DATASTAX

Best Practices in Deploying and Managing DataStax Enterprise

Table of Contents

Abstract

This document provides best practice guidelines for designing, deploying, and managing DataStax Enterprise (DSE) database clusters. Although each application is different and no one-size-fits-all set of recommendations can be applied to every situation, the information contained in this paper summarizes techniques proven to be effective for several DataStax customers who have deployed DataStax Enterprise in large production environments.

This paper does not replace the detailed information found in DataStax online documentation. For help with general installation, configuration, or other similar tasks, please refer to the most recent documentation for your specific DSE release. Note that information and recommendations for hardware selection, sizing, and architecture design is contained in the DataStax Enterprise Reference Architecture white paper.

DataStax Enterprise: The Fastest, Most Scalable Distributed Database Technology for the Internet Enterprise

DataStax Enterprise (DSE), built on Apache Cassandra™, delivers what Internet Enterprises need to compete in today's high-speed, always-on data economy. With in-memory computing capabilities, enterprise-level security, fast and powerful integrated analytics and enterprise search, visual management, and expert support, DataStax Enterprise is the leading distributed database choice for online applications that require fast performance with no downtime. DSE also includes DataStax OpsCenter, which is a visual management and monitoring tool to manage DSE clusters, as well as 24x7x365 expert support and services.

Best Practices for Apache Cassandra

Apache Cassandra is an always-on, massively scalable open source NoSQL database designed to deliver continuous uptime, linear scale performance, and operational simplicity for modern online applications. Cassandra's distributed database capabilities and masterless architecture allow it to easily span multiple data centers and cloud availability zones, which means customers can write and read data anywhere and be guaranteed of very fast performance.

> *"The data gathered by NREL comes in different formats, at different rates, from a wide variety of sensors, meters, and control networks. DataStax aligns it within one scalable database."*
> — Keith Searight, NREL

Foundational Recommendations

Success experienced by the vast majority of Cassandra users can be attributed to:

1. A proper data model
2. An appropriate hardware storage infrastructure

With Cassandra, modeling data as you would with an RDBMS and/or selecting a SAN or other shared storage device for your data versus non-shared storage on multiple nodes greatly increases chances of failure. In general, in managing a Cassandra cluster, you are validating or invalidating its data model and storage architecture selections.

Data Modeling Recommendations

Cassandra is a wide-row store NoSQL database and provides a rich and flexible data model that easily supports modern data types. The importance of a proper data model for an application cannot be overemphasized, with success for a new system nearly always being possible if data is modeled correctly and the proper storage hardware infrastructure is selected.

A running instance of Cassandra can have one or more keyspaces within it, which are akin to a database in RDBMS's such as MySQL and Microsoft SQL Server. Typically, a Cassandra cluster has one keyspace per application. A keyspace can contain one or more column families or tables, which are somewhat similar to an RDBMS table but much more flexible.

A typical best practice is to rarely, if ever, model your data in Cassandra as you have modeled it in an RDBMS. A relational engine is designed for very

normalized tables that minimize the number of columns used where the *exact reverse* is true in Cassandra where the best practice is to heavily de-normalize your data and have tables with many columns.

Each table requires a primary key as shown below in Figure 1. The primary key is made up of two parts: the *Partition Key* and the *Clustering Columns*.

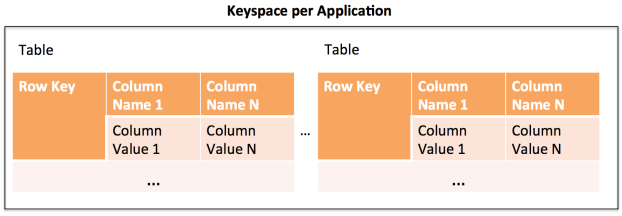

Figure 1 – A keyspace housing two tables

Keep in mind that a column value can never exceed 2GB, with a recommended size per column value being no more than single digit MBs. Also, a row key cannot exceed the maximum of 64KB.

While RDBMS's are designed in a typical Codd-Date fashion of creating entities, relationships, and attributes that are very normalized per table, the Cassandra data model differs in that it is based on the application's query needs and user traffic patterns. This oftentimes means that some tables may have redundancies. Cassandra's built-in data compression manages this redundancy to minimize storage overhead and incur no performance penalty.

Cassandra performs best when data needed to satisfy a given query is located in the same partition key. The data model can be planned so that one row in a single table is used to answer each query. Denormalization via the use of many columns versus rows helps the query performing as one read. This sacrifices disk space in order to reduce the number of disk seeks and the amount of network traffic.

Data modeling is high priority for a Cassandra-based system; hence, DataStax has created a special set of visual tutorials designed to help understand how data modeling data in Cassandra differs from modeling data in an RDBMS. It is recommended that you visit the DataStax data modeling web page to review each tutorial.

Hardware Storage Recommendations

DataStax recommends using SSDs for Cassandra as it provides extremely low-latency response times for random reads while supplying ample sequential write performance for compaction operations.

SSDs are usually not configured optimally by default by the majority of Linux distributions.

The following steps ensures best practice settings for SSDs:

- Ensure that the SysFS *rotational* flag is set to false (zero) as the initial first step. This overrides any detection that the operating system (OS) might do to ensure the drive is definitely considered an SSD by the OS. Also do the same for any block devices created from SSD storage, such as mdarrays.
- Set the IO scheduler to either *deadline* or *noop*. The noop scheduler is an appropriate choice if the target block device is an array of SSDs behind a high-end IO controller that will do its own IO optimization. The deadline scheduler will optimize requests to minimize IO latency. If in doubt, use the deadline scheduler.
- Set the read-ahead value for the block device to 0 (or 8 for slower SSDs). This tells the OS not to read extra bytes, which can increase the time an IO requires and also pollutes the cache with bytes that weren't actually requested by the user.

The table below shows how to implement these settings in an /etc/rc.local file.

```
in /etc/rc.local, assumes /dev/sda is the SSD
echo deadline > /sys/block/sda/queue/scheduler
#OR...
#echo noop > /sys/block/sda/queue/scheduler
echo 0 > /sys/block/sda/queue/rotational
blockdev --setra 0 /dev/sda
#OR, for slower SSDs...
#blockdev --setra 8 /dev/sda
```

For more information on proper storage selection, please refer to the DataStax Reference Architecture paper.

