, JMX port, and keyspace. For example:

```bash
$ nodetool -h localhost -p 7199 snapshot mykeyspace
```

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:
Backing up and restoring data

Cassandra Package installations:

/var/lib/cassandra/data/mykeyspace/users-081a1500136111e482d09318a3b15cc2/
snapshots/1406227071618/mykeyspace-users-ka-1-Data.db

Cassandra 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.

1. 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 memtable-flushed SSTable to a backups directory under the keyspace data directory. This allows storing backups offsite without transferring entire snapshots. Also, incremental backups combined with snapshots to provide a dependable, up-to-date backup mechanism. Compacted SSTables do not create hard links in the backup folder because a snapshot will include links to SSTables that can reconstitute any compacted SSTable. A snapshot at a point in time, plus all incremental backups and commit logs since that time form a compete backup.

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

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

1. Edit the cassandra.yaml configuration file on each node in the cluster and change the value of incremental_backups to true.

   The cassandra.yaml configuration file [The cassandra.yaml file is the main configuration file for Cassandra.]

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
Backing up and restoring data

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`.

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`

  The tokens for the cluster you are restoring must match exactly the tokens of the backed-up cluster at the
time of the snapshot. Furthermore, the snapshot must be copied to the correct node with matching tokens
matching. If the tokens do not match, or the number of nodes do not match, use the `sstableloader` procedure.

- Use the Node Restart Method.

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.

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:

<table>
<thead>
<tr>
<th>Package installations</th>
<th><code>/var/lib/cassandra/commitlog</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/data/commitlog</code></td>
</tr>
</tbody>
</table>

1. To ensure that data is not lost, run `nodetool drain`. This is especially important if only a single table is
restored.

2. Shut down the node.

3. 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.
4. Delete all *.db files in data_directory_location/keyspace_name/keyspace_name-table_name directory, but DO NOT delete the /snapshots and /backups subdirectories.
   where data_directory_location is:
   - Package installations: /var/lib/cassandra/data
   - Tarball installations: install_location/data/data

5. Locate the most recent snapshot folder in this directory:
   data_directory_location/keyspace_name/table_name-UUID/snapshots/snapshot_name

6. Copy its contents into this directory:
   data_directory_location/keyspace_name/table_name-UUID directory.

7. If using incremental backups, copy all contents of this directory:
   data_directory_location/keyspace_name/table_name-UUID/backups

8. Paste it into this directory:
   data_directory_location/keyspace_name/table_name-UUID

9. Restart the node.
   Restarting causes a temporary burst of I/O activity and consumes a large amount of CPU resources.

10. Run nodetool repair.

Repairing nodes [Node repair topics.]
Starting Cassandra as a service [Start the Cassandra Java server process for packaged installations.]
Starting Cassandra as a stand-alone process [Start the Cassandra Java server process for tarball installations.]
Stopping Cassandra as a service [Stopping the Cassandra Java server process on packaged installations.]
Stopping Cassandra as a stand-alone process [Stop the Cassandra Java server process on tarball installations.]
About the nodetool utility [A command line interface for managing a cluster.]

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.

This procedure assumes you are familiar with restoring a snapshot and configuring and initializing a cluster. If not, see Initializing a cluster.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package installations</td>
<td>/etc/cassandra/cassandra.yaml</td>
</tr>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

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.
Backing up and restoring data

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.

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:
Back up and restore data

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`.

   The node serves stale data until the repair is complete.

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.

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

   | Package installations | `/etc/cassandra/cassandra.yaml` |
   | Tarball installations  | `install_location/resources/cassandra/conf/cassandra.yaml` |
Chapter 13. Cassandra tools

About the nodetool utility
The nodetool utility is a command line interface for managing a cluster.

Command formats

```
$ nodetool [options] command [args]
```

installation_location/resources/cassandra/bin

- Cassandra package installations: /etc/cassandra
- Cassandra tarball installations: install_location/conf

Main options
The following options apply to all nodetool commands:

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address.</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number.</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path.</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password.</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>Username.</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates command parameters from a list of options.</td>
</tr>
</tbody>
</table>

- 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 location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

- 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 username -pw password describering demo_keyspace
```

### Using nodetool command help

**nodetool help**

Provides a listing of nodetool commands.

**nodetool help command name**

Provides help on a specific command. For example:

```
$ nodetool help upgradesstables
```

For more information, see **nodetool help**

**nodetool cfhistograms**

Provides statistics about a table that could be used to plot a frequency function.

### Synopsis

```
$ nodetool <options> cfhistograms -- <keyspace> <table>
```

### Table 10: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is the name of a keyspace.
- table is the name of a table.

### Description

The nodetool cfhistograms command provides statistics about a table, including read/write latency, partition size, column count, and number of SSTables. The report is incremental, not cumulative. It covers all operations since the last time you ran nodetool cfhistograms in this session. The use of the metrics-core library in Cassandra 2.1 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/bin/nodetool cfhistograms libdata libout
```

Output is:

```
libdata/libout histograms
Percentile  SSTables  Write Latency  Read Latency  Partition Size  Cell
Count
```

Apache Cassandra 2.1™
Cassandra tools

<table>
<thead>
<tr>
<th>Percentile</th>
<th>(micros)</th>
<th>(micros)</th>
<th>(bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>0.00</td>
<td>39.50</td>
<td>36.00</td>
</tr>
<tr>
<td>75%</td>
<td>0.00</td>
<td>49.00</td>
<td>55.00</td>
</tr>
<tr>
<td>95%</td>
<td>0.00</td>
<td>95.00</td>
<td>82.00</td>
</tr>
<tr>
<td>98%</td>
<td>0.00</td>
<td>126.84</td>
<td>110.42</td>
</tr>
<tr>
<td>99%</td>
<td>0.00</td>
<td>155.13</td>
<td>123.71</td>
</tr>
<tr>
<td>Min</td>
<td>0.00</td>
<td>3.00</td>
<td>3.00</td>
</tr>
<tr>
<td>Max</td>
<td>0.00</td>
<td>50772.00</td>
<td>314.00</td>
</tr>
</tbody>
</table>

The output shows the percentile rank of read and write latency values, the partition size, and the cell count for the table.

**nodetool cfstats**

Provides statistics about tables.

**Synopsis**

```
$ nodetool <options> cfstats -i -- <keyspace>.<table> ... -H
```

**Table 11: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- `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)

**Description**

The `nodetool cfstats` command provides statistics about one or more tables. It's updated when SSTables change through compaction or flushing.

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 makes the output more informative and easier to understand.

This table describes the `nodetool cfstats` output.
### Table 12: nodetool cfstats output

<table>
<thead>
<tr>
<th>Name of statistic</th>
<th>Example value</th>
<th>Brief description</th>
<th>Related information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyspace</td>
<td>libdata</td>
<td>Name of the keyspace</td>
<td>Keyspace and table</td>
</tr>
<tr>
<td>Read count</td>
<td>11207</td>
<td>Number of requests to read tables in the libdata keyspace since startup</td>
<td></td>
</tr>
<tr>
<td>Read latency</td>
<td>0.047. . . ms</td>
<td>Latency reading the tables in the libdata keyspace</td>
<td></td>
</tr>
<tr>
<td>Write count</td>
<td>17598</td>
<td>Number of requests to update tables in the libdata keyspace since startup</td>
<td></td>
</tr>
<tr>
<td>Write latency</td>
<td>0.053. . . ms</td>
<td>Latency writing tables in the libdata keyspace</td>
<td></td>
</tr>
<tr>
<td>Pending tasks</td>
<td>0</td>
<td>Tasks in the queue for reads, writes, and cluster operations of tables in the keyspace</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>libout</td>
<td>Name of the Cassandra table</td>
<td></td>
</tr>
<tr>
<td>SSTable count</td>
<td>3</td>
<td>Number of SSTable containing data from the table</td>
<td></td>
</tr>
<tr>
<td>Space used (live), bytes:</td>
<td>9592399</td>
<td>Space used by the table (depends on operating system)</td>
<td></td>
</tr>
<tr>
<td>Space used (total), bytes:</td>
<td>9592399</td>
<td>Same as above</td>
<td>Same as above</td>
</tr>
<tr>
<td>Space used by snapshots (total), bytes:</td>
<td>0</td>
<td>Same occupied by backup data</td>
<td></td>
</tr>
<tr>
<td>SSTable compression ratio</td>
<td>0.367. . .</td>
<td>Fraction of data-representation size resulting from compression</td>
<td>Types of compression option)</td>
</tr>
<tr>
<td>Memtable cell count</td>
<td>1022550</td>
<td>Number of cells (storage engine rows x columns) of data in the memtable</td>
<td>Cassandra memtable structure in memory</td>
</tr>
<tr>
<td>Memtable data size, bytes:</td>
<td>32028148</td>
<td>Size of the memtable data</td>
<td>Same as above</td>
</tr>
<tr>
<td>Memtable switch count</td>
<td>3</td>
<td>Number of times a full memtable was swapped for an empty one (Increases each time the memtable for a table is flushed to disk)</td>
<td>How memtables are measured article</td>
</tr>
<tr>
<td>Local read count</td>
<td>11207</td>
<td>Number of local read requests for the libout table since startup</td>
<td></td>
</tr>
<tr>
<td>Local read latency</td>
<td>0.048 ms</td>
<td>Round trip time in milliseconds to complete a request to read the libout table</td>
<td>Factors that affect read latency</td>
</tr>
<tr>
<td>Local write count</td>
<td>17598</td>
<td>Number of local requests to update the libout the table since startup</td>
<td></td>
</tr>
<tr>
<td>Local write latency</td>
<td>0.054 ms</td>
<td>Round trip time in milliseconds to complete an update to the libout table</td>
<td>Factors that affect write latency</td>
</tr>
<tr>
<td>Pending tasks</td>
<td>0</td>
<td>Number of read, write, and cluster operations that are pending</td>
<td></td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Name of statistic</th>
<th>Example value</th>
<th>Brief description</th>
<th>Related information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloom filter false positives</td>
<td>0</td>
<td>Number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers</td>
<td>Tuning bloom filters</td>
</tr>
<tr>
<td>Bloom filter false ratio</td>
<td>0.00000</td>
<td>Fraction of all bloom filter checks resulting in a false positive</td>
<td>Same as above</td>
</tr>
<tr>
<td>Bloom filter space used, bytes</td>
<td>11688</td>
<td>Size of bytes of the bloom filter data</td>
<td>Same as above</td>
</tr>
<tr>
<td>Compacted partition minimum bytes</td>
<td>1110</td>
<td>Lower size limit for the partition being compacted in memory</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 cfstats 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 cfstats 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, bytes: 11688
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
Average live cells per slice (last five minutes): 0.0
Average tombstones per slice (last five minutes): 0.0
```

The location of the cassandra.yaml file depends on the type of installation:
nodetool cleanup
Cleans up keyspaces and partition keys no longer belonging to a node.

Synopsis

$ nodetool <options> cleanup -- <keyspace> <table> ...

Table 13: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is a keyspace name.
- table is one or more table names, separated by a space.

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.

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> ...

Table 14: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Other options are:

- `-t means the following file contains the snapshot.
- `snapshot is the name of the snapshot.
- `keyspace is one or more keyspace names, separated by a space.

**Description**

Deletes snapshots in one or more keyspaces. To remove all snapshots, omit the snapshot name.

If `clearsnapshot` fails, Cassandra retries periodically, and also when the node is shutdown or restarted.

**node tool compact**

Forces a major compaction on one or more tables.

**Synopsis**

```
$ nodetool <options> compact <keyspace> <table> ...
```

**Table 15: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Other options are:

- `keyspace is the name of a keyspace.
- `table is one or more table names, separated by a space.

**Description**

- 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 The write path to compaction and Starting compaction.

A major compaction can cause considerably more disk I/O than minor compactions.

**nodetool compactionhistory**

Provides the history of compaction operations.

**Synopsis**

```
$ nodetool <options> compactionhistory
```

**Table 16: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**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
```

<table>
<thead>
<tr>
<th>Compaction History:</th>
<th>keyspace_name</th>
<th>columnfamily_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d06f7080-07a5-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_columnfamilies</td>
</tr>
<tr>
<td>d198ae40-07a5-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>users</td>
</tr>
<tr>
<td>0381bc30-07b0-11e4-9b36-abc3a0ec9088</td>
<td>Keyspacel</td>
<td>Standard1</td>
</tr>
<tr>
<td>74eb69b0-0621-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>local</td>
</tr>
<tr>
<td>e35dd980-07ae-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>8d5cf160-07ae-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>ba376020-07af-11e4-9b36-abc3a0ec9088</td>
<td>Keyspacel</td>
<td>Standard1</td>
</tr>
<tr>
<td>d18cc760-07a5-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>libout</td>
</tr>
<tr>
<td>64009bf0-07a4-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>libout</td>
</tr>
<tr>
<td>d04700f0-07a5-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>sstable_activity</td>
</tr>
<tr>
<td>c2a97370-07a9-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>users</td>
</tr>
<tr>
<td>cb928a80-07ae-11e4-9b36-abc3a0ec9088</td>
<td>Keyspacel</td>
<td>Standard1</td>
</tr>
<tr>
<td>cdbd1540-079e-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_columns</td>
</tr>
<tr>
<td>62ced2b0-07a4-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_keyspaces</td>
</tr>
<tr>
<td>d19c3cc0-07a5-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>640bf80-07a4-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>users</td>
</tr>
<tr>
<td>6cd54e50-07ae-11e4-9b36-abc3a0ec9088</td>
<td>Keyspacel</td>
<td>Standard1</td>
</tr>
<tr>
<td>c29241f0-07a9-11e4-9b36-abc3a0ec9088</td>
<td>libdata</td>
<td>libout</td>
</tr>
<tr>
<td>c2a30a40-07a9-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>compactions_in_progress</td>
</tr>
<tr>
<td>e3a6d920-079d-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_keyspaces</td>
</tr>
<tr>
<td>62c5c40-07a4-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_columnfamilies</td>
</tr>
<tr>
<td>62b07540-07a4-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_columns</td>
</tr>
<tr>
<td>cdd038c0-079e-11e4-9b36-abc3a0ec9088</td>
<td>system</td>
<td>schema_keyspaces</td>
</tr>
<tr>
<td>b797af00-07af-11e4-9b36-abc3a0ec9088</td>
<td>Keyspacel</td>
<td>Standard1</td>
</tr>
</tbody>
</table>
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 \(1\) and 1 row taken from 3 SSTables \(3\) to make the one SSTable in that compaction operation.