Secondary Index

It is best to avoid using Cassandra's built-in secondary indexes where possible. Instead, it is recommended to denormalize data and manually maintain a dynamic table as a form of an index instead of using a secondary index. If and when secondary indexes are to be used, they should be created only on columns containing low-cardinality data (for example: fields with less than 1000 states).

Configuration Parameter Recommendations

This section contains best practice advice for some of the most important Cassandra configuration parameter options.

Data Distribution Parameters

Data distribution inside a Cassandra database cluster is determined by the type of *partitioner* selected. There are different types of partitioners in Cassandra that can be configured via Cassandra's configuration file: cassandra.yaml.

The *Murmur3Partitioner* and *RandomPartitioner* partitioners uniformly distribute data to each node across the cluster. Read and write requests to the cluster are evenly distributed while using these partitioners. Load balancing is further simplified as each part of the hash range receives an equal number of rows on average. The *Murmur3Partitioner* is the default partitioner in Cassandra 1.2 and above. It provides faster hashing and better performance compared to *RandomPartitioner* and is recommended for your DSE deployments.

The *ByteOrderedPartitioner* keeps an ordered distribution of data lexically by key bytes and is not recommended except in special circumstances. It is only useful during key range queries, but note that you will lose the key benefits of load balancing and even distribution of data, and so much more manual maintenance may be necessary.

To help automatically maintain a cluster's data distribution balance when nodes are added or subtracted from a cluster, a feature called virtual nodes or *vnodes* was introduced in Cassandra 1.2. Enabling vnodes on Cassandra will allow each node to own a large number of small data ranges (defined by tokens) distributed throughout the cluster. The recommended value for the vnodes configuration value *(num_tokens)* is 256 and must be set before starting a cluster for the first time.

Virtual nodes are recommended because they save time and effort where manually generating and assigning tokens to nodes is concerned. Vnodes are enabled by default in Cassandra 2.0 and higher versions. Vnodes should not be used if your DSE cluster contains nodes devoted to analytics (integrated/external Hadoop) or enterprise search. Those node types do not perform well with vnodes.

Memory Parameters

General Memory Recommendations

JVM heap settings are a potential source of trouble for Cassandra and DSE deployments. While administrators may be tempted to use large JVM footprints when DRAM is plentiful on the available nodes, but this can lead to serious problems. JVM uses a garbage collector (GC) to free up unused memory. Large heaps can introduce GC pauses that can lead to latency, or even make a Cassandra node appear to have gone offline. Proper heap settings can minimize the impact of the GC in the JVM.The minimum recommendation for DRAM is 16GB and the heap should be set to 8GB, but note that the minimum recommended by DataStax for production deployments is 32GB of DRAM.

Data Caches

Cassandra supports two types of data caches: a key and row cache. The *key cache* is essentially a cache of the primary key index data for a Cassandra table. It helps save CPU time and memory over just relying on the OS page cache for key data. Enabling just the key cache will still result in disk (or OS page cache) activity to actually read the requested data rows.

It is best to aim for a hit rate of 95% when using the key cache. If your hit rate is higher than 95%, then the cache might be too large.

The *row cache* is more similar to a traditional cache in other databases: when a row is accessed, the entire row is pulled into memory (merging from multiple SStables, if necessary) and cached so that further reads against that row can be satisfied without accessing disk. It is best used when you have a small subset of data to keep hot and you frequently need most or all of the columns returned. DataStax recommends not to use *row cache* unless your use case is > 95% read oriented.

Note that the *row cache* can decrease performance in some situations such as when you have tables that are not cacheable at all.

General Deployment Considerations

The following sections contain recommendations for various Cassandra deployment scenarios.

Node Addition/Removal Practices

Cassandra provides linear scale performance with elastic expansion capabilities so adding capacity to a database cluster is straightforward and easy. General guidelines for adding new nodes to an existing cluster-

- If you are not using virtual nodes (vnodes),

set the *inital_token* value of the new node in the cassandra.yaml file and ensure it balances the ring without token conflicts. This step is not necessary with vnodes.

- Change the cassandra.yaml setting of *auto_bootstrap* to True
- Point the *seed_list* parameter of the new node to the existing *seed_list* that exists within the cluster.
- Set the *listen_address* and, *broadcast_address* parameters to the appropriate addresses of the new node.
- Start the new Cassandra node(s) two minutes apart. Upon bootstrapping, nodes will join the cluster and start accepting new writes, but not reads, while the data these nodes will be responsible for is streamed to them from new nodes.
- Later, the cleanup command can be used on each node to remove stale data. This step is not immediately required.

In the rare instance that existing nodes are removed and capacity of other systems is increased to handle the workload (scaling up versus scaling out), the same steps above are required. The only additional step is removing the old nodes with the decommission operation, one at a time, to lessen impact of the operations on the cluster. When the decommission command is called on a node, the node streams all data for which it is responsible to the new nodes that are taking over responsibility.

Avoiding Downtime by Using Rack Awareness

Cassandra provides the ability to distribute and replicate data among different hardware racks, which helps keep a database cluster online should a particular rack fail while another stays operational. While defining one rack for an entire cluster is the simplest and most common implementation, there are times when data replication between different racks to help ensure continuous database availability is required.

A rack aware snitch ensures high availability within a single datacenter. A snitch is configured at the cluster level and controls the placement of replicas within the cluster.

With the use of a rack-aware snitch, Cassandra gains knowledge of the physical location of each node within the cluster. Cassandra leverages this information to ensure replicas of each row exist in multiple server racks to ensure that rack loss due to fire, mechanical problems, or routine maintenance does not impact availability. Please refer to DataStax documentation for a list of rack-aware snitches and their configuration information.

The following are some general principles that can be followed for using multiple racks:

- Designate nodes in an alternating and equal pattern in your data centers. Doing so allows you to realize the benefits of Cassandra's rack feature, while allowing for quick and fully functional expansions.
- Once a cluster is deployed, you can swap nodes and make the appropriate moves to ensure that nodes are placed in the ring in an alternating fashion with respect to the racks, as shown in Figure 2 below.