<table>
<thead>
<tr>
<th>compacted_at</th>
<th>bytes_in</th>
<th>bytes_out</th>
<th>rows_merged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1404936947592</td>
<td>8096</td>
<td>7211</td>
<td>{(1,3), (3,1)}</td>
</tr>
<tr>
<td>1404936949540</td>
<td>144</td>
<td>144</td>
<td>{(1,1)}</td>
</tr>
<tr>
<td>1404941328243</td>
<td>1305838191</td>
<td>1305838191</td>
<td>{(1,467111)}</td>
</tr>
<tr>
<td>1404940844824</td>
<td>576</td>
<td>155</td>
<td>{(1,1, 2,2)}</td>
</tr>
<tr>
<td>1404940700534</td>
<td>379</td>
<td>79</td>
<td>{(2,2)}</td>
</tr>
<tr>
<td>1404941205282</td>
<td>766331398</td>
<td>766331398</td>
<td>{(1,2727158)}</td>
</tr>
<tr>
<td>1404936949462</td>
<td>8901649</td>
<td>8901649</td>
<td>{(1,9315)}</td>
</tr>
<tr>
<td>1404936336175</td>
<td>8900821</td>
<td>8900821</td>
<td>{(1,9315)}</td>
</tr>
<tr>
<td>1404936947327</td>
<td>223</td>
<td>108</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404938642471</td>
<td>144</td>
<td>144</td>
<td>{(1,1)}</td>
</tr>
<tr>
<td>1404940804904</td>
<td>383020422</td>
<td>383020422</td>
<td>{(1,1363062)}</td>
</tr>
<tr>
<td>1404933936276</td>
<td>4889</td>
<td>4177</td>
<td>{(1,4)}</td>
</tr>
<tr>
<td>1404936334171</td>
<td>441</td>
<td>281</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404936949567</td>
<td>379</td>
<td>79</td>
<td>{(2,2)}</td>
</tr>
<tr>
<td>1404936336248</td>
<td>144</td>
<td>144</td>
<td>{(1,1)}</td>
</tr>
<tr>
<td>1404940645958</td>
<td>307520780</td>
<td>307520780</td>
<td>{(1,1094380)}</td>
</tr>
<tr>
<td>1404938642319</td>
<td>8901649</td>
<td>8901649</td>
<td>{(1,9315)}</td>
</tr>
<tr>
<td>1404938642429</td>
<td>416</td>
<td>165</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404933534358</td>
<td>692</td>
<td>281</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404936334109</td>
<td>7760</td>
<td>7186</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404936333972</td>
<td>4860</td>
<td>4724</td>
<td>{(1,2, 2,1)}</td>
</tr>
<tr>
<td>1404933936715</td>
<td>441</td>
<td>281</td>
<td>{(1,3, 2,1)}</td>
</tr>
<tr>
<td>1404941200880</td>
<td>1269180898</td>
<td>1003196133</td>
<td>{(1,2623528, 2,1946565)}</td>
</tr>
<tr>
<td>1404940699201</td>
<td>297639696</td>
<td>297639696</td>
<td>{(1,1059216)}</td>
</tr>
<tr>
<td>1404940556463</td>
<td>592</td>
<td>148</td>
<td>{(1,2, 2,2)}</td>
</tr>
<tr>
<td>1404936334033</td>
<td>5760</td>
<td>5680</td>
<td>{(2,1)}</td>
</tr>
<tr>
<td>1404936947428</td>
<td>8413</td>
<td>5316</td>
<td>{(1,2, 3,1)}</td>
</tr>
<tr>
<td>1404941205571</td>
<td>429</td>
<td>42</td>
<td>{(2,2)}</td>
</tr>
<tr>
<td>1404933936584</td>
<td>7994</td>
<td>6789</td>
<td>{(1,4)}</td>
</tr>
<tr>
<td>1404940844664</td>
<td>306699417</td>
<td>306699417</td>
<td>{(1,1091457)}</td>
</tr>
<tr>
<td>1404936947746</td>
<td>601</td>
<td>281</td>
<td>{(1,3, 3,1)}</td>
</tr>
<tr>
<td>1404936947498</td>
<td>5840</td>
<td>5680</td>
<td>{(3,1)}</td>
</tr>
<tr>
<td>1404933936472</td>
<td>5861</td>
<td>5670</td>
<td>{(3,1)}</td>
</tr>
<tr>
<td>1404936336275</td>
<td>378</td>
<td>80</td>
<td>{(2,2)}</td>
</tr>
<tr>
<td>1404940556293</td>
<td>302170540</td>
<td>281000000</td>
<td>{(1,924660, 2,75340)}</td>
</tr>
</tbody>
</table>

**nodetool compactionstats**

Provide statistics about a compaction.
Synopsis

$ nodetool <options> compactionstats -H

Table 17: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- datacenter is the name of an arbitrarily chosen datacenter 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)

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

$ bin/nodetool compactionstats

pending tasks: 5

<table>
<thead>
<tr>
<th>compaction type</th>
<th>keyspace</th>
<th>table</th>
<th>completed</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bytes</td>
<td>93.43%</td>
<td>Keyspace1</td>
<td>Standard1</td>
<td>282310680</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bytes</td>
<td>19.01%</td>
<td>Keyspace1</td>
<td>Standard1</td>
<td>58457931</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Active compaction remaining time: 0h00m16s

nodetool decommission

Deactivates a node by streaming its data to another node.

Synopsis

$ nodetool <options> decommission

Table 18: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
</tbody>
</table>
Description
Causes a live node to decommission itself, streaming its data to the next node on the ring. Use netstats to monitor the progress.

`nodetool describecluster`

Provide the name, snitch, partitioner and schema version of a cluster

Synopsis

```
$ nodetool <options> describecluster -- <datacenter>
```

Table 19: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument</td>
</tr>
<tr>
<td></td>
<td></td>
<td>that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- `--` separates an option and argument that could be mistaken for a option.
- `datacenter` is the name of an arbitrarily chosen datacenter from which to select sources for streaming.

Description

Describe cluster is typically used to validate the schema after upgrading. If a schema disagreement occurs, check for and resolve schema disagreements. For help, contact DataStax Support.

Example

```
$ bin/nodetool describecluster
```

Cluster Information:
Name: Test Cluster
Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
Schema versions:
If a schema disagreement occurs, the last line of the output includes information about unreachable nodes.

$ bin/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>

**Table 20: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
- keyspace is a keyspace name.

**Example**

This example shows the sample output of the command on a three-node cluster.

$ nodetool describering demo_keyspace

```
Schema Version:1b04bd14-0324-3fc8-8bb2-9256d1e15f82
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),
```

Apache Cassandra 2.1 ™
Cassandra tools

```java
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.

```
$ bin/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> ...
```

**Table 21: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is the name of a keyspace.
- table is one or more table names, separated by a space.

**Description**

The keyspace can be followed by one or more tables.

**nodetool disablebackup**

Disables incremental backup.
Synopsis

$ nodetool <options> disablebackup

Table 22: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool disablebinary
Disables the native transport.

Synopsis

$ nodetool <options> disablebinary

Table 23: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Description
Disables the binary protocol, also known as the native transport.

nodetool disablegossip
Disables the gossip protocol.

Synopsis

$ nodetool <options> disablegossip

Table 24: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

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
```

Table 25: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

**nodetool disablethrift**

Disables the Thrift server.

**Synopsis**

```
$ nodetool <options> disablethrift
```

Table 26: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Description

The **nodetool disablethrift** command disables 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.

Page 168
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.

**Examples**

```bash
$ nodetool -u cassandra -pw cassandra disablethrift 192.168.100.1
```

**nodetool drain**

Drains the node.

**Synopsis**

```bash
$ nodetool <options> drain
```

<table>
<thead>
<tr>
<th>Table 27: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short</strong></td>
</tr>
<tr>
<td><code>-h</code></td>
</tr>
<tr>
<td><code>-p</code></td>
</tr>
<tr>
<td><code>-pw</code></td>
</tr>
<tr>
<td><code>-pwf</code></td>
</tr>
<tr>
<td><code>-u</code></td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

**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**

```bash
$ nodetool <options> enableautocompaction -- <keyspace> <table> ...
```

<table>
<thead>
<tr>
<th>Table 28: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short</strong></td>
</tr>
<tr>
<td><code>-h</code></td>
</tr>
<tr>
<td><code>-p</code></td>
</tr>
<tr>
<td><code>-pw</code></td>
</tr>
<tr>
<td><code>-pwf</code></td>
</tr>
<tr>
<td><code>-u</code></td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>
Cassandra tools

Other options are:

- keyspace is the name of a keyspace.
- table is the name of one or more keyspaces, separated by a space.

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
```

Table 29: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool enablebinary
Re-enables native transport.

Synopsis

```
$ nodetool <options> enablebinary
```

Table 30: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Description
Re-enables the binary protocol, also known as native transport.

nodetool enablegossip
Re-enables gossip.
Synopsis

$ nodetool <options> enablegossip

Table 31: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool enablehandoff

Re-enables the storing of future hints on the current node.

Synopsis

$ nodetool <options> enablehandoff

Table 32: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- `<dc-name>,<dc-name>` means enable hinted handoff only for these datacenters

nodetool enablethrift

Re-enables the Thrift server.

Synopsis

$ nodetool <options> enablethrift

Table 33: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
</tbody>
</table>
### Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**nodetool flush**

Flushes one or more tables from the memtable.

**Synopsis**

```
$ nodetool <options> flush -- <keyspace> <table> ...
```

**Table 34: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is the name of a keyspace.
- table is the name of one or more tables, separated by a space.

**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 35: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

- 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`.

**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
```

The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/conf/cassandra-env.sh</code></td>
</tr>
</tbody>
</table>

**nodetool getcompactionthreshold**

Provides the minimum and maximum compaction thresholds in megabytes for a table.

**Synopsis**

```
$ nodetool <options> getcompactionthreshold -- <keyspace> <table>
```

**Table 36: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- `keyspace` is the name of a keyspace.
- `table` is the name of a table.
nodetool getcompactionthroughput
Print the throughput cap (in MB/s) for compaction in the system.

Synopsis

$ nodetool [options] getcompactionthroughput

Table 37: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

• 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`.

Description

The `nodetool getcompactionthroughput` command prints the current compaction throughput.

Examples

$ nodetool -u cassandra -pw cassandra getcompactionthroughput

The location of the `cassandra-env.sh` file depends on the type of installation:

| Package installations | /etc/cassandra/cassandra-env.sh |
| Tarball installations  | `install_location/conf/cassandra-env.sh` |

nodetool getendpoints
Provides the IP addresses or names of replicas that own the partition key.

Synopsis

$ nodetool <options> getendpoints -- <keyspace> <table> key

Table 38: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
</tbody>
</table>
Other options are:

- keyspace is a keyspace name.
- table is a table name.
- key is the partition key of the end points you want to get.

**Example**

For example, which nodes own partition key_1, key_2, and key_3?

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.

```
$ bin/nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_1
127.0.0.2

$ bin/nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_2
127.0.0.2

$ bin/nodetool -h 127.0.0.1 -p 7100 getendpoints myks mytable key_3
127.0.0.1
```

### nodetool getlogginglevels

Get the runtime logging levels.

**Synopsis**

```
$ nodetool <options> getlogginglevels
```

<table>
<thead>
<tr>
<th>Table 39: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short</strong></td>
</tr>
<tr>
<td>-h</td>
</tr>
<tr>
<td>-p</td>
</tr>
<tr>
<td>-pwf</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool getsstables
Provides the SSTables that own the partition key.

Synopsis

```
$ nodetool <options> getsstables -- <keyspace> <table> key
```

Table 40: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is a keyspace name.
- table is a table name.
- key is the partition key of the SSTables.

nodetool getstreamthroughput
Provides the Mb (megabits) per second throughput limit for outbound streaming in the system.

Synopsis

```
$ nodetool <options> getstreamthroughput
```

Table 41: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>
nodetool gossipinfo
Provides the gossip information for the cluster.

Synopsis

```
$ nodetool <options> gossipinfo
```

Table 42: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

nodetool help
Provides nodetool command help.

Synopsis

```
$ nodetool help <command>
```

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>
```
Cassandra tools

nodetool info
Provides node information, such as load and uptime.

Synopsis

$ nodetool <options> info -T | --tokens

Table 43: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for an option.</td>
</tr>
</tbody>
</table>

Other options are:
- -T or --tokens means provide all token information.

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 44: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for an option.</td>
</tr>
</tbody>
</table>

- 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`.

**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
```

The location of the `cassandra-env.sh` file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th><code>/etc/cassandra/cassandra-env.sh</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/conf/cassandra-env.sh</code></td>
</tr>
</tbody>
</table>

**nodetool invalidatekeycache**

Resets the global key cache parameter to the default, which saves all keys.

**Synopsis**

```
$ nodetool <options> invalidatekeycache
```

**Table 45: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h</code></td>
<td><code>--host</code></td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td><code>-p</code></td>
<td><code>--port</code></td>
<td>Port number</td>
</tr>
<tr>
<td><code>-pwf</code></td>
<td><code>--password-file</code></td>
<td>Password file path</td>
</tr>
<tr>
<td><code>-pw</code></td>
<td><code>--password</code></td>
<td>Password</td>
</tr>
<tr>
<td><code>-u</code></td>
<td><code>--username</code></td>
<td>User name</td>
</tr>
<tr>
<td><code>--</code></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

By default the `key_cache_keys_to_save` is disabled in the `cassandra.yaml`. This command resets the parameter to the default.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th><code>/etc/cassandra/cassandra.yaml</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td><code>install_location/resources/cassandra/conf/cassandra.yaml</code></td>
</tr>
</tbody>
</table>

**nodetool invalidaterowcache**

Resets the global key cache parameter, `row_cache_keys_to_save`, to the default (not set), which saves all keys.
Cassandra tools

Synopsis

```
$ nodetool <options> invalidaterowcache
```

Table 46: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool join

Causes the node to join the ring.

Synopsis

```
$ nodetool <options> join
```

Table 47: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

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
```

Table 48: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Description
Available in Cassandra 2.1 and later.

Example

Snapshot Details:

<table>
<thead>
<tr>
<th>Snapshot Name</th>
<th>Keyspace</th>
<th>Column Family</th>
<th>True Size</th>
<th>Size on Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1387304478196</td>
<td>Keyspace1</td>
<td>Standard1</td>
<td>0 bytes</td>
<td>308.66 MB</td>
</tr>
<tr>
<td>1387304417755</td>
<td>Keyspace1</td>
<td>Standard1</td>
<td>0 bytes</td>
<td>107.21 MB</td>
</tr>
<tr>
<td>1387305820866</td>
<td>Keyspace1</td>
<td>Standard2</td>
<td>0 bytes</td>
<td>41.69 MB</td>
</tr>
</tbody>
</table>

nodetool move
Moves the node on the token ring to a new token.

Synopsis

$ nodetool <options> move -- <new token>

Table 49: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
- new token is a number in the range $-2^{63}$ to $+2^{63}$-1.

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.
Cassandra tools

## Synopsis

$ nodetool <options> netstats -H

### Table 50: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
</tbody>
</table>

Other options are:

- H converts bytes to a human readable form: kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). (Cassandra 2.1.1)

### 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 possible operational modes for the node are: 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 net.stats
```

The output is:

Mode: NORMAL
Not sending any streams.
Read Repair Statistics:
Attempsed: 0
Mismatch (Blocking): 0
Mismatch (Background): 0

<table>
<thead>
<tr>
<th>Pool Name</th>
<th>Active</th>
<th>Pending</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>n/a</td>
<td>0</td>
<td>1156</td>
</tr>
<tr>
<td>Responses</td>
<td>n/a</td>
<td>0</td>
<td>2750</td>
</tr>
</tbody>
</table>

**nodetool pausehandoff**
Pauses the hints delivery process.

**Synopsis**

```
$ nodetool <options> pausehandoff
```

**Table 51: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**nodetool proxyhistograms**
Provides a histogram of network statistics at the time of the command.

**Synopsis**

```
$ nodetool <options> proxyhistograms
```

**Table 52: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**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.
Cassandra tools

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
```

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Read Latency (micros)</th>
<th>Write Latency (micros)</th>
<th>Range Latency (micros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1502.50</td>
<td>375.00</td>
<td>446.00</td>
</tr>
<tr>
<td>75%</td>
<td>1714.75</td>
<td>420.00</td>
<td>498.00</td>
</tr>
<tr>
<td>95%</td>
<td>31210.25</td>
<td>507.00</td>
<td>800.20</td>
</tr>
<tr>
<td>98%</td>
<td>36365.00</td>
<td>577.36</td>
<td>948.40</td>
</tr>
<tr>
<td>99%</td>
<td>36365.00</td>
<td>740.60</td>
<td>1024.39</td>
</tr>
<tr>
<td>Min</td>
<td>616.00</td>
<td>230.00</td>
<td>311.00</td>
</tr>
<tr>
<td>Max</td>
<td>36365.00</td>
<td>55726.00</td>
<td>59247.00</td>
</tr>
</tbody>
</table>

nodetool rangekeysample
Provides the sampled keys held across all keyspaces.

Synopsis
```
$ nodetool <options> rangekeysample
```

**Table 53: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

nodetool rebuild
Rebuilds data by streaming from other nodes.

Synopsis
```
$ nodetool [(-h <host> | --host <host>)] [(-p <port> | --port <port>)] [(-pw <password> | --password <password>)] [(-pwf <passwordFilePath> | --password-file <passwordFilePath>)] [(-u <username> | --username <username>)] rebuild [--] <src-dc-name> [-dc <source_dc>]
```

**Table 54: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
</tbody>
</table>
### Cassandra tools

#### Short | Long | Description
--- | --- | ---
-pw | --password | Password
-u | --username | User name
-- |  | Separates an option from an argument that could be mistaken for a option.

Other options are:

#### Short | Long | Description
--- | --- | ---
-dc | --dcs | The name of the datacenter that Cassandra uses as the source for streaming. Cassandra rebuilds from any datacenter. If a datacenter is not specified, Cassandra chooses at random.
-ts | --tokens | Token ranges, specify in this format:
{start_token_1,end_token_1], {start_token_2,end_token_2],...
{start_token_n,end_token_n].

### Description

This command operates on multiple nodes in a cluster. The `rebuild` tool only streams data from a single source replica when rebuilding a token range.

If `rebuild` fails because some token ranges cannot be retrieved, you can rebuild selectively by using the `-ts` or `--token` option to specify a list of tokens, or one or more token ranges.

If `nodetool rebuild` is interrupted before completing its process, it cannot be restarted from the point of failure. If you run `rebuild` again, it starts from the beginning.

#### nodetool rebuild_index

Performs a full rebuild of the index for a table

### Synopsis

```
$ nodetool <options> rebuild_index -- <keyspace>.<table>.<indexName> ...
```

### Table 55: 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
-- |  | Separates an option from an argument that could be mistaken for a option.

Other options are:

- 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`
Cassandra tools

**Description**

Fully rebuilds one or more indexes for a table.

**nodetool refresh**

Loads newly placed SSTables onto the system without a restart.

**Synopsis**

```bash
$ nodetool <options> refresh -- <keyspace> <table>
```

**Table 56: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- keyspace is a keyspace name.
- table is a table name.

**nodetool reloadtriggers**

Reloads trigger classes.

**Synopsis**

```bash
$ nodetool <options> reloadtriggers
```

**Table 57: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**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>

Table 58: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Other options are:

- status provides status information.
- force forces completion of the pending removal.
- ID is the host ID, in UUID format.

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.

This command triggers cluster streaming. In large environments, the additional streaming activity causes more pending gossip tasks in the output of nodetool tpstats. Nodes can start to appear offline and may need to be restarted to clear up the back log of pending gossip tasks.

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
<p>| State=Normal/Leaving/Joining/Moving |</p>
<table>
<thead>
<tr>
<th>Address</th>
<th>Load</th>
<th>Tokens</th>
<th>Owns (effective)</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN</td>
<td>192.168.2.101</td>
<td>112.82 KB</td>
<td>256</td>
<td>31.7%</td>
</tr>
<tr>
<td>DN</td>
<td>192.168.2.103</td>
<td>91.11 KB</td>
<td>256</td>
<td>33.9%</td>
</tr>
</tbody>
</table>
Cassandra tools

$ nodetool removenode d0844a21-3698-4883-ab66-9e2fd5150edd

View the status of the operation to remove the node:

$ nodetool removenode status

RemovalStatus: No token removals in process.

Confirm that the node has been removed.

$ 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 37.7% 420129fc-0d84-42b0-be41-
ef7d3a8ad06 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.

Synopsis

$ nodetool [(-h <host> | --host <host>)] [(-p <port> | --port <port>)] [(-pw <password> | --password <password>)]
[(-pf <password-file-path> | --password-file <password-file-path>)]
[(-u <username> | --username <username>)] repair [(-<dc <specific_dc> | --in-dc
<specific_dc>)... [(-dcpar | --dc-parallel)] [(-et <end_token> | --end-token
<end_token>)] [(-hosts <specific_host> | --in-hosts <specific_host>)...]
[(-inc | --incremental)] [(-local | --in-local-dc)] [(-par | --parallel)]
[(-pr | --partitioner-range)] [(-st <start_token> | --start-token <start_token>)]
[|--] [<keyspace> <cfnames>...]

Table 59: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
• -dc, or --in-dc, followed by dc_name restricts repair to nodes in the named datacenter, which must be the local datacenter.
• -dcpar, or --dc-parallel repairs datacenters in parallel.
• -et or --end-token repairs a subset of the node’s data ending with this token. (Also specify --start-token.)
• -hosts host_name or --in-hosts host_name repairs specific hosts.
• -inc or --incremental performs an incremental repair.
• -local or --in-local-dc repairs nodes only in the local datacenter.
• -par or --parallel performs repairs in parallel.
• -pr or --partitioner-range repair only the first range returned by the partitioner.
• -st or --start-token repairs a subset of the node’s data starting with this token. (Also specify --end-token.)

keyspace is the keyspace name. The default is all.

• table is a tablename or a space-delimited list of table names. If no tables are listed, the tool operates on all tables.

Description
Performing an anti-entropy node repair on a regular basis is important, especially in an environment that deletes data frequently. The repair command repairs one or more nodes in a cluster, and provides options for restricting 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.

By default, Cassandra 2.1 does a full, sequential repair.

Using options
Use options to do these other types of repair:

• 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 -inc option for an incremental repair. An incremental repair persists already repaired SSTables and calculates the Merkle trees only for unrepaired SSTables. If you run repairs frequently, this repair process is more performant than the other types of repair even as datasets grow. Before doing an incremental repair for the first time, perform the Incremental repair migration steps.

• Use the -par option for a parallel repair. Unlike sequential repair, parallel repair constructs the Merkle trees for all nodes at the same time. Therefore, no snapshots are required (or generated). Use a parallel repair to complete the repair quickly or when you have operational downtime that allows the resources to be completely consumed during the repair.

• Use the -pr option to perform non-incremental partitioner range repairs across an entire cluster. Do not use this option for incremental repairs.

Examples
All nodetool repair arguments are optional. The following examples show the following types of repair:

# A full repair on a specific keyspace
$ nodetool repair <keyspace_name>

# An incremental, parallel repair of all keyspaces on the current node
$ nodetool repair -par -inc
Cassandra tools

# 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
$ nodetool repair -pr -hosts 10.2.2.20,10.2.2.21

# A start-point-to-end-point repair of all nodes between two nodes on the ring
$ nodetool repair -st -9223372036854775808 -et -3074457345618258603

To restrict the repair to the local datacenter, use the -dc option followed by the name of the datacenter. Issue the command from a node in the datacenter you specify in the command. If you issue the command from different datacenter, Cassandra returns an error. Do not use -dc and -pr together to repair only a local datacenter.

$ nodetool repair -dc DC1


An inspection of the system.log shows repair taking place only on IP addresses in DC1.

nodetool resetlocalschema
Reset the node's local schema and resynchronizes.

Synopsis

$ nodetool <options> resetlocalschema

Table 60: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>
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 reset.localschema
```
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
```

Table 61: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

```
nodetool ring
```
Provides node status and information about the ring.

Synopsis

```
$ nodetool <options> ring -r | --resolve-ip -- <keyspace>
```

Table 62: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
Cassandra tools

- `-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.

**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 (datacenter)**
  The datacenter 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 datacenter 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**
Creates a snapshot and then rebuilds SSTables on a node. If possible use `nodetool upgradesstables` instead of `scrub`.

`Scrub` automatically discards broken data and removes any tombstoned rows that have exceeded `gc_grace` period of the table. If partition key values do not match the column data type, the partition is considered corrupt and the process automatically stops.

When using LCS, resets all SSTables back to Level 0 and requires re-compaction of all SSTables.
Synopsis

$ nodetool main_options scrub [(-j <jobs> | --jobs <jobs>) \ ] [(--n | --no-validate)] [(--ns | --no-snapshot)] \ [(-s | --skip-corrupted)] \ [--] [<keyspace> <tables>...]

Main options

The following options apply to all nodetool commands:

Table 63: Main options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address.</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number.</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path.</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password.</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>Username.</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates command parameters from a list of options.</td>
</tr>
</tbody>
</table>

- 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 location of the cassandra-env.sh file depends on the type of installation:

| Package installations | /etc/cassandra/cassandra-env.sh |
| Tarball installations  | install_location/conf/cassandra-env.sh |

- 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 username -pw password describering demo_keyspace

Scrub parameters

Use the following parameters with the scrub command:

- -j | --jobs jobs
  Number of SSTables to simultaneously scrub. Zero (0) uses all available compaction threads.
  Default: 2.
- -n | --no-validate
  Suppresses validation of columns.
  Default: Validate all columns.
- -ns | --no-snapshot
  Suppresses creation of snapshot.
  Default: Create a snapshot before rebuilding SSTables.
- -s | --skip-corrupted

Cassandra tools

Forces scrub to skip corrupt partitions and continue. Corrupt partitions have a column value that does not match the column data type. Logs skipped partitions in the system.log.

Default: Stop scrubbing if a corrupted partition is detected.

Skipping corrupted partitions on tables with counter columns results in under-counting.

--reinsert-overflowed-ttl
Rewrite SSTables containing rows with overflowed expiration time with the maximum expiration date of 2038-01-19T03:14:06+00:00 using the original timestamp + 1 (ms).

keyspace_name table_name [...]
Identifies the keyspace and targets specific tables using a space separated list.
Default: Include all keyspaces and tables on the node when no arguments are specified.

nodetool setcachecapacity
Set global key and row cache capacities in megabytes.

Synopsis

```
$ nodetool <options> setcachecapacity -- <key-cache-capacity> <row-cache-capacity>
```

Table 64: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Other options are:

- 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.

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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>
**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>
```

**Table 65: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- 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.

**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.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

**nodetool setcompactionthreshold**

Sets minimum and maximum compaction thresholds for a table.

**Synopsis**

```
$ nodetool <options> setcompactionthreshold -- <keyspace> <table> <minthreshold> <maxthreshold>
```

**Table 66: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- 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.

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.