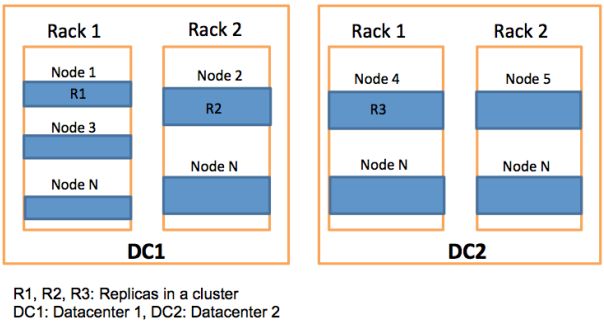

Figure 2 – Multi-data center deployments with multiple racks

Selecting a Compaction Strategy

Compaction is a process in Cassandra where multiple data files (SSTables) are combined to improve the performance of partition scans and to reclaim space from deleted data. Compaction issues can degrade performance in Cassandra so the selection of a proper compaction plan is important. When considering a compaction strategy, the general rule of thumb is that size-tiered compaction (which is the default in Cassandra 2.0 and above) is good when write performance is of primary importance or when rows are always written entirely at once and are never updated. This type of compaction strategy triggers a minor compaction whenever there are a number of similar sized SSTables on disk. Read workloads can suffer due to factors such as compaction getting behind and multiple compactions running simultaneously.

A leveled compaction strategy is worth considering when read performance is of primary importance and is also the recommended strategy if you are using SSDs. It is also useful if the following conditions are present in your database cluster-

- A high read/write ratio
- Rows are frequently updated
- Heavy delete workloads or you have TTL columns in wide rows

More information on compaction strategies can be found in a technical article on the DataStax Tech Blog.

Benchmarking and Assessing Throughput Capacity

The approach taken to benchmarking and testing the capacity of a new database cluster will depend on the type of application it is targeting. Benchmarks such as YCSB and the Cassandra stress tool can be utilized to gain a general idea of how well your initially configured cluster will perform.

When first beginning to work with and test DSE, a general approach is to:

- Run a sample workload against 3 nodes
- Make a note of the point when throughput maxes out and latency becomes an issue for the application
- Run the same workload against 6 nodes in order to get a second interpolation point

Since Cassandra scales linearly, assessing the improvements is typically quite easy (the transaction per second rate doubles, etc.) For examples on benchmarking DataStax Enterprise using YCSB, either standalone or in comparison to other NoSQL databases, see the DataStax Benchmarking Top NoSQL Databases white paper. For recommendations on how not to benchmark Cassandra, see this article on the DataStax Tech Blog.

Recommended Monitoring Practices

This section contains general tips for monitoring and troubleshooting Cassandra performance.

Key Monitoring Tools and Metrics

The primary tools recommended for monitoring Cassandra performance are:

1. CQL Performance Objects
2. DataStax OpsCenter
3. The nodetool utility's tpstats and cfhistograms options

DSE Performance Service

DataStax Enterprise 4.5 and beyond includes a new performance data dictionary containing various diagnostic objects that can be queried from any CQL-based utility. It provides granular details on user activities and other statistics that helps DBAs and operational team to monitor database performance and respond to issues immediately.

There are a number of settings in dse.yaml that need to be enabled for the collection of specific statistics, with the objects being stored in the "dse_perf" keyspace.

For example: - enabling "CQL slow log" will record any CQL statement in a table that takes longer than the certain "set" threshold. A best practice is to monitor this table for any slow queries that affects the performance and take necessary steps to tune it for better performance.

In addition, there are also various performance objects that collect system info, summary stats, histogram and also latency tracking of users/objects.

A sample query is shown below when a query is executed on the "user_io" table that is used to track per node read/write metrics, broken down by client connection and aggregated for all keyspaces and tables.

```
cqlsh:dse_perf> select * from user_io;

 node_ip   | conn_id         | last_activity           | read_latency | total_reads | total_writes | user_ip   | username  | write_latency
-----------+-----------------+-------------------------+--------------+-------------+--------------+-----------+-----------+---------------
 127.0.0.1 | localhost:55405 | 2014-06-19 08:59:09-0700 |            0 |           0 |         1418 | 127.0.0.1 | anonymous |        47.572
 127.0.0.1 | localhost:55735 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       102.67
 127.0.0.1 | localhost:55736 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       105.08
 127.0.0.1 | localhost:55737 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        90.66
 127.0.0.1 | localhost:55738 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       95.905
 127.0.0.1 | localhost:55739 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        98.96
 127.0.0.1 | localhost:55740 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        91.48
 127.0.0.1 | localhost:55741 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        86.74
 127.0.0.1 | localhost:55742 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       104.67
 127.0.0.1 | localhost:55743 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       104.62
 127.0.0.1 | localhost:55744 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |       99.255
 127.0.0.1 | localhost:55745 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        83.58
 127.0.0.1 | localhost:55746 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        82.63
 127.0.0.1 | localhost:55747 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        99.56
 127.0.0.1 | localhost:55748 | 2014-06-19 09:01:39-0700 |            0 |           0 |          200 | 127.0.0.1 | anonymous |        97.58
```

DataStax OpsCenter enables the visual management of key metrics and setting proactive alerts that notify the operator when performance on a database cluster is beginning to degrade. With OpsCenter, administrators can drill into a node and graphically view information from utilities like tpstats.

Nodetool

While several Cassandra statistics are available, those contained in the tpstats (thread pool status) and cfhistograms utilities provide the shortest route in determining whether a database cluster is performing well or not. For detailed information on

how to use both utilities, please see the online documentation for the version of Cassandra you are using.

When reviewing the output of both utilities, general monitoring rules-of-thumb to keep in mind include the following:

For tpstats:

- Any blocks seen for Flushwriters indicates that the disk is getting behind in writes and that Cassandra is trying to flush memtable data to disk but cannot.

For cfhistograms:

- Check to see how many SStable seeks/SStables were accessed (first column in the output). Seeing larger numbers after 1 in the Offset column indicate potential read I/O slowdown's because more SSTables have to be accessed to satisfy a read request.
- Seeing large numbers of operations in the read and write latency columns that are far down in the list (i.e. the Offset column for these statistics equals microseconds, so large Offset values with many operations is usually bad) indicate read and/or write bottlenecks.

These and other key metrics can be monitored visually in OpsCenter. In general, the following characteristics indicate degrading performance for a node or cluster:

- Increasing number of blocked flushwriters
- Many requests for multiple SSTable seeks (as opposed to just one)
- Increasing number of pending compactions
- Increasing number and time of garbage collection operations

Using OpsCenter's built-in auto failover option (available in version 5.1) is recommended where monitoring and managing critical data without any interruption is a requirement. Auto failover option enables users to have multiple instances of OpsCenter with an active-passive setup (only one OpsCenter can be active at a time). This addresses the HA needs by automatically failing over to the standby OpsCenter should a failure in the primary occur. The failover will be transparent to the user with zero interruption in most of the services.