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>
```

Table 67: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- value_in_mb is the throughput capacity in MB per second for compaction.

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>

Table 68: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- value_in_kb/sec is the throttle time.

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 >

Table 69: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
Cassandra tools

- 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.

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 (megabits) for outbound streaming in the system, or to disable throttling.

Synopsis
```
$ nodetool <options> setstreamthroughput -- <value_in_mb>
```

Table 70: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:
value_in_mb is the throughput capacity in MB per second for streaming.

**Description**
Set value_in_mb to 0 to disable throttling.

**nodetool settraceprobability**
Sets the probability for tracing a request.

**Synopsis**

```
$ nodetool <options> settraceprobability -- <value>
```

<table>
<thead>
<tr>
<th>Table 71: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-h</td>
</tr>
<tr>
<td>-p</td>
</tr>
<tr>
<td>-pbf</td>
</tr>
<tr>
<td>-pw</td>
</tr>
<tr>
<td>-u</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

Other options are:
- value is a probability between 0 and 1.

**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.

**nodetool sjk**
Runs Swiss Java Knife (SJK) commands to execute, troubleshoot, and monitor the database using MBeans.

**Synopsis**

```
nodetool sjk options command command_options
```

**Nodetool general options**
The following options apply to all nodetool commands.

<table>
<thead>
<tr>
<th>Table 72: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>-h</td>
</tr>
</tbody>
</table>
### Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for an option.</td>
</tr>
</tbody>
</table>

#### Options

General Swiss Java Knife options.

--commands
- Displays a list of all SJK commands.

--help
- Displays command specific help, for example `nodetool sjk ttop --help`.

-X, --verbose
- Displays detailed diagnostics.

#### Commands

DataStax supports running SJK commands from nodetool.

**mxdump**

Prints serialized MBeans to JSON format.

The `nodetool sjk mxdump` command tries to print all exposed MBeans to the console. The command can fail if it encounters an improperly formatted MBean that exposes a non-serializable type. To interact with a specific list of MBeans, use the `nodetool sjk mx` command.

Run the `nodetool sjk mxdump` command with the --query switch to avoid errors if a non-serializable type (such as an mx4j MBean) is returned.

-q, --query
- Output selected MBeans by specifying the ObjectName; match patterns using the asterisk (*) and question mark (?) wildcards.

For example, to get all values for Garbage Collector:

```
nodetool sjk mxdump -q java.lang:type=GarbageCollector,name=*  
```

**gc**

Print GC (garbage collection) log messages for remote process.

For example, to print the GC log messages:

```
nodetool sjk gc  
```

No options for GC.

**hh**

Heap histogram (hh) prints the class histogram, similar to `jmap -histo`. When no options are specified, prints complete histogram for all objects.

For package installations, run nodetool with the account the runs the database.
For example, to print a histogram for young with a sample duration of 10000 milliseconds on a system with a package installation:

```
sudo -u cassandra nodetool sjk hh --young --sample-depth 10000ms -n 10
```

- **--sample-depth**
  Specify an integer and unit to set the sample duration for `--dead-young` or `--young`; default: 10000ms. To set the unit, use the following notation:
  - ms (milliseconds)
  - s (seconds)
  - m (minutes)
  - h (hours)
  - d (days)

- **--dead**
  Prints histogram of dead objects.

- **--dead-young**
  Prints histogram for sample of dead young objects. Use with `-d` to set the duration.

- **--live**
  Prints histogram of live objects.

- **-n, --top-number**
  Limits the number of buckets for `N` top buckets; default: 2147483647.

- **--young**
  Prints histogram for a sample of the new objects. Use with `-d` to set the duration.

### mx
Provides query and execution for Mbeans from the nodetool interface.

- **-all, --allMatched**
  Process all matched MBeans.

- **-a, --arguments**
  Arguments for MBean operation invocation.

- **-f, --field, --attribute**
  MBean attribute

- **-b, --bean**
  Mbean name.

- **-mc, --call**
  Invokes MBean method. Default false.

- **-mg, --get**
  Retrieves value of MBean attribute. Default false.

- **-mi, --info**
  Display metadata for MBean. Default false.

- **--max-col-width**
  Table column width threshold for formatting tabular data. Default 40.

- **-op, --operation**
  MBean operation name to be called.

- **--quiet**
  Avoid non-essential output; default: false.

- **-ms, --set**
  Sets value for MBean attribute; default: false.

- **-v, --value**
  Value to set for MBean attribute.

### jps
Enhanced version of `jps` tool from the Java Development Kit (JDK) that lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system.
Cassandra tools

- **fd, --filter-description**
  Use a wildcard expression to match the process description.

- **fp, --filter-property**
  Use a wildcard expressions to match JVM system properties.

- **pd, --process-details**
  Prints custom information related to a process; set to PID, MAIN, FDQN_MAIN, ARGS, Dsys-prop, or Dsys-prop.

- **Xjvm-flag**
  Prints custom information related to the specified JVM flag.

stcap
Stack Capture dumps stack traces to file for further processing.

- **-e, --empty**
  Set to true to retain threads without stack trace in dump (ignored by default); default: false.

- **-f, --filter**
  Filters threads by name using Java RegEx syntax; default: .* The default value, period (.) followed by asterisk (*), allows any number of any characters, including nulls.

- **-l, --limit**
  Target number of traces to collect, once reached command terminates where 0 is unlimited; default: 0.

- **-m, --match-frame**
  Set to filtering string and only traces containing are included in the dump.

- **-o, --output**
  Write the thread dump to the specified file name.

- **-r, --rotate**
  When specified output file would be rotated every N traces. Specify 0 for no rotation; default: 0.

- **-s, --sampler-interval**
  Interval between polling MBeans; default: 0.

- **-t, --timeout**
  Time until command terminate even without enough traces collected; default: 30000.

stcpy
Stack Copy utility copies and filters dumps.

- **-e, --empty**
  Set to true to retain threads without stack trace in dump (ignored by default); default: false.

- **-i, --input**
  Input files.

- **--mask**
  One or more masking rules, for example com.mycompany:com.somecompany; default: null.

- **-m, --match-frame**
  Frame filter, only traces containing the string are included in dump.

- **-o, --output**
  Set to filename. Writes thread dump to the specified file.

- **-ss, --subsample**
  If below 1.0 some frames are randomly thrown away. For example, 0.1 retains only every 10th frame; default: 1.0.

- **-tf, --thread-filter**
  Filter threads by name using Java RegEx syntax; default: .*

ssa
Stack Sample Analyzer for stack trace dumps.

- **--categorize**
  Prints summary for provided categorization; no summary printed by default.

- **-cf, --categorizer-file**
  Path to file with stack trace categorization definition.

- **-co, --csv-output**
Output data in CSV format; default: false.

-\texttt{f}, --file
Path to stack dump file.

--flame
Exports flame graph to SVG format; default: not included.

--histo
Prints frame histogram; default: not included.

-nc, --named-class
Defines name stack trace. Use with other options.

\textbf{classes}
Use name=filter expression notation; default: no expression.

--print
Print traces from file; default: not included.

-rc, --rainbow
List of filters for rainbow coloring.

--ssa-help
Provides additional information about SSA options; default: not shown.

-si, --summary-info
Lists summaries.

--thread-info
Prints per thread information summary; default: false.

-tn, --thread-name
Filters thread name using Java RegEx syntax.

-tr, --time-range
Filters time range.

-tz, --time-zone
Time zone used for timestamps; default: UTC. Use --ssa-help for timezone notation.

--title
Inserts a flame graph title; default: Flame Graph.

-tf, --trace-filter
Applies filter to traces before processing. Use --ssa-help for more details about filter notation.

-\texttt{tt}, --trace-trim
Positional filter trim frames to process. Use --ssa-help for more details about trace filters.

--width
Flame graph width in pixels; default: 1200.

\texttt{ttop}
Displays top threads from JVM process.

-\texttt{f}, --filter
Wild card expression to filter threads by name.

-o, --order
Sort order. Value tags: CPU, USER, SYS, ALLOC, NAME; default: CPU.

-ri, --report-interval
Interval between CPU usage reports; default: 10000.

-si, --sampler-interval
Interval between polling MBeans; default: 500.

-n, --top-number
Number of threads to show; default: 20.

\textbf{nodetool snapshot}
Take a snapshot of one or more keyspaces, or of a table, to backup data.
Cassandra tools

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> ... )

Table 73: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- -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.
- 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.

Description

Use this command to backup and restore 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.

Example: All keyspaces

Take a snapshot of all keyspaces on the node. On Linux, in the Cassandra bin directory, for example:

$ bin/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.
Example: Single keyspace snapshot
Assuming you created the keyspace and tables in the music service example, take a snapshot of the music keyspace and name the snapshot 2014.06.24.

$ bin/nodetool snapshot -t 2014.06.24 music

The following message appears:
Requested creating snapshot(s) for [music] with snapshot name [2014.06.24]
Snapshot directory: 2014.06.24

Assuming the music keyspace contains two tables, songs and playlists, taking a snapshot of the keyspace creates multiple snapshot directories named 2014.06.24. A number of .db files containing the data are located in these directories. For example, from the installation directory:

$ cd data/data/music/playlists-bf8118508cfd11e3972273ded3cb6170/snapshots/1404936753154 $ ls
music-playlists-ka-1-CompressionInfo.db  music-playlists-ka-1-Index.db  music-playlists-ka-1-TOC.txt
music-playlists-ka-1-Data.db  music-playlists-ka-1-Statistics.db
music-playlists-ka-1-Filter.db  music-playlists-ka-1-Summary.db

$ cd data/data/music/songs-b8e385a08cfd11e3972273ded3cb6170/2014.06.24/snapshots/1404936753154
music-songs-ka-1-CompressionInfo.db  music-songs-ka-1-Index.db  music-songs-ka-1-TOC.txt
music-songs-ka-1-Data.db  music-songs-ka-1-Statistics.db
music-songs-ka-1-Filter.db  music-songs-ka-1-Summary.db

Example: Multiple keyspaces snapshot
Assuming you created a keyspace named mykeyspace in addition to the music keyspace, take a snapshot of both keyspaces.

$ bin/nodetool snapshot mykeyspace music

The following message appears:
Requested creating snapshot(s) for [mykeyspace, music] with snapshot name [1391460334889]
Snapshot directory: 1391460334889

Example: Single table snapshot
Take a snapshot of only the playlists table in the music keyspace.

$ bin/nodetool snapshot -cf playlists music

Requested creating snapshot(s) for [music] with snapshot name [1391461910600]
Cassandra tools

Snapshot directory: 1391461910600

Cassandra creates the snapshot directory named 1391461910600 that contains the backup data of playlists table in data/data/music/playlists-bf8118508cf0d60023ded3cb670/snapshots, for example.

Example: List of different keyspace.tables snapshot

Take a snapshot of several tables in different keyspaces, such as the playlists table in the music keyspace and the users table in the test keyspace. The keyspace.table list should be comma-delimited with no spaces.

$ bin/nodetool snapshot -kt music.playlists,test.users

Requested creating snapshot(s) for [music.playlists,test.users] 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>

Table 74: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for an option.</td>
</tr>
</tbody>
</table>

Other options are:

- -r, or --resolve-ip, means to provide node names instead of IP addresses.
- keyspace is a keyspace name.

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
Cassandra tools

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 datacenter times the replication factor. For example, a node can own 33% of the ring, but show 100% if the replication factor is 3.

  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% aaa1b7c1-6049-4a08-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-4306-afdf-5c1e95b8732e rack1
```

nodetool statusbackup

Synopsis

```
$ nodetool <options> statusbackup
```

Table 75: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Description

Provides the status of incremental backup.

nodetool statusbinary

Provide the status of native transport.
Cassandra tools

Synopsis

$ nodetool <options> statusbinary

Table 76: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Description

Provides the status of the binary protocol, also known as the native transport.

nodetool statusgossip

Synopsis

$ nodetool <options> statusgossip

Table 77: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Description

Provides the status of gossip.

nodetool statushandoff

Synopsis

$ nodetool <options> statushandoff

Table 78: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
</tbody>
</table>
### Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Provides the status of hinted handoff.

**nodetool statusthrift**

Provide the status of the Thrift server.

**Synopsis**

```
$ nodetool <options> statusthrift
```

**Table 79: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

**nodetool stop**

Stops the compaction process.

**Synopsis**

```
$ nodetool <options> stop -- <compaction_type>
```

**Table 80: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

Other options are:

- A compaction type: COMPACTION, VALIDATION, CLEANUP, SCRUB, INDEX_BUILD
Cassandra tools

Description
Stops an operation 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.

**nodetool stopdaemon**
Stops the cassandra daemon.

Synopsis

```
$ nodetool <options> stopdaemon
```

Table 81: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

**nodetool toppartitions**
Synopsis

```
$ nodetool <options> toppartitions <keyspace> <table> <duration> ...
```

Table 82: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>

Other options are:

- `-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
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
```

**Table 83: Options**

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
</tbody>
</table>
### 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 when it exists on the same node. The queues can back up if executing at the next stage is too busy and cause performance bottlenecks.

The `nodetool tpstats` command provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool. It's updated when SSTables change through compaction or flushing.

Run the `nodetool tpstats` command on a local node to get thread pool statistics. This table describes key indicators:

| Table 84: nodetool tpstats output |

<table>
<thead>
<tr>
<th>Name of statistic</th>
<th>Task</th>
<th>Related information</th>
</tr>
</thead>
<tbody>
<tr>
<td>AntiEntropyStage</td>
<td>Repair consistency</td>
<td></td>
</tr>
<tr>
<td>CacheCleanupExecutor</td>
<td>Clears the cache</td>
<td></td>
</tr>
<tr>
<td>CommitlogArchiver</td>
<td>Archives commitlog</td>
<td></td>
</tr>
<tr>
<td>CompactionExecutor</td>
<td>Runs compaction</td>
<td></td>
</tr>
<tr>
<td>CounterMutationStage</td>
<td>Local counter changes</td>
<td>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.</td>
</tr>
<tr>
<td>GossipStage</td>
<td>Handle gossip rounds every second</td>
<td>Out of sync schemas can cause issues. <code>nodetool resetlocalschema</code> may need to be used.</td>
</tr>
<tr>
<td>HintedHandoff</td>
<td>Send missed mutations to other nodes</td>
<td>Usually symptom of a problem elsewhere. Use <code>nodetool disablehandoff</code> and run repair.</td>
</tr>
<tr>
<td>InternalResponseStage</td>
<td>Respond to non-client initiated messages, including bootstrapping and schema checking</td>
<td></td>
</tr>
<tr>
<td>MemtableFlushWriter</td>
<td>Writes memtable contents to disk</td>
<td>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. <code>nodetool tpstats</code> does not report blocked threads in the MemtableFlushWriter pool.</td>
</tr>
<tr>
<td>MemtablePostFlush</td>
<td>Operations after flushing the memtable</td>
<td>Discard commit log files and flush secondary indexes.</td>
</tr>
<tr>
<td>MemtableReclaimMemory</td>
<td>Makes unused memory available</td>
<td></td>
</tr>
<tr>
<td>MigrationStage</td>
<td>Make schema changes</td>
<td></td>
</tr>
<tr>
<td>MiscStage</td>
<td>Miscellaneous operations</td>
<td>Snapshotting, replicating data after node remove completed.</td>
</tr>
<tr>
<td>Name of statistic</td>
<td>Task</td>
<td>Related information</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MutationStage</td>
<td>Local writes</td>
<td>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.</td>
</tr>
<tr>
<td>Native-Transport-Requests</td>
<td>Requests to the server using the CQL Native Protocol</td>
<td></td>
</tr>
<tr>
<td>PendingRangeCalculator</td>
<td>Calculate pending ranges per bootstraps and departed nodes</td>
<td>Developer notes</td>
</tr>
<tr>
<td>ReadRepairStage</td>
<td>A digest query and update of replicas of a key</td>
<td>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 <code>read_repair_chance</code> value, like 0.11.</td>
</tr>
<tr>
<td>ReadStage</td>
<td>Local reads</td>
<td>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.</td>
</tr>
<tr>
<td>RequestResponseStage</td>
<td>Handle responses from other nodes</td>
<td></td>
</tr>
<tr>
<td>ValidationExecutor</td>
<td>Validates schema</td>
<td></td>
</tr>
</tbody>
</table>

Table 85: Droppable Messages

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Stage</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>n/a</td>
<td>This is deprecated and no longer has any use</td>
</tr>
<tr>
<td>_TRACE</td>
<td>n/a (special)</td>
<td>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</td>
</tr>
<tr>
<td>MUTATION</td>
<td>MutationStage</td>
<td>If a write message is processed after its timeout <code>write_request_timeout_in_ms</code> 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.</td>
</tr>
<tr>
<td>COUNTER_MUTATION</td>
<td>MutationStage</td>
<td>If a write message is processed after its timeout <code>write_request_timeout_in_ms</code> 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.</td>
</tr>
<tr>
<td>READ_REPAIR</td>
<td>MutationStage</td>
<td>Times out after <code>write_request_timeout_in_ms</code></td>
</tr>
<tr>
<td>READ</td>
<td>ReadStage</td>
<td>Times out after <code>read_request_timeout_in_ms</code>. No point in servicing reads after that point since it would of returned error to client</td>
</tr>
<tr>
<td>RANGE_SLICE</td>
<td>ReadStage</td>
<td>Times out after <code>range_request_timeout_in_ms</code></td>
</tr>
<tr>
<td>PAGED_RANGE</td>
<td>ReadStage</td>
<td>Times out after <code>request_timeout_in_ms</code>.</td>
</tr>
<tr>
<td>REQUEST_RESPONSE</td>
<td>RequestResponseStage</td>
<td>Times out after <code>request_timeout_in_ms</code>. Response was completed and sent back but not before the timeout</td>
</tr>
</tbody>
</table>
Cassandra tools

Example

Run the command every two seconds.

$ nodetool -h labcluster tpstats

Example output is:

<table>
<thead>
<tr>
<th>Pool Name</th>
<th>Active</th>
<th>Pending</th>
<th>Completed</th>
<th>Blocked</th>
<th>All time blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterMutationStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadStage</td>
<td>0</td>
<td>0</td>
<td>103</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RequestResponseStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MutationStage</td>
<td>0</td>
<td>0</td>
<td>13234794</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadRepairStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GossipStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CacheCleanupExecutor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AntiEntropyStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MigrationStage</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ValidationExecutor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CommitLogArchiver</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MiscStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtableFlushWriter</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtableReclaimMemory</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PendingRangeCalculator</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtablePostFlush</td>
<td>0</td>
<td>0</td>
<td>1468</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CompactionExecutor</td>
<td>0</td>
<td>0</td>
<td>254</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>InternalResponseStage</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HintedHandoff</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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> ...

Table 86: Options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--host</td>
<td>Hostname or IP address</td>
</tr>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
</tr>
</tbody>
</table>
Other options are:

- endpoint is one or more endpoint IP addresses or host names which hints are to be deleted.

**nodetool upgradesstables**
Rewrites older SSTables to the current version of Cassandra.

**Synopsis**