Monitoring and Tuning Garbage Collection

JVM uses a garbage collector (GC) process to free up unused memory. Very large heaps can introduce GC pauses that can increase wait times or even make a Cassandra node look like it is down. Additionally, JVM can throw out of memory exceptions due to heap size being too small or GC not being able to keep up with the workload to free memory quickly enough to be reused by Cassandra.

Basic recommendations for monitoring/tuning garbage collection:

1. Ensure the JVM heap has the correct setting, which at most times equates to 8GB of RAM.
2. Edit cassandra-env.sh, and set heap_newsize parameter to a minimum of 100MB * number of cores (200MB is becoming common).
3. Set up a method to monitor a number of key GC statistics, which can easily be done in DataStax OpsCenter. These statistics include *JVM ParNew collection count, JVM CMS collection time* and *JVM ParNew collection time*.

Tuning involves adjusting the heap_newsize parameter in the following way:

- If the GC collection/pause times are a problem, reduce *heap_newsize*
- If the GC events are too frequent and/or CMS collection time is too expensive, then increase *heap_newsize*

DSE Search: Best Practices

DataStax Enterprise includes an integrated enterprise search component powered by Apache Solr. This allows you to easily search your line-of-business data stored in Cassandra.

General Recommendations for Search Nodes

Several Cassandra best practices can be applied to nodes in a DSE cluster devoted to search. Key differences for properly configuring and deploying DSE-search/Solr nodes include:

- Do not enable virtual nodes/vnodes for a cluster containing Solr nodes.
- If using traditional spinning disks, set up at least 4 volumes with a set of dedicated heads for the operating system, commit log, SSTables, and Solr data. SSDs over spinning disks are strongly encouraged.

- Set the JVM heap size to 14GB.
- Set the *heap_newsize* parameter to 200MB per cpu ore, with maximum not to exceed 1600MB.
- Disable the Cassandra row cache, and leave the Cassandra key cache at default value.
- Set the Solr *soft autocommit* max time to 10s.
- DSE 3.1 and higher uses a custom per-segment filter implementation that uses the *filter cache.* The *filter cache* is the only meaningful cache for Solr nodes; all others should be disabled. Start with a *filter cache* setting of 128. Note that 512 is often too big for a 14G heap and will cause GC pressure. Don't enable auto warming unless you have frequently used filters.
- Ensure the document and query result caches are disabled.
- Set merge factor to 10. This is a good compromise value for most situations. For read only use cases this value may be lowered to 5 or 2, while for write heavy or balanced use cases leave it at 10.
- Ensure that the Lucene version is set to the most recent version supported by the release.
- Set DSE type mapping to the most recent version supported by the release (currently 1). If this value is changed, the Solr metadata must be removed and Solr must be re-enabled. It is recommended that this value remain unchanged unless absolutely necessary.

Sizing Determinations

To determine whether a proposed DSE-search/Solr configuration is sized properly, run through the following steps:

- Install DSE and create a cluster with a Solr node.
- Create a column family with the Solr schema and configuration.
- Load one thousand mock/sample records.
- Get the index size for the Solr core: (example: http://localhost:8983/solr/admin/cores?action=STATUS&memory=true).
- Extrapolate from those numbers the index size for the expected total record count.

For example, if the index size is 1GB, and you expect one million records, then the index size will be 1000GB. The database cluster must be large enough so that the total cluster memory is large enough to cache the total index size, and hot dataset, subtracting for the JVM heap and operating system overhead. Assume 1GB of memory for the operating system and 14 GB of memory for the JVM heap, or an overhead of 15GB.

A useful sizing equation is:
((Nodes * Memory Per Node) - (15GB * Nodes)) / (Index Size * (Expected Rows / 1000))

If the value is less than 1 then you can expect problems (e.g. you won't have enough memory to cache the index, let alone cache the rows; every query will hit the disk multiple times, etc.)

Schema Recommendations

General recommendations for schema design and practice:

- The schema version is defined in the root node of the XML document. Avoid specifying a schema version if possible, and avoid specifying an older version as it may result in undesirable behavior.
- Avoid or limit the use of dynamic fields. Lucene allocates memory for each unique field (column) name, which means if you have a row with columns A,B,C, and another row with D,E, Lucene will allocate 5 chunks of memory. If this were done for millions of rows, it is fairly easy to blow up your heap.
- Instead of using dynamic fields, copy field contents using the *CopyField* directive, and perform queries against the combined field.

Monitoring and Troubleshooting Recommendations for Search

With Search/Solr response times, a general rule of thumb is that search execution times of 20ms or less are considered good. Anything less than 10ms is considered very good. Under heavy load with large datasets, it is not uncommon to see search latencies of 100ms to 200ms. The most common causes for slow queries include:

- The disk/file system is laid out incorrectly.
- There is not enough memory to cache Solr index files and the hot dataset. To verify on Linux, you can check IOwait metrics in top/vmstat/iostat.
- GC pressure can result in high latency variance.
- Un-tuned queries.

Other troubleshooting advice for Solr includes the following:

- If slow startup times are experienced, decrease the commit log and/or SSTable sizes.
- If inconsistent query results are seen, this is

likely due to inconsistent data across the nodes. Either script repair jobs or use the DSE automatic repair service to ensure data consistency across the cluster.

- If manually running repair, and a high system load is experienced during repair operations, then repair is not being run often enough. Switch to using the DSE automatic repair service.
- If dropped mutations are seen in the tpstats utility, then it is likely that the system load is too high for the configuration and additional nodes should be added.
- If read/socket timeouts are experienced, then decrease the *max_solr_concurrency_per_core* parameter in the dse.yaml configuration file to 1 per CPU core.

DSE 4.6 includes a new performance data dictionary for Search containing various diagnostic objects that can be queried from any CQL-based utility.
There are a number of settings in dse.yaml that need to be enabled for the collection of specific statistics, with the objects being stored in the "dse_perf" keyspace.

For example: - enabling "solr slow query log" will record any search statement in a table that takes longer than the certain "set" threshold. A best practice is to monitor this table for any slow queries that affect the performance and to tune for better performance.

In addition, there are also various performance objects that collect latency of query/update/commit/merge, indexing error, index stats and so on.