```
$ nodetool <options> upgradesstables -a | --include-all-sstables -- <keyspace> <table> ...
```

<table>
<thead>
<tr>
<th>Table 87: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short</strong></td>
</tr>
<tr>
<td>--h</td>
</tr>
<tr>
<td>--p</td>
</tr>
<tr>
<td>--pwf</td>
</tr>
<tr>
<td>--pw</td>
</tr>
<tr>
<td>--u</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

Other options are:
- -a or --include-all-sstables includes all SSTables, even those with the current settings. See **Examples** below.
- keyspace is the keyspace name.
- table is one or more table names, separated by a space.

**Description**
Rewrites SSTables on a node that are incompatible with the current version. Use this command when upgrading your server or changing compression options.

**Examples**

```
$ upgradesstables
    Reads only SSTables created by old major versions of Cassandra and re-writes them to the current version one at a time.
```

```
$ upgradesstables -a
    Reads all existing SSTables and re-writes them to the current Cassandra version one at a time.
```

```
$ upgradesstables -a keyspace table
    Reads specific SSTables and re-writes them to the current Cassandra version one at a time.
```

**nodetool version**
Provides the version number of Cassandra running on the specified node.

**Synopsis**

```
$ nodetool <options> version
```

<table>
<thead>
<tr>
<th>Table 88: Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Short</strong></td>
</tr>
<tr>
<td>--h</td>
</tr>
</tbody>
</table>
Cassandra tools

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>--port</td>
<td>Port number</td>
</tr>
<tr>
<td>-pwf</td>
<td>--password-file</td>
<td>Password file path</td>
</tr>
<tr>
<td>-pw</td>
<td>--password</td>
<td>Password</td>
</tr>
<tr>
<td>-u</td>
<td>--username</td>
<td>User name</td>
</tr>
<tr>
<td>--</td>
<td>Separates an option from an argument that could be mistaken for a option.</td>
<td></td>
</tr>
</tbody>
</table>

The cassandra utility

Cassandra start-up parameters can be run from the command line (Tarball installations) or specified in the cassandra-env.sh file (Package or Tarball installations).

You can also use the cassandra-env.sh file to pass additional options, such as maximum and minimum heap size, to the Java virtual machine rather than setting them in the environment.

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>

The location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

Usage

Add the following to the cassandra-env.sh file:

```
JVM_OPTS="$JVM_OPTS -D[PARAMETER]"
```

The location of the cassandra-env.sh file depends on the type of installation:

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra-env.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/conf/cassandra-env.sh</td>
</tr>
</tbody>
</table>

For Tarball installations, you can run this tool from the command line:

```
$ cassandra [OPTIONS]
```

Examples:

- `cassandra-env.sh`: JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"
- Command line: `bin/cassandra -Dcassandra.load_ring_state=false`

The Example section contains more examples.
Command line only options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-f</code></td>
<td>Start the cassandra process in foreground. The default is to start as background process.</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Help.</td>
</tr>
<tr>
<td><code>-p filename</code></td>
<td>Log the process ID in the named file. Useful for stopping Cassandra by killing its PID.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Print the version and exit.</td>
</tr>
</tbody>
</table>

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.ignore_dynamic_snitch_severity=true/false (Default: false)`
  Setting this property to `true` causes the dynamic snitch to ignore the severity indicator from gossip when scoring nodes. Future versions will default to `true` and this setting will be removed. See Failure detection and recovery and Dynamic snitching in Cassandra: past, present, and future.

- `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)

  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.prepared_statements_cache_size_in_bytes=cache_size`
  Set the cache size for prepared statements, see the Cassandra driver documentation for more information.

- `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.

  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.rpc_port=port`
  Set the port for the Thrift RPC service, which is used for client connections. (Default: 9160).
Cassandra tools

**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 trigger JARs. (Default: conf/triggers)

**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.

**consistent.rangemovement=true**
True makes Cassandra 2.1 bootstrapping behavior effective. False makes Cassandra 2.0 behavior effective.

Clear gossip state when starting a node:
- Command line: `bin/cassandra -Dcassandra.load_ring_state=false`
- `cassandra-env.sh`: `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`
- `cassandra-env.sh`: `JVM_OPTS="$JVM_OPTS -Dcassandra.join_ring=false"

Replacing a dead node:
- Command line: `bin/cassandra -Dcassandra.replace_address=10.91.176.160`
- `cassandra-env.sh`: `JVM_OPTS="$JVM_OPTS -Dcassandra.replace_address=10.91.176.160"

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

The `cassandra-stress` tool
The `cassandra-stress` tool is a Java-based stress testing utility for basic benchmarking and load testing a Cassandra cluster.

The choices you make when data modeling your application can make a big difference in how your application performs. Creating the best data model requires significant load testing and multiple iterations. The `cassandra-stress` tool helps you in this endeavor by populating your cluster and supporting stress testing of arbitrary CQL tables and arbitrary queries on tables. 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.

This tool works for Cassandra 2.0 and later clusters.
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:

- **Cassandra Package installations**: `/usr/share/doc/cassandra/examples`
- **Cassandra Tarball installations**: `install_location/tools`

For a complete description on using these sample files, see [Improved Cassandra 2.1 Stress Tool: Benchmark Any Schema – Part 1](#).

The `cassandra-stress` tool creates a keyspace called `keyspace1` and within that, tables named `standard1` or `counter1`, depending on what type of table is being tested. These are automatically created the first time you run the stress test and are reused on subsequent runs unless you drop the keyspace using CQL. You cannot change the names; they are hard-coded.

**Usage:**

- **Cassandra Package installations**:
  ```
  $ /usr/bin/cassandra-stress command [options]
  ```

- **Cassandra 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>Multiple concurrent reads. The cluster must first be populated by a write test.</td>
</tr>
<tr>
<td>write</td>
<td>Multiple concurrent writes against the cluster.</td>
</tr>
<tr>
<td>mixed</td>
<td>Interleave basic commands with configurable ratio and distribution. The cluster must first be populated by a write test.</td>
</tr>
<tr>
<td>counter_write</td>
<td>Multiple concurrent updates of counters.</td>
</tr>
<tr>
<td>counter_read</td>
<td>Multiple concurrent reads of counters. The cluster must first be populated by a counter_write test.</td>
</tr>
<tr>
<td>user</td>
<td>Interleave user provided queries with configurable ratio and distribution.</td>
</tr>
<tr>
<td>print</td>
<td>Inspect the output of a distribution definition.</td>
</tr>
<tr>
<td>legacy</td>
<td>Legacy support mode.</td>
</tr>
</tbody>
</table>

Additional sub options are available for each option in the following table. Format:

```
$ cassandra-stress help option
```

For an example, see [View schema help](#).
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-pop</strong></td>
<td>Population distribution and intra-partition visit order.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -pop \ [seq=?] [no-wrap] [read-lookback=DIST(??)] [contents=?] or $ -pop \ [dist=DIST(??)] [contents=?]</td>
</tr>
<tr>
<td><strong>-insert</strong></td>
<td>Insert specific options relating to various methods for batching and splitting partition updates.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -insert \ [revisit=DIST(??)] [visits=DIST(??)] partitions=DIST(??) [batchtype=?] select-ratio=DIST(??)</td>
</tr>
<tr>
<td><strong>-col</strong></td>
<td>Column details, such as size and count distribution, data generator, names, and comparator.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -col \ [n=DIST(??)] [slice] [super=?] [timestamp=?] [size=DIST(??)]</td>
</tr>
<tr>
<td><strong>-rate</strong></td>
<td>Thread count, rate limit, or automatic mode (default is auto).</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -rate \ [threads=?] [limit=?] or $ -rate \ [threads=?] [threads=&lt;auto]</td>
</tr>
<tr>
<td><strong>-mode</strong></td>
<td>Thrift or CQL with options.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -mode \ [threads=?] [limit=?] or $ -mode \ [threads&gt;=?] [threads&lt;=?] [auto]</td>
</tr>
<tr>
<td><strong>-errors</strong></td>
<td>How to handle errors when encountered during stress.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -errors \ [retries=?] [ignore] [skip-read-validation]</td>
</tr>
<tr>
<td><strong>-sample</strong></td>
<td>Specify the number of samples to collect for measuring latency.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -sample \ [history=?] [live=?] [report=?]</td>
</tr>
<tr>
<td><strong>-schema</strong></td>
<td>Replication settings, compression, compaction, and so on.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -schema \ [replication(??)] [keyspace=?] [compaction(??)] [compression=?]</td>
</tr>
<tr>
<td><strong>-node</strong></td>
<td>Nodes to connect to.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -node \ [whitelist] [file=?] []</td>
</tr>
<tr>
<td><strong>-log</strong></td>
<td>Where to log progress and the interval to use.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -log \ [level=?] [no-summary] [file=?] [interval=?]</td>
</tr>
<tr>
<td><strong>-transport</strong></td>
<td>Custom transport factories.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -transport \ [factory=?] [truststore=?] [truststore-password=?] [ssl-protocol=?] [ssl-alg=?] [store-type=?]</td>
</tr>
<tr>
<td><strong>-port</strong></td>
<td>Specify port for connecting Cassandra nodes.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -port \ [native=?] [thrift=?] [jmx=?]</td>
</tr>
<tr>
<td><strong>-sendto</strong></td>
<td>Specify stress server to send this command to.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>$ -sendToDaemon &lt;host&gt;</td>
</tr>
</tbody>
</table>

In Cassandra2.1.5 and later, there are additional options:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>profiles</strong></td>
<td>Designate the YAML file to use with cassandra-stress.</td>
</tr>
<tr>
<td><strong>ops(?)</strong></td>
<td>Specify what operations (inserts and/or queries) to run and the number of each.</td>
</tr>
<tr>
<td><strong>clusterings=DIST(??)</strong></td>
<td>Distribution clustering runs of operations of the same kind.</td>
</tr>
<tr>
<td><strong>err=??</strong></td>
<td>Specify a standard error of the mean; when this value is reached, cassandra-stress will end. Default is 0.02.</td>
</tr>
<tr>
<td><strong>n=??</strong></td>
<td>Specify a minimum number of iterations to run before accepting uncertainty convergence.</td>
</tr>
<tr>
<td><strong>nc=??</strong></td>
<td>Specify a maximum number of iterations to run before accepting uncertainty convergence.</td>
</tr>
<tr>
<td><strong>nw=??</strong></td>
<td>Specify the number of operations to run.</td>
</tr>
<tr>
<td><strong>duration=??</strong></td>
<td>Specify the time to run, in seconds, minutes or hours.</td>
</tr>
<tr>
<td><strong>no-warmup</strong></td>
<td>Do not warmup the process, do a cold start.</td>
</tr>
<tr>
<td><strong>truncate=??</strong></td>
<td>Truncate the table created during cassandra-stress. Options are never, once, or always. Default is never.</td>
</tr>
<tr>
<td><strong>cls=??</strong></td>
<td>Set the consistency level to use during cassandra-stress. Options are ONE, QUORUM, LOCAL_QUORUM, EACH, QUORUM, ALL, and ANY. Default is LOCAL_ONE.</td>
</tr>
</tbody>
</table>
Example: Simple read and write examples

$ tools/bin/cassandra-stress write n=1000000

Insert (write) one million rows.

$ tools/bin/cassandra-stress read n=200000

Read two hundred thousand rows.

$ tools/bin/cassandra-stress read duration=3m

Read rows for a duration of 3 minutes.

$ tools/bin/cassandra-stress read n=200000 no-warmup

Read 200,000 rows without a warmup of 50,000 rows first.

Example: View schema help

$ tools/bin/cassandra-stress help -schema

<table>
<thead>
<tr>
<th>replication([strategy=?][factor=?][&lt;option 1..N&gt;=?]):</th>
<th>Define the replication strategy and any parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy=? (default=org.apache.cassandra.locator.SimpleStrategy)</td>
<td>The replication strategy to use</td>
</tr>
<tr>
<td>factor=? (default=1)</td>
<td>The number of replicas</td>
</tr>
<tr>
<td>keyspace=? (default=keyspace1)</td>
<td>The keyspace name to use</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>compaction([strategy=?][&lt;option 1..N&gt;=?]):</th>
<th>Define the compaction strategy and any parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy=?</td>
<td>The compaction strategy to use</td>
</tr>
<tr>
<td>compression=?</td>
<td>Specify the compression to use for SSTable, default:no compression</td>
</tr>
</tbody>
</table>

Example: 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
$ tools/bin/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
Example: 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)"
```

Example: 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.

```
$ tools/bin/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 that 16 or more than 256 client threads.

Example: 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
```
Example: 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

$ tools/bin/cassandra-stress write n=1000000 cl=one -mode native cql3 -schema
dkeyspace="keyspace1" -pop seq=1..1000000 -log file=/node1_load.log -node $NODES

# On Node2

$ tools/bin/cassandra-stress write n=1000000 cl=one -mode native cql3 -schema
dkeyspace="keyspace1" -pop seq=1000001..2000000 -log file=/node2_load.log -node $NODES

Example: 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 operation defined as simple1 will be completed once. No warmup is specified, and the consistency level is set to QUORUM.

$ tools/bin/cassandra-stress user profile=tools/cqlstress-example.yaml ops
\(\text{simple1}=1\) no-warmup cl=QUORUM

For a complete description on using these sample files, see Improved Cassandra 2.1 Stress Tool: Benchmark Any Schema – Part 1.

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 through it using the --send-to option.

- Insert 1,000,000 rows to given host:

/tools/bin/cassandra-stress -node 192.168.1.101

When the number of rows is not specified, one million rows are inserted.
Cassandra tools

- Read 1,000,000 rows from given host:
  
  ```
  tools/bin/cassandra-stress read -node 192.168.1.101
  ```

- 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 --send-to 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.

Sleeping 2s...

Warming up WRITE with 50000 iterations...

Running WRITE with 200 threads for 1000000 iterations

<table>
<thead>
<tr>
<th>type</th>
<th>total ops</th>
<th>op/s</th>
<th>pk/s</th>
<th>row/s</th>
<th>mean</th>
<th>med</th>
<th>.95</th>
<th>.99</th>
<th>.999</th>
<th>max</th>
<th>time</th>
<th>stderr</th>
<th>errors</th>
<th>gc: #</th>
<th>max ms</th>
<th>sum ms</th>
<th>sdv ms</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>239.3</td>
<td>255.4</td>
<td>1.0</td>
<td>0.00000</td>
<td>0</td>
<td>1</td>
<td>49</td>
<td>49</td>
<td>0</td>
<td>612</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>98715</td>
<td>43857</td>
<td>43857</td>
<td>43857</td>
<td>4.6</td>
<td>1.7</td>
<td>8.5</td>
<td>8.5</td>
<td>98.6</td>
<td>204.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>157777</td>
<td>47283</td>
<td>47283</td>
<td>47283</td>
<td>4.1</td>
<td>1.4</td>
<td>8.3</td>
<td>8.3</td>
<td>70.6</td>
<td>251.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o/s        : 46751 [WRITE:46751]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partition rate : 46751 [WRITE:46751]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row rate : 46751 [WRITE:46751]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency mean : 4.3 [WRITE:4.3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency median : 1.3 [WRITE:1.3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency 95th percentile : 7.2 [WRITE:7.2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency 99th percentile : 60.5 [WRITE:60.5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency 99.9th percentile : 223.2 [WRITE:223.2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>latency max : 503.1 [WRITE:503.1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total partitions : 1000000 [WRITE:1000000]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total errors : 0 [WRITE:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total gc count : 18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total gc mb : 10742</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total gc time (s) : 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg gc time(ms) : 73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdev gc time(ms) : 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total operation time : 00:00:21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
### Data Description

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row/s</td>
<td>Number of row operations per second performed during the run.</td>
</tr>
<tr>
<td>mean</td>
<td>Average latency in milliseconds for each operation during that run.</td>
</tr>
<tr>
<td>med</td>
<td>Median latency in milliseconds for each operation during that run.</td>
</tr>
<tr>
<td>.95</td>
<td>95% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>.99</td>
<td>99% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>.999</td>
<td>99.9% of the time the latency was less than the number displayed in the column.</td>
</tr>
<tr>
<td>max</td>
<td>Maximum latency in milliseconds.</td>
</tr>
<tr>
<td>time</td>
<td>Total operation time.</td>
</tr>
<tr>
<td>stderr</td>
<td>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.</td>
</tr>
<tr>
<td>gc: #</td>
<td>Number of garbage collections.</td>
</tr>
<tr>
<td>max ms</td>
<td>Longest garbage collection in milliseconds.</td>
</tr>
<tr>
<td>sum ms</td>
<td>Total of garbage collection in milliseconds.</td>
</tr>
<tr>
<td>adv ms</td>
<td>Standard deviation in milliseconds.</td>
</tr>
<tr>
<td>mb</td>
<td>Size of the garbage collection in megabytes.</td>
</tr>
</tbody>
</table>

### 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:**

- **Cassandra Package installations:**
  
  ```bash
  $ sstableexpiredblockers [--dry-run] keyspace table
  ```

- **Cassandra Tarball installations:**
  
  ```bash
  $ cd install_location/tools $ bin/sstableexpiredblockers [--dry-run] keyspace table
  ```

1. Choose a keyspace and table to check for any SSTables that are blocking the specified table from dropping.

   ```bash
   $ sstableexpiredblockers cycling cyclist_name
   ```

#### sstablekeys

The sstablekeys utility dumps table keys. 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. The location of the data directory, listed in the "Install locations" section, depends on the type of installation. After finding the name of the file, use the name as an argument to the sstablekeys command.
Cassandra tools

- Cassandra Package installations:
  
  \$ sstablekeys <sstable_name>

- Cassandra Tarball installations:
  
  \$ cd install_location/resources/cassandra \$ bin/sstablekeys <sstable_name>

1. Create the playlists table in the music keyspace as shown in Data modeling.

2. Insert the row of data about ZZ Top in playlists:

```
INSERT INTO music.playlists (id, song_order, song_id, title, artist, album)
VALUES (62c36092-82a1-3a00-93d1-46196ee77204,
        1,
        a3e64f8f-bd44-4f28-b8d9-6938726e34d4,
        'La Grange',
        'ZZ Top',
        'Tres Hombres');
```

3. Flush the data to disk.

\$ nodetool flush music playlists

4. 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:

\$ sstablekeys <path to data>/data/data/music/
playlists-8b9f4cc0229211e4b02073ded3cb6170/music-playlists-ka-1-Data.db

The output appears, for example:

```
62c360928a13a0093d146196ee77204
```

**sstablelevelreset**

Reset level to 0 on a given set of SSTables that use LeveledCompactionStrategy.

Usage:

- Cassandra Package installations:  $ sstablelevelreset [--really-reset] keyspace table
- Cassandra Tarball installations:
  
  \$ cd install_location/tools \$ bin/sstablelevelreset [--really-reset] keyspace table

The option **--really-reset** is a warning that Cassandra is stopped before the tool is run.

- Stop Cassandra on the node. Choose a keyspace and table to reset to level 0.

  \$ sstablelevelreset --really-reset cycling cyclist_name

If the designated table is already at level 0, then no change occurs. If the SSTable is releveled, the metadata is rewritten to designate the level to 0.

**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.

Running the sstableloader against the live data directory can cause snapshots to fail. Specify the snapshots directory when running the sstableloader.

In the /var/lib/cassandra/data directory, select a keyspace and a table to access the associated snapshots directory, as shown in the following example.

$ cd /var/lib/cassandra/data/keyspace/table_name/snapshots/snapshot_name

Run sstableloader specifying the path to the SSTables and passing it the location of the target cluster. When using the sstableloader be aware of the following:

• Bulkloading SSTables created in versions prior to Cassandra 3.0 is supported only in Cassandra 3.0.5 and later.

• Repairing tables that have been loaded into a different cluster does not repair the source tables.

Prerequisites:

• The source data loaded by sstableloader must be in SSTables.

• Because sstableloader uses the streaming protocol, it requires a direct connection over the port 7000 (storage port) to each connected node.

Generating SSTables

When using sstableloader to load external data, you must first put the external data into 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. For a complete example, see https://www.datastax.com/blog/2014/09/using-cassandra-bulk-loader-updated.

Restoring Cassandra snapshots

For information about preparing snapshots for sstableloader import, see Restoring from a snapshot.

Importing SSTables from an existing cluster

Before importing existing SSTables, run nodetool flush on each source node to assure that any data in memtables is written out to the SSTables on disk.

Preparing the target environment

Before loading the data, you must define the schema of the target tables with CQL or Thrift.

Usage

• Cassandra Package installations:

$ sstableloader -d host_url (,host_url ...) [options] path_to_keyspace

• Cassandra Tarball installations:

$ cd install_location
Cassandra tools

```
$ bin/sstableloader -d host_url (...host_url ...) [options] path_to_keyspace
```

The `sstableloader` bulk loads the SSTables found in the specified directory, where the parent directories of the path are used for the target keyspace and table name, to the indicated target cluster.

Verify the location of the sstables to be streamed ends with directories named for the keyspace and table:

```
$ ls /var/lib/cassandra/data/Keyspace1/Standard1/snapshot/snapshot_name
```

<table>
<thead>
<tr>
<th>Keyspace1-Standard1-jb-60-CRC.db</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyspace1-Standard1-jb-60-Data.db</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Keyspace1-Standard1-jb-60-TOC.txt</td>
</tr>
</tbody>
</table>

For more `sstableloader` options, see `sstableloader options`

**Using sstableloader**

1. If restoring snapshot data from some other source: make sure that the snapshot files are in a `keyspace/tablename/snapshot/snapshot_name` directory path whose names match those of the target `keyspace/tablename/snapshot/snapshot_name`. In this example, make sure the snapshot files are in `/Keyspace1/Standard1/snapshot/snapshot_name`.

2. Go to the location of the SSTables:
   - **Cassandra Package installations:**
     ```
     $ cd /var/lib/cassandra/data/Keyspace1/Standard1/snapshot/snapshot_name
     ```
   - **Cassandra Tarball installations:**
     ```
     $ cd install_location/data/data/Keyspace1/Standard1/snapshot/snapshot_name
     ```

3. To view the contents of the keyspace:
   ```
   $ ls
   ```

<table>
<thead>
<tr>
<th>Keyspace1-Standard1-jb-60-CRC.db</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyspace1-Standard1-jb-60-Data.db</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Keyspace1-Standard1-jb-60-TOC.txt</td>
</tr>
</tbody>
</table>

4. To bulk load the files, **indicate one or more nodes in the target cluster with the -d flag, which takes a comma-separated list of IP addresses or hostnames, and specify the path to ..//Keyspace1/Standard1/snapshot/snapshot_name in the source machine:**

   ```
   $ sstableloader -d 110.82.155.1 /var/lib/cassandra/data/Keyspace1/Standard1/snapshot/snapshot_name ## package installation
   ```

   ```
   $ install_location/bin/sstableloader -d 110.82.155.1 /var/lib/cassandra/data/Keyspace1/Standard1/snapshot/snapshot_name ## tarball installation
   ```

This command bulk loads all files.
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.

Table 90: sstableloader options

<table>
<thead>
<tr>
<th>Short option</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-alg</td>
<td>--ssl-alg &lt;ALGORITHM&gt;</td>
<td>Client SSL algorithm (default: SunX509).</td>
</tr>
<tr>
<td>-ap</td>
<td>--auth-provider &lt;auth provider class name&gt;</td>
<td>Allows the use of a third party auth provider. Can be combined with -u &lt;username&gt; and -pw &lt;password&gt; if the auth provider supports plain text credentials.</td>
</tr>
<tr>
<td>-ciphers</td>
<td>--ssl-ciphers &lt;CIPHER-SUITES&gt;</td>
<td>Client SSL. Comma-separated list of encryption suites.</td>
</tr>
<tr>
<td>-cph</td>
<td>--connections-per-host &lt;connectionsPerHost&gt;</td>
<td>Number of concurrent connections-per-host.</td>
</tr>
<tr>
<td>-d</td>
<td>--nodes &lt;initial_hosts&gt;</td>
<td>Required. Connect to a list of (comma separated) hosts for initial cluster information.</td>
</tr>
<tr>
<td>-f</td>
<td>--conf-path &lt;path_to_config_file&gt;</td>
<td>Path to the cassandra.yaml path for streaming throughput and client/server SSL.</td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>Display help.</td>
</tr>
<tr>
<td>-i</td>
<td>--ignore &lt;NODES&gt;</td>
<td>Do not stream to this comma separated list of nodes.</td>
</tr>
<tr>
<td>-ks</td>
<td>--keystore &lt;KEYSTORE&gt;</td>
<td>Client SSL. Full path to the keystore.</td>
</tr>
<tr>
<td>-kspw</td>
<td>--keystore-password &lt;KEYSTORE-PASSWORD&gt;</td>
<td>Client SSL. Password for the keystore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overrides the client_encryption_options option in cassandra.yaml</td>
</tr>
<tr>
<td>-no-progress</td>
<td></td>
<td>Do not display progress.</td>
</tr>
<tr>
<td>-p</td>
<td>--port &lt;rpc port&gt;</td>
<td>RPC port (default: 9160 [Thrift]).</td>
</tr>
<tr>
<td>-prtcl</td>
<td>--ssl-protocol &lt;PROTOCOL&gt;</td>
<td>Client SSL. Connections protocol to use (default: TLS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overrides the server_encryption_options option in cassandra.yaml</td>
</tr>
<tr>
<td>-pw</td>
<td>--password &lt;password&gt;</td>
<td>Password for Cassandra authentication.</td>
</tr>
<tr>
<td>-st</td>
<td>--store-type &lt;STORE-TYPE&gt;</td>
<td>Client SSL. Type of store.</td>
</tr>
<tr>
<td>-t</td>
<td>--throttle &lt;throttle&gt;</td>
<td>Throttle speed in Mbits (default: unlimited).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overrides the stream_throughput_outbound_megabits_per_sec option in cassandra.yaml</td>
</tr>
<tr>
<td>-tf</td>
<td>--transport-factory &lt;transport factory&gt;</td>
<td>Fully-qualified ITransportFactory class name for creating a connection to Cassandra.</td>
</tr>
<tr>
<td>-ts</td>
<td>--truststore &lt;TRUSTSTORE&gt;</td>
<td>Client SSL. Full path to truststore.</td>
</tr>
<tr>
<td>-tspw</td>
<td>--truststore-password &lt;TRUSTSTORE-PASSWORD&gt;</td>
<td>Client SSL. Password of the truststore.</td>
</tr>
<tr>
<td>-u</td>
<td>--username &lt;username&gt;</td>
<td>User name for Cassandra authentication.</td>
</tr>
<tr>
<td>-v</td>
<td>--verbose</td>
<td>Verbose output.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Package installations</th>
<th>/etc/cassandra/cassandra.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarball installations</td>
<td>install_location/resources/cassandra/conf/cassandra.yaml</td>
</tr>
</tbody>
</table>
Cassandra tools