DSE Analytics: Best Practices

DataStax Enterprise delivers three options for running analytics on Cassandra data:

1. Integrated real-time analytics including in-memory processing, enabled via certified Apache Spark. Included is a Hive-compatible SQL-like interface for Spark
2. Integrated batch analytics using built-in MapReduce, Hive, Pig, Mahout, and Sqoop
3. Integration with external Hadoop vendors (Cloudera and HortonWorks) merging operational information in Cassandra with historical information stored in Hadoop using Hive, Pig etc.

DSE Analytics Option	Data Size Range	Performance
Integrated Spark	Terabytes (TB)	High
External Hadoop Integration	Petabytes (PB)	Low
Integrated Hadoop	Terabytes (TB)	Low

General Recommendations for Analytics Nodes

To properly configure and deploy DSE-Analytics nodes, DataStax makes the following recommendations:

- Enable vnodes while using Spark/Shark on analytic nodes.
- Both integrated and external Hadoop cannot be run on the same node.
- DSE's Spark Streaming capability takes executor slots just like any other spark application. This means that batch Spark jobs cannot be run on the same Spark cluster that is running streaming unless you are manually limiting the resources both applications can use. Because of this limitation it is advised to run streaming applications on a different datacenter (workload isolation) from the datacenter, running batch jobs. All nodes/DC belonging to the same cluster.
- The minimum number of streaming nodes is one although it must have at least two executors (cores) available. If you want redundancy you'll need to have a storage option like MEMORY_ONLY_2 that will duplicate partitions and at least two nodes for this type of configuration.
- It is recommended not to enable vnodes on analytics nodes when using integrated Hadoop or external Hadoop, unless data size is very large and increased latency is not a concern.
- Always start nodes one at a time. The first node started is automatically selected as a JobTracker (JT).
- Set up a reserve JobTracker on a node different than the primary JobTracker (use dsetool movejt for it). Therefore, if the primary JT node goes down, it will automatically fail over to the reserve JT.
- Verify settings in dse-mapred-default.xml and dse-core-default.xml. DSE tries its best

to auto detect hardware settings and adapt parameters appropriately, but they might need further tuning. Please refer to DSE-Hadoop documentation for more information.

- Avoid putting millions of small files into Cassandra File System (CFS). CFS is best optimized for storing files sizes of at least 64 MB (1 block of default size).
- Do not lower the consistency level on CFS from the default values. Using CL.ONE for writes or reads won't increase throughput, but may cause inconsistencies and may lead to incorrect results or job failures.
- Ensure that map tasks are large enough. A single map task should run for at least a few seconds. Adjust the size of the splits (cassandra.input.split.size) that controls the amount of rows read by single map task if necessary. Too many small splits, and too short map tasks, will cause task scheduling overhead that dominates other work.
- Running small jobs which processes tiny data or setting cassandra.input.split.size too high will spawn only a few mappers and may cause hot spots (e.g. one node executing more mappers than another one).

Monitoring and Tuning Recommendations for Analytics

Analytics-enabled nodes (integrated Hadoop) include additional JobTracker and/or TaskTracker threads and CFS operations can increase memtable usage for column-families in the CFS keyspace. Monitor Cassandra JVM memory usage and adjust accordingly if low memory conditions arise. Also, Map/Reduce tasks will be running as separate JVM processes and you'll need to keep some memory in reserve for them as well.

DSE's Spark analytics is much lighter on memory than Hadoop as it does not use separate JVMs per every task. Allow at least 50% more memory or even 2x more for Spark alone than the size of the data planned to cache in order to fully leverage Spark's in-memory caching. Monitor the heap size as larger heaps can introduce GC pauses/issues that can increase wait times.

Spark needs memory for heavy operators like joins. In case of joins, large amounts of RAM must be allocated to ensure one of the sides of the join fits in memory. Additionally, if you plan to run several different Spark applications at the same time, then allocate enough RAM for all of them.

For Shark, a Hive-compatible system built on Spark can store the result of a query in memory. Administrators must plan on allocating additional memory when in-built caching is enabled.

DSE's new Spark Streaming can require high CPU or memory depending on the application workload. If the job requires a high amount of processing of a small amount of data, high CPU (16 cores or more) should be the goal. If the job requires holding a large amount of data in memory and high-performance analytics, then high memory (64GB or more) should be the aim. We also recommend a minimum of 8 cores on each node that runs DSE Spark Streaming.

One key thing to note about Spark streaming setup in DSE is that it requires a minimum of 2 executors to run streaming applications. Also it is important to ensure that no growing backlog of batches waiting to be processed occurs. This can be seen on the spark UI.

A general guideline is to monitor network bandwidth because streaming when writing to Cassandra will require a significant amount of network IO. To lessen this burden more nodes should be added so the load can be distributed over more machines.
Also monitor "Processing Time" and "Scheduling Delay" metrics in the Spark web UI. The first metric is the time to process each batch of data, and the second metric is the time a batch waits in a queue for the processing of previous batches to finish. If the batch processing time is consistently more than the batch interval and/or the queuing delay keeps increasing, then it indicates the application is not able to process the batches as fast as they are being generated and falling behind. Reduce the batch processing time during these situations.

DataStax Enterprise 3.1 and beyond include an auto-tuning capability for common integrated Hadoop configuration parameters for good out-of-the-box performance. Depending on the use case and workload, the following areas can be further tuned.

- Set the *Dfs.block.size* to 128MB or 256MB. The larger the size of data being processed, the larger the data block size should be. The number of map tasks depends on the input data size and the block size; a larger block size will result in fewer map tasks. A larger block size will also reduce the number of partitions stored in the CFS 'sblocks' column family resulting in a smaller footprint for partition-dependent structures, e.g. key cache, bloom filter, index.
- Set *mapred.local.dir* to a separate physical device(s) from Cassandra data files and commit log directories. Map/reduce tasks

process a lot of data and are I/O intensive, so the location for this data should be isolated as much as possible, especially from Cassandra commit log and data directories. A list of directories can be specified to further spread the I/O load.

- Use 80% or higher for the *mapred.reduce.slowstart.completed.maps* value. With the default value, after 5% of the map tasks have completed, reduce tasks will start to copy and shuffle intermediate output; however, the actual reduce won't start until the mappers are all completed, so a low value tends to occupy reducer slots. Using higher values decreases the overlap between mappers and reduces and allows reducers to wait less before starting.

Auto-tuned parameters and other information can be found in this DataStax Technical blog article.

Security: Best Practices

Establishing a SSL connection when clients (CQL, Spark, Shark etc.) are connecting to DSE nodes (as the username and password will be in plain text) and also during node-node communication is recommended.

DSE 4.6 integrates with the industry standard LDAP and Active Directory servers.
There are certain settings in dse.yaml config file under "LdapAuthenticator" section that one can tune- the search cache and the credentials cache. The search cache can be kept for a long time because you wouldn't normally expect a user to be moved within a directory server. The credentials cache settings in DSE is configurable and it depends on internal IT/security policies.

Plan to use PermissiveTransitionalAuthenticator or TransitionalAuthenticator if a directory server admin does not allow the creation of a 'cassandra' user in the directory server. In this case the DSE admin could use the TransitionalAuthenticator to create a super user with a name available in the directory server. After doing this you can switch over to using the LdapAuthenticator. The problem being that you can't create users in cassandra if the AllowAllAuthenticator (default) is being used.

Ensure passwords are encrypted in the config file by running the command "bin/dsetool encryptconfigvalue" that can encrypt "search_password" and "truststore_password" in dse.yaml file.

OpsCenter 5.0 offers granular based security with user creation and role based permission control. This is very useful for organizations dealing with compliance and strict internal security requirements.

Multi-Data Center and Cloud: Best Practices

Cassandra delivers strong support for replicating and synchronizing data across multiple data centers and cloud availability zones. Additionally, Cassandra supports active-active deployments. Several DataStax customers deploy Cassandra across 2 or more data centers and/or availability zones on cloud providers like Amazon to ensure (1) data is kept close to customers in various geographic locations, and (2) their database remains online should a particular data center or cloud availability zone fail.

The sections that follow outline best practices when deploying DSE across multiple data centers (DCs) and the cloud.

General Recommendations

Cassandra replication is configured on a per keyspace and per DC level. The replication factor (RF) used should not exceed the number of nodes in a data center. If this occurs, writes can be rejected (dependent on the consistency level used), but reads are served as long as the desired consistency level is met.

As a general rule, DataStax recommends a replication factor of 3 within a single data center. This protects against data loss due to single machine failure once the data has been persisted to at least two machines. When accessing data, a consistency level of QUORUM is typically used in a single datacenter deployment. The recommended replication strategy for multiple datacenter is "*NetworkTopologyStrategy*".

Multi-Data Center and Geographic Considerations

Best practice for serving customers in multiple geographies is to deploy a copy of data in a data center closest to the customer base. This ensures faster performance for those users.

As an example, consider one DC in the United States and one DC in Europe. This can be either two physical datacenters with each having a virtual data

center (if required) or two data centers in the cloud as shown below.

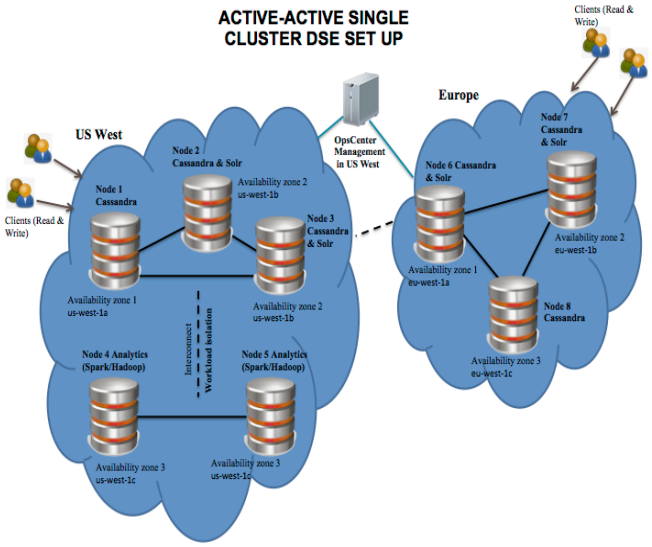

Figure 3 – Using multiple data centers in different geographic regions (cloud)

Best practices for these two data centers:

- Ensure that *LOCAL_QUORUM* is being used for requests and not *EACH_QUORUM* since the latter's latency will negatively impact the end user's performance experience.
- Ensure that the clients to which users connect can only see one data center, based on the list of IPs provided to the client.
- Run repair operations (without the -pr option) more frequently than the required once per *gc_grace_seconds*. You could also use the automated repair service available in DataStax Enterprise to do this.

Natural disasters such as hurricanes, typhoons, and earthquakes can shut down data centers, and cause rolling power outages in a geographic area. When a data center becomes unavailable, there are a number of tasks you may need to perform.

One of the first things to do is to stop client writes to the downed data center. The next task is to add new nodes to the remaining data centers, to handle any increase in load that comes from traffic previously served at the downed data center. When a downed data center is brought back online, a repair operation should be performed to ensure that all data is consistent across all active nodes. If necessary, additional nodes can be decommission and additional nodes added to other DCs to handle the user traffic from the once-downed DC.

Cassandra and DSE are data center aware, meaning that multiple DCs can be used for remote backups or as a failover site in case of a disaster. Note, though, that such an implementation does not guard against situations such as a table mistakenly being dropped or a large amount of data being deleted in error.

An example of using multiple data centers for failover/backup might be the following: an architect may use Amazon EC2 to maintain a primary DC in the U.S. East Coast and a failover DC on the West Coast. The West Coast DC could have a lower RF to save on data storage. As shown in the diagram below, a client application sends requests to EC2's US-East-1 region at a consistency level (CL) of *LOCAL_QUORUM*. EC2's US-West-1 region will serve as a live backup. The replication strategy can reflect a full live backup ({US-East-1: 3, US-West-1: 3}) or a smaller live backup ({US-East-1: 3, US-West-1: 2}) to save costs and disk usage for this regional outage scenario. All clients continue to write to the US-East-1 nodes by ensuring that the client's pools are restricted to just those nodes, to minimize cross data center latency.

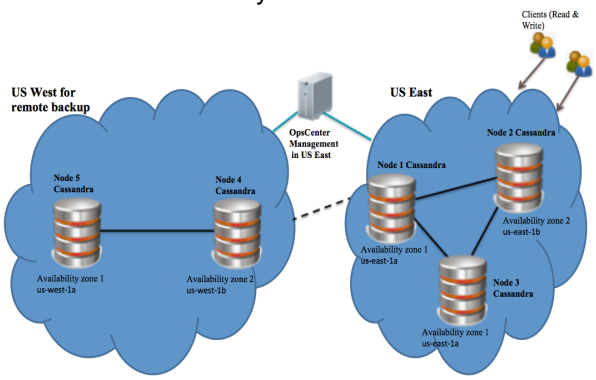

Figure 4 – Setting up a multiple DC configuration for backup or disaster recovery

To implement a better cascading fallback, initially the client's connection pool can be restricted to nodes in the US-East-1 region. In the event of client errors, all requests can retry at a CL of *LOCAL_QUORUM*, for X times, then decrease to a CL of ONE while escalating the appropriate notifications. If the requests are still unsuccessful, using a new connection pool consisting of nodes from the US-West-1 data center, requests can begin contacting US-West-1 at a higher CL, before ultimately dropping down to a CL of ONE. Meanwhile, any writes to US-West-1 can be asynchronously tried on US-East-1 via the client, without waiting for confirmation and logging any errors separately.

More details on these types of scenarios are

available in this DataStax Tech blog article.

Cloud Considerations

This section provides general best practices specifically for the Amazon Web Services (AWS) environment. Any other sizing information (data and instance size) on cloud can be found in DataStax Reference Architecture.

For Apache Cassandra installations in Amazon Web Services' (AWS) Elastic Compute Cloud (EC2), an m1.large is the smallest instance that DataStax recommends for development purposes. Anything below that will have too small a vCPU count, small memory allocation, along with lighter network capacity, all of which are below recommended Cassandra requirements. Also, m1.xlarge is the minimum recommended for low-end production usage, as the stress on an m1.large machine's Java Virtual Machine (JVM) is high during garbage collection (GC) periods.

Another attribute to consider is the number of CPUs. Multiple CPUs are also important if heavy write loads are expected, but ultimately Cassandra is limited by disk contention. Also, one of the most critical attributes to include in your cluster instances are SSDs, and c3.2xlarge would be the best upgrade in such cases. With 2x vCPUs, approximately 4x EC2 Compute Units, and 2x160 SSDs, the c3.2xlarge will be more performant than the m1.xlarge. This option is ideal if your data size isn't too large and latency is more important than cost.

Choosing the right AMI with an official image, up-to-date kernels and EC2 fixes are important factors to consider in a cloud deployment. DataStax recommends using Amazon Linux AMI as it is the most accurately configured AMI for EC2. For i2 instances, use the Amazon Linux AMI 2013.09.02 or any Linux AMI with a version 3.8 or newer kernel for the best I/O performance.

The DataStax AMI uses an Ubuntu 12.04 LTS image and can be found in this link: http://cloud-images.ubuntu.com/locator/ec2.
http://cloud-images.ubuntu.com/locator/ec2
DataStax recommends against EBS devices for storing your data, as Cassandra is a disk-intensive database typically limited by disk I/O. EBS is a service similar to that of legacy Network Attached Storage (NAS) devices, and is not recommended for Cassandra for the following reasons:

- Network disks are a single point of failure (SPOF). Within the NAS infrastructure, you can have multiple layers of redundancy and ways of validation. But ultimately, you will never really know how reliable that is until disks start failing. With Cassandra, nodes may come and go frequently within the ring and you must know that your choice for storage redundancy works as expected.
- If the network goes down, disks will be inaccessible, and thus your database will go offline. Using networked storage for Cassandra data, stored on one device, circumvents the innate, tangible redundancy that a distributed database grants by default. Granted, the network connection between your application and your database can be severed, but this is independent of your database.
- Local disks are faster than NAS. Eliminating the need to traverse the network and stream data across a data center will reduce the latency of each Cassandra operation

EC2 network performance can be inconsistent, so a general recommendation is to increase your *phi_convict_threshold* to 12, in the cassandra.yaml file. Otherwise, you may see issues with flapping nodes, which occur when Cassandra's gossip protocol doesn't recognize a node as being UP anymore and periodically marks it as DOWN before getting the UP notification. Leave the *phi_convict_threshold* at its default setting, unless you see flapping nodes.

Clock skew will happen, especially in a cloud environment, and cause timestamps to get offset. Because Cassandra relies heavily on reliable timestamps to resolve overwrites, keeping clocks in sync is of utmost importance to your Cassandra deployment. If you're handling timestamps in the application tier as well, keeping it in sync there is also highly recommended. You can do this by installing and ensuring the NTP service is active and running successfully.

More details on making good choices for hosting Cassandra on the Amazon Web Services (AWS) environment can be found in this DataStax Tech blog article.

Backup and Restore: Best Practices

Proper backup and restore practices are a critical part of a database maintenance plan, protecting the database against data loss. Cassandra provides snapshot backup capabilities, while OpsCenter Enterprise supplies visual, scheduled backups, restores (including object-level restores), and monitoring tools. Users can also take a table level

backup or backup to a remote location like Amazon S3, perform compression on the backup files or restore to a specific point using point in time restore capabilities (available in OpsCenter 5.1 and beyond).

Backup Recommendations

Disk space is a key consideration when backing up database systems. It is recommended to have enough disk space on any node to accommodate making snapshots of the data files. Snapshots can cause disk usage to grow more quickly over time because a snapshot prevents obsolete data files from being deleted.

OpsCenter's capacity planning services helps monitor and forecast the disk usage for such situations. For example, the graph below shows 20% disk usage cluster-wide with plenty of space for backup files. One could predict the disk usage (including backups) for coming months using the Capacity Service's forecasting feature, as shown in the right side of Figure 5.