**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.

1. Run the *sstablemetadata* command. Multiple SSTable filenames can be included in the command.

   **Cassandra Package installations:**
   
   ```
   $ sstablemetadata <sstable_name filenames>
   ```

   **Cassandra Tarball installations:**
   
   ```
   $ cd install_location/tools $ 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
   9              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
<table>
<thead>
<tr>
<th>Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7007506</td>
<td>0</td>
</tr>
<tr>
<td>8409007</td>
<td>0</td>
</tr>
<tr>
<td>10090808</td>
<td>0</td>
</tr>
<tr>
<td>12108970</td>
<td>0</td>
</tr>
<tr>
<td>14530764</td>
<td>0</td>
</tr>
<tr>
<td>17436917</td>
<td>0</td>
</tr>
<tr>
<td>20924300</td>
<td>0</td>
</tr>
<tr>
<td>25109160</td>
<td>0</td>
</tr>
<tr>
<td>30130992</td>
<td>0</td>
</tr>
<tr>
<td>36157190</td>
<td>0</td>
</tr>
<tr>
<td>43388628</td>
<td>0</td>
</tr>
<tr>
<td>52066354</td>
<td>0</td>
</tr>
<tr>
<td>62479625</td>
<td>0</td>
</tr>
<tr>
<td>74975550</td>
<td>0</td>
</tr>
<tr>
<td>89970660</td>
<td>0</td>
</tr>
<tr>
<td>107964792</td>
<td>0</td>
</tr>
<tr>
<td>129557750</td>
<td>0</td>
</tr>
<tr>
<td>155469300</td>
<td>0</td>
</tr>
<tr>
<td>186563160</td>
<td>0</td>
</tr>
<tr>
<td>223875792</td>
<td>0</td>
</tr>
<tr>
<td>268650950</td>
<td>0</td>
</tr>
<tr>
<td>322381140</td>
<td>0</td>
</tr>
<tr>
<td>386857368</td>
<td>0</td>
</tr>
<tr>
<td>464228842</td>
<td>0</td>
</tr>
<tr>
<td>557074610</td>
<td>0</td>
</tr>
<tr>
<td>668489532</td>
<td>0</td>
</tr>
<tr>
<td>802074383</td>
<td>0</td>
</tr>
<tr>
<td>962624926</td>
<td>0</td>
</tr>
<tr>
<td>1155149911</td>
<td>0</td>
</tr>
<tr>
<td>1386179893</td>
<td>0</td>
</tr>
<tr>
<td>1663415872</td>
<td>0</td>
</tr>
<tr>
<td>1996099046</td>
<td>0</td>
</tr>
<tr>
<td>2395318855</td>
<td>0</td>
</tr>
<tr>
<td>2874382626</td>
<td>0</td>
</tr>
<tr>
<td>3449259151</td>
<td>0</td>
</tr>
<tr>
<td>4139110981</td>
<td>0</td>
</tr>
<tr>
<td>4966931777</td>
<td>0</td>
</tr>
<tr>
<td>5960319812</td>
<td>0</td>
</tr>
<tr>
<td>7152387774</td>
<td>0</td>
</tr>
<tr>
<td>8582860529</td>
<td>0</td>
</tr>
<tr>
<td>10299432635</td>
<td>0</td>
</tr>
<tr>
<td>12359319162</td>
<td>0</td>
</tr>
<tr>
<td>14831182994</td>
<td>0</td>
</tr>
<tr>
<td>17797419593</td>
<td>0</td>
</tr>
<tr>
<td>21356903512</td>
<td>0</td>
</tr>
<tr>
<td>25628284214</td>
<td>0</td>
</tr>
<tr>
<td>3075391057</td>
<td>0</td>
</tr>
<tr>
<td>36904729268</td>
<td>0</td>
</tr>
<tr>
<td>44285575122</td>
<td>0</td>
</tr>
<tr>
<td>53142810146</td>
<td>0</td>
</tr>
<tr>
<td>63771372175</td>
<td>0</td>
</tr>
<tr>
<td>76525646610</td>
<td>0</td>
</tr>
<tr>
<td>91830775932</td>
<td>0</td>
</tr>
<tr>
<td>110196931118</td>
<td>0</td>
</tr>
<tr>
<td>132236317342</td>
<td>0</td>
</tr>
<tr>
<td>158683580810</td>
<td>0</td>
</tr>
<tr>
<td>190420296972</td>
<td>0</td>
</tr>
<tr>
<td>228504356366</td>
<td>0</td>
</tr>
<tr>
<td>274205227639</td>
<td>0</td>
</tr>
<tr>
<td>329046273167</td>
<td>0</td>
</tr>
<tr>
<td>394855278000</td>
<td>0</td>
</tr>
<tr>
<td>473826633360</td>
<td>0</td>
</tr>
<tr>
<td>568591960032</td>
<td>0</td>
</tr>
<tr>
<td>68231032038</td>
<td>0</td>
</tr>
<tr>
<td>818772422446</td>
<td>0</td>
</tr>
<tr>
<td>982526906935</td>
<td>0</td>
</tr>
</tbody>
</table>
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 (e.g., 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:
- Cassandra Package installations:
  
  ```
  $ sstableofflinerelevel [--dry-run] keyspace table
  ```
- Cassandra Tarball installations:
  
  ```
  $ cd install_location/tools $ bin/sstableofflinerelevel [--dry-run] keyspace table
  ```

1. 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.

Do not run this command until you have stopped Cassandra on the node.

Use this tool in the process of migrating a Cassandra installation to incremental repair. For details, see Migrating to incremental repairs

Usage:
- Cassandra Package installations:
  
  ```
  $ sstablerepairedset [--is-repaired | --is-unrepaired] [-f sstable-list | sstables]
  ```
- Cassandra Tarball installations:
  
  ```
  $ cd install_location/tools $ bin/sstablerepairedset [--is-repaired | --is-unrepaired]
  [-f sstable-list | sstables]
  ```

- Choose SSTables to mark as repaired.

  ```
  $ sstablerepairedset --is-repaired /var/lib/cassandra/data/cycling/cyclist_name-
a882dca02aaf11e58c78b496c707234/la-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-2.1.10/data/data/test/cyclist_by_country-82246fc065ff11e5a4c58b496c707234/ma-1-big-Data.db
```
Cassandra tools

A command to use to list all the Data.db files in a keyspace is the following:

```
$ find '/home/user/apache-cassandra-2.1.10/data/data/test/' -iname ".*Data.db*"
```

**sstablescrub**

The sstablescrub utility is an offline version of `nodetool scrub`. It attempts to remove the corrupted parts while preserving non-corrupted data. Because sstablescrub 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.

1. **Before using sstablescrub**, 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:**

   - **Cassandra Package installations:**
     
     ```
     $ sstablescrub [options] <keyspace> <table>
     ```

   - **Cassandra Tarball installations:**
     
     ```
     $ cd install_location $ bin/sstablescrub [options] <keyspace> <table>
     ```

**Table 91: Options**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td></td>
<td>Display stack traces.</td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>Display help.</td>
</tr>
<tr>
<td>-m</td>
<td>--manifest-check</td>
<td>Only check and repair the leveled manifest, without actually scrubbing the SSTables.</td>
</tr>
<tr>
<td></td>
<td>--reinsert-overflowed-ttl</td>
<td>Rewrites SSTables containing rows with overflowed expiration time with the maximum expiration date of 2038-01-19T03:14:06+00:00 using the original timestamp + 1 (ms).</td>
</tr>
<tr>
<td>-s</td>
<td>--skip-corrupted</td>
<td>Skip corrupt rows in counter tables.</td>
</tr>
<tr>
<td>-v</td>
<td>--verbose</td>
<td>Verbose output.</td>
</tr>
</tbody>
</table>

**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:
• Cassandra Package installations:

$ sudo service cassandra stop

• Cassandra Tarball installations:

$ ps auwx | grep cassandra

Usage:

• Cassandra Package installations:

$ sstablessplit [options] <filename> [<filename>]*

• Cassandra Tarball installations:

$ cd install_location/tools

$ bin/sstablessplit [options] <filename> [<filename>]*

Example:

$ sstablessplit -s 40 /var/lib/cassandra/data/Keyspace1/Standard1/*

<table>
<thead>
<tr>
<th>Flag</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td></td>
<td>Display stack traces.</td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>Display help.</td>
</tr>
<tr>
<td>--no-snapshot</td>
<td></td>
<td>Do not snapshot the SSTables before splitting.</td>
</tr>
<tr>
<td>-s</td>
<td>--size &lt;size&gt;</td>
<td>Maximum size in MB for the output SSTables (default: 50).</td>
</tr>
<tr>
<td>-v</td>
<td>--verbose</td>
<td>Verbose output.</td>
</tr>
</tbody>
</table>

The sstableupgrade tool

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:

• Cassandra Package installations:

$ sstableupgrade [options] keyspace table [snapshot]

• Cassandra Tarball installations:

$ cd install_location $ bin/sstableupgrade [options] keyspace table [snapshot]

The snapshot option only upgrades the specified snapshot.
Cassandra tools

Table 93: Options

<table>
<thead>
<tr>
<th>Flag</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Display stack traces.</td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>Display help.</td>
</tr>
</tbody>
</table>

sstable2json

Converts the on-disk SSTable representation of a table into a JSON formatted document. Converting SSTables this way is useful for testing and debugging.

This tool is not recommended as a production tool and is not suitable for production operations!

Starting with version 0.7, json2sstable and sstable2json must be run so that the schema can be loaded from system tables. This means that the cassandra.yaml file must be in the classpath and refer to valid storage directories.

1. Usage:

   • Cassandra Package installations:

     ```
     $ sstable2json SSTABLE [ -k KEY [-k KEY [...]]] [ -x KEY [ -x KEY [...]]] [-e ]
     ```

   • Cassandra Tarball installations:

     ```
     $ cd install_location/tools $ bin/sstable2json SSTABLE [ -k KEY [-k KEY [...]]] [ -x KEY [ -x KEY [...]]] [-e ]
     ```

SSTABLE should be a full path to a `{table-name}-Data.db` file in Cassandra’s data directory. For example, `/var/lib/cassandra/data/Keyspace1/Standard1-e-1-Data.db`.

   • `-k` allows you to include a specific set of keys. The KEY must be in HEX format. Limited to 500 keys.
   
   • `-x` allows you to exclude a specific set of keys. Limited to 500 keys.
   
   • `-e` causes keys to only be enumerated.

The output is:

```json
{
    ROW_KEY:
    {
        [COLUMN_NAME, COLUMN_VALUE, COLUMN_TIMESTAMP, IS_MARKED_FOR_DELETE],
        [COLUMN_NAME, ... ],
        ...
    },
    ROW_KEY:
    {
        ...
    },
    ...
}
```

Row keys, column names and values are written in as the HEX representation of their byte arrays. Line breaks are only in between row keys in the actual output.

Related topics

The cassandra.yaml configuration file
Tracking counter columns
You can use the sstable2json command to get information about a counter column.

1. Run the counter example presented earlier that loads data into a counter column and flushes data to disk.
The counter is initialized to 1.

2. Run the sstable2json command.

   ```
   $ sudo ./sstable2json 
   /var/lib/cassandra/data/counterks/page_view_counts/counterks-
   page_view_counts-ib-1-Data.db
   ```

   ```
   ["key": "777772e64617461737461782e636f6d","columns": [[
   "home:","","13701871642560000",["home:counter_value","0001000058852cd0cb9311e2940971f75c7d064100000000000000010000000000000001",1370187164256,"c",-9223372036854775808]
   ]
   ]
   ```

3. Increase the counter column by 2 and flush the data to disk again.

4. Run the sstable2json command again.

   ```
   $ sudo ./sstable2json /var/lib/cassandra/data/counterks/page_view_counts/counterks-
   page_view_counts-ib-2-Data.db
   ```

   ```
   ["key": "777772e64617461737461782e636f6d","columns": [[
   "home:","","13701873156830000",["home:counter_value","0001000058852cd0cb9311e2940971f75c7d064100000000000000010000000000000002",1370187315683,"c",-9223372036854775808]
   ]
   ```

   -9223372036854775808 is the timestamp of the last delete.

Tracking data expiration
The output of the sstable2json command reveals the life cycle of Cassandra data. In this procedure, you use
the sstable2json to view data in a row that is not scheduled to expire, data that has been evicted and marked
with a tombstone, and a row that has had data removed from it.

1. Create the playlists table in the music keyspace as shown in Data modeling.

2. Insert the row of data about ZZ Top in playlists:

   ```
   INSERT INTO music.playlists (id, song_order, song_id, title, artist, album)
   VALUES (62c36092-82a1-3a00-93d1-46196ee77204,
   1,
   a3e64f8f-bd44-4f28-b8d9-6938726e34d4,
   'La Grange',
   'ZZ Top',
   'Tres Hombres');
   ```

3. Flush the data to disk.

   ```
   $ sudo ./nodetool flush music playlists
   ```

   You need to have access permission to the data directories to flush data to disk.
4. Look at the JSON representation of the SSTable data, for example:

```
$ sudo ./sstable2json /var/lib/cassandra/data/music/playlists/music-playlists-ib-1-Data.db
```

Output is:

```
[
{"key": "62c3609282a13a093d146196ee77204","columns": [
 "1:","",1370179611971000],
 "1:album","Tres Hombres",1370179611971000],
 "1:artist","ZZ Top",1370179611971000],
 "1:song_id","a3e64f8f-bd44-4f28-b8d9-6938726e34d4",1370179611971000],
 "1:title","La Grange",1370179611971000]}
```

5. Specify the time-to-live (TTL) for the ZZ Top row, for example 300 seconds.