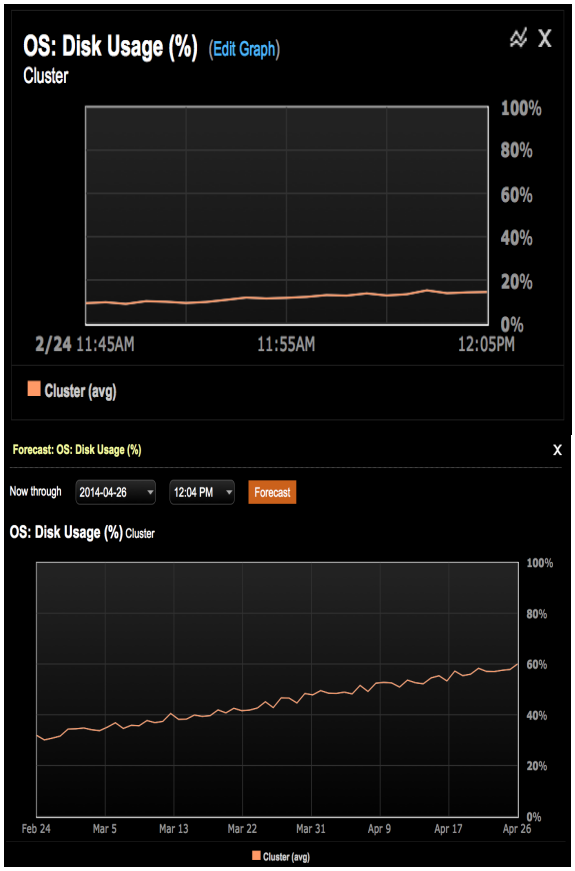

Figure 5 – An example of using OpsCenter to monitor and forecast used disk space

Old snapshot files should be moved to a separate backup location and then cleared on the node as part of your maintenance process. The nodetool clearsnapshot command removes all existing snapshot files from the snapshot directory of each keyspace. A suggestion is for you to move/clear old snapshot files before creating new ones.

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. One way to automate this process is to use OpsCenter. OpsCenter comes with an option to cleanup old backup data, with options for you to specify how long backups should be kept.

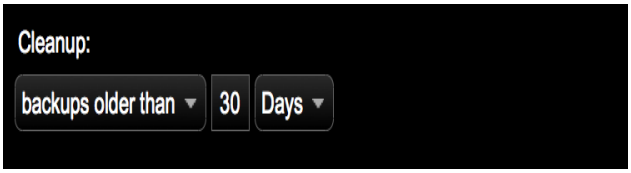

Figure 6 – OpsCenter's automatic backup purge utility

Scheduling backups at regular intervals is also a good practice; this can easily be done through OpsCenter. The frequency of backups will depend on how static or dynamic your data is.

An alternative backup storage strategy to consider is to use Amazon's Simple Storage Service. S3 is an AWS storage service that has configurable security (using Access Control Lists (ACLs)) and tunable redundancy. There are independent S3 services for each region, providing good redundancy, if needed. Note that you can customize OpsCenter's backup utility to store backup data on S3.

Restore Considerations

Restoring data from a backup in the event of data loss or other failure can be accomplished in a number of different ways. OpsCenter's visual restore utility makes the process easy and error-free, and it is recommended you use OpsCenter to do either full or object-level restores when possible.

You can now restore data if stored in Amazon S3 to an existing or a different cluster (available in OpsCenter 5.1 and beyond).

Outside of OpsCenter, either re-create the schema before restoring a snapshot, or truncate tables being targeted for restore. Cassandra can only restore data from a snapshot when the table schema exists.

A snapshot can be manually restored in several ways:

- Use the sstableloader tool.
- Copy the snapshot SSTable directory (see Taking a snapshot) to the data directory (/var/lib/cassandra/data/*keyspace*/*table*/), and then call the JMX method

loadNewSSTables() in the column family MBean for each column family through JConsole. Instead of using the loadNewSSTables() call, users can also use nodetool refresh.

- Use the Node Restart Method.

Software Upgrades: Best Practices

The procedures for upgrading an existing software installation depend on several things such as the current software version used, how far behind it is, and similar considerations. General practices to follow when upgrading an installation include the following:

- Take a snapshot of all keyspaces before the upgrade. Doing so allows you to rollback to the previous version if necessary. Cassandra is able to read data files created by the previous version, but the inverse is not always true. Taking a snapshot is fast, especially if JNA is installed, and takes effectively zero disk space until you start compacting the live data files again.
- Check NEWS.txt and CHANGES.txt for of the target software version any new information about upgrading. Note that the News.txt is on the Apache Cassandra github site.

The order of a node upgrade (taking the node down and restarting it) matters. To help ensure success, you can follow these guidelines:

- Upgrade an entire datacenter before moving on to a different datacenter.
- Upgrade DSE Analytics data centers first, then Cassandra data centers and finally DSE Search datacenters.
- Upgrade the Spark Master/ Job Tracker node in all Analytics datacenters first.
- Upgrade seed nodes before non-seed nodes.

To perform an upgrade with zero downtime, DataStax recommends performing the upgrade as a rolling restart. A rolling upgrade involves performing upgrades to the nodes one at a time without stopping the database.The process for rolling upgrade on each node is as follows:

1. Backup data by taking a snapshot of the node to be upgraded.
2. Run “nodetool drain” on the node. This puts the node into a read-only state and flushes the memtable to disk in preparation for shutting down operations.
3. Stop the node.
4. Install the new software version.
5. Configure the new software (use the yaml_diff tool that filters differences between two cassandra.yaml files).
6. Start the node.
7. Check the logs for warnings, errors and exceptions. Frequent specific errors in the logs are expected and may even provide additional steps that need to be run. You must refer to the documentation for the target version to identify these kinds of errors.
8. DataStax recommends waiting at least 10 minutes after starting the node before moving on to the next node. This allows the node to fully start up, receive hints, and generally become a fully participating member of the cluster.
9. Repeat these steps (1- 8) for each node in the cluster.

Upgrading from DataStax Community (DSC) or Apache Cassandra to DataStax Enterprise is similar to upgrading from one version to another. Before upgrading to DataStax Enterprise, it is recommended to upgrade to the version Cassandra supported in the DSE release you are targeting. The only additional step is configuring the cluster (step 4 above), the snitch needs to move from being specified in the cassandra.yaml file to being specified in the dse.yaml file. Details can be found here. Frequently, upgrades from major versions of Solr, Hadoop, or Cassandra require special steps to be followed during the upgrade process. Please review the recommendations mentioned in the documentation of the version you are targeting.

About DataStax

DataStax, the leading distributed database management system, delivers Apache Cassandra to the world’s most innovative enterprises. Datastax is built to be agile, always-on, and predictably scalable to any size.

DataStax has more than 500 customers in 45 countries including leaders such as Netflix, Rackspace, Pearson Education and Constant Contact, and spans verticals including web, financial services, telecommunications, logistics, and government. Based in Santa Clara, Calif., DataStax is backed by industry-leading investors including Lightspeed Venture Partners, Meritech Capital, and Crosslink Capital. For more information, visit DataStax.com or follow us @DataStax.