```
INSERT INTO music.playlists
(id, song_order, song_id, title, artist, album)
VALUES (62c36092-82a1-3a00-93d1-46196ee77204,
1,
a3e64f8f-bd44-4f28-b8d9-6938726e34d4,
'La Grange',
'ZZ Top',
'Tres Hombres')
USING TTL 300;
```

After inserting the TTL property on the row to expire the data, Cassandra marks the row with tombstones. You need to list all columns when re-inserting data if you want Cassandra to remove the entire row.

6. Flush the data to disk again.

Do this while the data is evicted, but before the time-to-live elapses and data is removed.

7. Run the sstable2json command again.

```
sudo ./sstable2json
/var/lib/cassandra/data/music/playlists/music-playlists-ib-2-Data.db
```

The tombstone markers--"e" followed by the TTL value, 300--are visible in the JSON representation of the data.

```
[
{"key": "62c3609282a13a093d146196ee77204","columns": [
 "1:","",1370179816450000,"e",300,1370180116],
 "1:album","Tres Hombres",1370179816450000,"e",300,1370180116],
 "1:artist","ZZ Top",1370179816450000,"e",300,1370180116],
 "1:song_id","a3e64f8f-bd44-4f28-b8d9-6938726e34d4",1370179816450000,"e",300,1370180116],
 "1:title","La Grange",1370179816450000,"e",300,1370180116]}
```

8. After the TTL elapses, flush the data to disk again.

9. Run the sstable2json command again.

```
$ sudo ./sstable2json
```
The JSON representation of the column data shows that the tombstones and data values for the ZZ Top row have been deleted from the SSTable. The values are now marked with "d":

```

```

### json2sstable

Converts a JSON representation of a table (aka column family) to a Cassandra usable SSTable format.

This tool is not recommended as a production tool and is not suitable for production operations!

1. Usage:

   $ cd install_location/tools
   $ bin/json2sstable [-s] -K KEYSPACE <keyspace> -c <table> [-n num_keys] <json> <sstable>

   `<json>` should be a path to the JSON file. `<sstable>` should be a full path to a `{table-name}-Data.db` file in Cassandra’s data directory. For example, `/var/lib/cassandra/data/Keyspace1/Standard1-e-1-Data.db`.

   - `-s` assumes the JSON file is already sorted and was created by the sstable2json tool.
   - `-n` specifies the number of keys to import.
Starting and stopping Cassandra

Starting Cassandra as a service
Start the Apache Cassandra™ Java server process for packaged installations.
Start-up scripts are provided in the /etc/init.d directory. The service runs as the cassandra user.

You must have root or sudo permissions to start Cassandra as a service.

1. On initial start-up, each node must be started one at a time, starting with your seed nodes:

   ```bash
   $ sudo service cassandra start #Starts Cassandra
   ```

   If Cassandra fails to start:

   ```bash
   Reloading systemd: [ OK ]
   Starting cassandra (via systemctl): Job for cassandra.service failed because a configured resource limit was exceeded.
   See "systemctl status cassandra.service" and "journalctl -xe" for details.
   [FAILED]
   ```

   The Cassandra service is not enabled on newer Linux systems, which use systemd. To verify use:

   ```bash
   $ sudo systemctl is-enabled cassandra.service
   ```

   ```bash
   cassandra.service is not a native service, redirecting to /sbin/chkconfig.
   Executing /sbin/chkconfig cassandra --level=5 disabled
   ```

   To start Cassandra:

   a. Enable the service:

   ```bash
   $ sudo systemctl enable cassandra.service
   ```

   ```bash
   cassandra.service is not a native service, redirecting to /sbin/chkconfig.
   ```
b. Start Cassandra:

$ sudo service cassandra start

2. To check the status of Cassandra:

$ nodetool status

The status column in the output should report UN which stands for Up/Normal.

Starting Cassandra as a stand-alone process

Start the Cassandra Java server process for tarball installations.

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 #Starts Cassandra

- To start Cassandra in the foreground:

$ cd install_location

$ bin/cassandra -f #Starts Cassandra

- To monitor the progress of the startup:

$ tail -f logs/system.log

Cassandra is ready when it shows an entry like this in the system.log:

INFO [main] 2020-01-09 01:57:03,846 Server.java:194 - Starting listening for CQL clients on localhost/127.0.0.1:9042...
INFO [main] 2020-01-09 01:57:03,887 ThriftServer.java:119 - Binding thrift service to localhost/127.0.0.1:9160
To check the status of Cassandra:

```
$ bin/nodetool status
```

The status column in the output should report UN which stands for Up/Normal.

<table>
<thead>
<tr>
<th>Datacenter: datacenter1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status=Up/Down</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-- Address Load Tokens Owns (effective) Host ID Rack</td>
</tr>
<tr>
<td>UN 127.0.0.1 163.39 KB 256 100.0% 054b5c11-32dd-43c3-8f30-abcd66ba977b rack1</td>
</tr>
</tbody>
</table>

**Stopping Cassandra as a service**

Stopping the Cassandra Java server process on packaged installations.

1. You must have root or sudo permissions to stop the Cassandra service:

```
$ sudo service cassandra stop
```

2. Find the Cassandra Java process ID (PID), and then kill the process using its PID number:

```
$ ps auwx | grep cassandra
$ sudo kill pid #Stop Cassandra
```

**Stopping Cassandra as a stand-alone process**

Stop the Cassandra Java server process on tarball installations.

1. Find the Cassandra Java process ID (PID), and then kill the process using its PID number:

```
$ ps auwx | grep cassandra
$ sudo kill pid #Stop Cassandra
```

**Clearing the data as a service**

Remove all data from a package installation.

To clear the data from the `default` directories:

1. After stopping the service, run the following command:

```
$ sudo rm -rf /var/lib/cassandra/*
```

**Clearing the data as a stand-alone process**

Remove all data from a tarball installation.

1. To clear all data from the `default` directories, including the commitlog and saved_caches:

```
1. Stop the process.
```
2. Run the following command from the install directory:

```
$ cd install_location

$ sudo rm -rf data/*
```

- To clear the only the data directory:

1. **Stop** the process.

2. Run the following command from the install directory:

```
$ cd install_location

$ sudo rm -rf data/data/*
```

## Install locations

### Tarball installation directories

The configuration files are located in the following directories:

<table>
<thead>
<tr>
<th>Configuration Files</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra.yaml</td>
<td>install_location/conf</td>
</tr>
<tr>
<td>cassandra-topology.properties</td>
<td>install_location/conf</td>
</tr>
<tr>
<td>cassandra-rackdc.properties</td>
<td>install_location/conf</td>
</tr>
<tr>
<td>cassandra-env.sh</td>
<td>install_location/conf</td>
</tr>
<tr>
<td>cassandra.in.sh</td>
<td>install_location/bin</td>
</tr>
</tbody>
</table>

The binary tarball releases install into the installation directory.

<table>
<thead>
<tr>
<th>Directories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Files for commitlog, data, and saved_caches (unless set in cassandra.yaml)</td>
</tr>
<tr>
<td>bin</td>
<td>Utilities and start scripts</td>
</tr>
<tr>
<td>conf</td>
<td>Configuration files and environment settings</td>
</tr>
<tr>
<td>interface</td>
<td>Thrift and Avro client APIs</td>
</tr>
<tr>
<td>javadoc</td>
<td>Cassandra Java API documentation</td>
</tr>
<tr>
<td>lib</td>
<td>JAR and license files</td>
</tr>
<tr>
<td>tools</td>
<td>Cassandra tools and sample cassandra.yaml files for stress testing.</td>
</tr>
</tbody>
</table>

### Package installation directories

The configuration files are located in the following directories:

<table>
<thead>
<tr>
<th>Configuration Files</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra.yaml</td>
<td>/etc/cassandra</td>
</tr>
<tr>
<td>cassandra-topology.properties</td>
<td>/etc/cassandra</td>
</tr>
<tr>
<td>cassandra-rackdc.properties</td>
<td>/etc/cassandra</td>
</tr>
<tr>
<td>cassandra-env.sh</td>
<td>/etc/cassandra</td>
</tr>
<tr>
<td>cassandra.in.sh</td>
<td>/usr/share/cassandra</td>
</tr>
</tbody>
</table>

The packaged releases install into these directories:
### Cassandra include file

To set environment variables (Linux only), Cassandra can use an include file, `cassandra.in.sh`. This file is located in:

- Cassandra Tarball installations: `install_location/bin/cassandra.in.sh`
- Cassandra Package installations: `/usr/share/cassandra/cassandra.in.sh`

### Cassandra-CLI utility (deprecated)

The CLI utility is deprecated and will be removed in Cassandra 3.0. For ease of use and performance, switch from Thrift and CLI to CQL and cqlsh.

### Keyspace attributes

Cassandra stores storage configuration attributes in the system keyspace. You can set storage engine configuration attributes on a per-keyspace or per-table basis on the command line using the Cassandra-CLI utility. A keyspace must have a user-defined name, a replica placement strategy, and options that specify the number of copies per datacenter or node.

- **name**
  - Required. The name for the keyspace.

- **placement_strategy**
  - Required. Determines how Cassandra distributes replicas for a keyspace among nodes in the ring.
  - Values are:
    - `SimpleStrategy` or `org.apache.cassandra.locator.SimpleStrategy`

  NetworkTopologyStrategy requires a snitch to be able to determine rack and datacenter locations of a node. For more information about replication placement strategy, see Data replication.

- **strategy_options**
  - Specifies configuration options for the chosen replication strategy class. The replication factor option is 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 there are 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.
When the replication factor exceeds the number of nodes, writes are rejected, but reads are served as long as the desired consistency level can be met. For more information about configuring the replication placement strategy for a cluster and datacenters, see Choosing keyspace replication options.

**durable_writes**
(Default: **true**) When set to **false**, data written to the keyspace bypasses the commit log. Be careful using this option because you risk losing data.

**Table attributes**
The following attributes can be declared per table.

**bloom_filter_fp_chance**
- See CQL properties in CQL for Cassandra 2.x.

**bucket_high**
- See CQL Compaction Subproperties in CQL for Cassandra 2.x.

**bucket_low**
- See CQL Compaction Subproperties in CQL for Cassandra 2.x.

**caching**
- See CQL properties in CQL for Cassandra 2.x.

**chunk_length_kb**
- See CQLCompression Subproperties in CQL for Cassandra 2.x.

**column_metadata**
( Default: N/A - container attribute) Column metadata defines these attributes of a column:

- **name**: Binds a validation_class and (optionally) an index to a column.
- **validation_class**: Type used to check the column value.
- **index_name**: Name of the index.
- **index_type**: Type of index. Currently the only supported value is KEYS.

Setting a value for the name option is required. The **validation_class** is set to the default_validation_class of the table if you do not set the **validation_class** option explicitly. The value of **index_type** must be set to create an index for a column. The value of index_name is not valid unless **index_type** is also set.

Setting and updating column metadata with the Cassandra-CLI utility requires a slightly different command syntax than other attributes; note the brackets and curly braces in this example:

```
[default@demo ] UPDATE COLUMN FAMILY users WITH  comparator=UTF8Type
AND  column_metadata =
    |{column_name: full_name, validation_class: UTF8Type,
     |index_type: KEYS });
```

**column_type**
( Default: **Standard**) The standard type of table contains regular columns.

**comment**
- See CQL properties in CQL for Cassandra 2.x.

**compaction_strategy**
- See compaction in CQL properties in CQL for Cassandra 2.x.

**compaction_strategy_options**
( Default: N/A - container attribute) Sets attributes related to the chosen compaction-strategy. Attributes are:

- **bucket_high**
- **bucket_low**
References

- max_compaction_threshold
- min_compaction_threshold
- min_sstable_size
- sstable_size_in_mb
- tombstone_compaction_interval
- tombstone_threshold

**comparator**
(Default: BytesType) Defines the data types used to validate and sort column names. There are several built-in column comparators available. The comparator cannot be changed after you create a table.

**compression_options**
(Default: N/A - container attribute) Sets the compression algorithm and sub-properties for the table. Choices are:
- sstable_compression
- chunk_length_kb
- crc_check_chance

**crc_check_chance**
See CQLCompression Subproperties in CQL for Cassandra 2.x.

**default_time_to_live**
See CQL properties in CQL for Cassandra 2.x.

**default_validation_class**
(Default: N/A) Defines the data type used to validate column values. There are several built-in column validators available.

**gc_grace**
See CQL properties in CQL for Cassandra 2.x.

**index_interval**
See CQL properties in CQL for Cassandra 2.x.

**key_validation_class**
(Default: N/A) Defines the data type used to validate row key values. There are several built-in key validators available, however CounterColumnType (distributed counters) cannot be used as a row key validator.

**max_compaction_threshold**
See max_threshold in CQL Compaction Subproperties in CQL for Cassandra 2.x.

**min_compaction_threshold**
See min_threshold in CQL Compaction Subproperties in CQL for Cassandra 2.x.

**max_index_interval**
See CQL properties in CQL for Cassandra 2.x.

**min_index_interval**
See CQL properties in CQL for Cassandra 2.x.

**memtable_flush_after_mins**
Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility. Use commitlog_total_space_in_mb.

**memtable_flush_period_in_ms**
See CQL properties in CQL for Cassandra 2.x.

**memtable_operations_in_millions**
Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility. Use commitlog_total_space_in_mb.
**memtable_throughput_in_mb**
Depreciated as of Cassandra 1.0, but can still be declared for backward compatibility. Use `commitlog_total_space_in_mb`.

**min_sstable_size**
See CQL Compaction Subproperties in CQL for Cassandra 2.x.

**name**
(Default: N/A) Required. The user-defined name of the table.

**read_repair_chance**
See CQL properties in CQL for Cassandra 2.x.

**speculative_retry**
See CQL properties in CQL for Cassandra 2.x.

**sstable_size_in_mb**
See CQL Compaction Subproperties in CQL for Cassandra 2.x.

**sstable_compression**
See compression in CQL properties in CQL for Cassandra 2.x.

**tombstone_compaction_interval**
See CQL Compaction Subproperties in CQL for Cassandra 2.x.

**tombstone_threshold**
See CQL Compaction Subproperties in CQL for Cassandra 2.x.
Chapter 15. 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 RDBMS’s 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 community Cassandra.
Chapter 16. Troubleshooting

Troubleshooting has moved to DataStax Support.
Chapter 17. Release notes

New features, improvements, and notable changes are described in What's new in Cassandra 2.1.

The latest version of Apache Cassandra™ 2.1 is 2.1.21. The CHANGES.txt describes the changes in detail. You can view all version changes by branch or tag in the branch drop-down list: