

White Paper

Apache Cassandra® NoSQL for the Relational DBA

Lewis DiFelice
Professional Services Consultant

Overview

In this white paper, Instaclustr Professional Services Consultant Lewis DiFelice reflects on his experience of being a long-time SQL Database Administrator to quickly learning how to operate and manage a NoSQL Apache Cassandra® cluster of his own.

Between the key technological differences and an examination of the difficulties in making the switch, Lewis offers a critical roadmap to help others ease their transitions from SQL to NoSQL and avoid ruinous mishaps along the way.

About the author

Lewis DiFelice is a Senior Cassandra Consultant who has been with Instaclustr since March 2020.

During this time he has provided approximately 1500 hours of expert Cassandra consulting to companies ranging from small start-ups to those in the Fortune 500. He is a certified Cassandra Trainer and has conducted Cassandra DBA workshops for many clients.

Lewis began his career at The University of Texas at Austin Computation Center providing expert consulting and non-accredited training courses on using statistical analysis software. He has 20 years of experience in database operations managing relational databases, MS SQL and MySQL, as well as Cassandra.

Prior to working at Instaclustr, he worked as the Lead Cassandra DBA for an online gaming company where he deployed and managed clusters in Amazon Web Service, Google Cloud Platform, and hosted data centers.

He is also the proud father of a rocket scientist! His daughter is a Lead Engineer with one of the teams that monitors the James Webb Space Telescope.



**Discover
more**

Click [here](#) to
discover more
original articles
from Lewis.



Introduction

After nearly 2 decades as a SQL database administrator, my boss came into my office one day to inform me that I would now be managing the company's 40-node NoSQL Apache Cassandra® cluster—and that I would be starting today.

To say the next 6 months were rough is an understatement to put it kindly—not just for me, but for the company as well. Making the change from SQL to NoSQL Cassandra was no easy feat, but through a lot of trial and error, I managed to make it work.

There are some critical differences and difficulties I had in managing and operating a Cassandra cluster after a career as a SQL developer. My goal is to point out the critical differences you must learn to navigate and ensure your transition to NoSQL is far smoother than my own. It is intended for those who will be managing their Cassandra cluster whether it is on-premises or in the cloud.

And here's the first difference you should know: the terminology used with Cassandra is very different from that used with a SQL Server. As such, a glossary of common terms can be found [here](#) and at the end of this article for your convenience.

Ready to make the switch? Let's get started.

Accept you know nothing

My experience taught me that some SQL Server DBAs become dangerous after 6 months on the job. They think they know more than they do, which can lead to some foolish actions.

That was me with Cassandra.

Forget your experience with SQL Server. For example, SQL servers just love more memory—in general, assigning more memory to the data cache improves performance.

With Cassandra, there may be a point of diminishing returns when allocating memory.

Stick to the basics. Do not make changes unless you are sure about their effects.

Above all, be wary of advice or recommendations you find on the Web. Some of it will be wrong. Some will be good but not applicable to your workload. Some will be outdated.

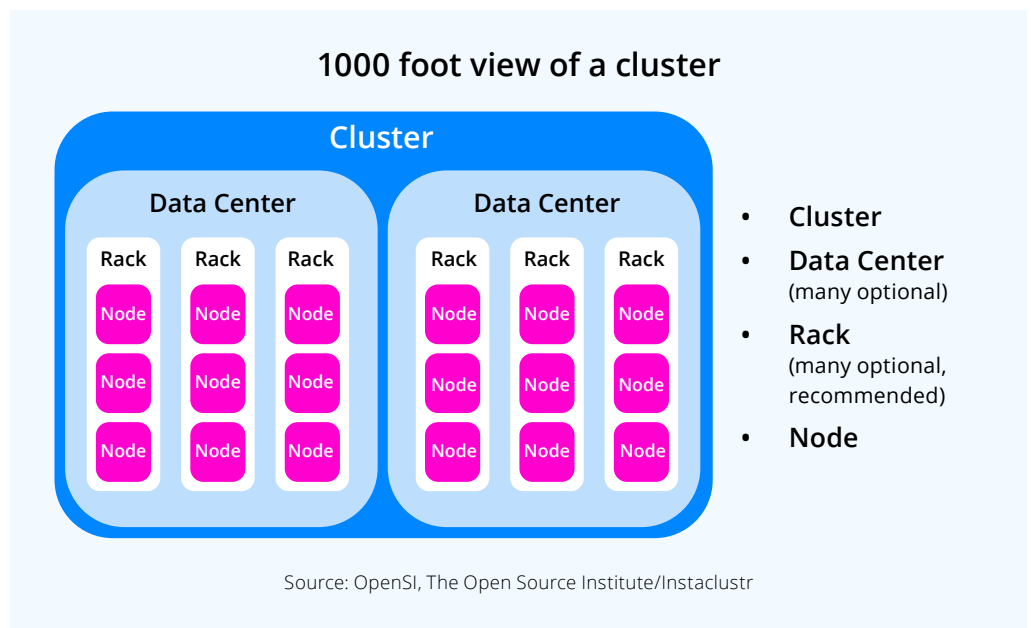
Learn from my errors and stick to the Apache Cassandra documentation.

At the end of this article, I will discuss some trusted sources that I use time and again.

Cassandra is not a SQL server

This is an obvious statement, sure, but I had some experience with SQL Server Cluster and Always On Availability Groups, so I thought I would have some familiarity with Cassandra. I was wrong. Cassandra is different from any of the SQL Server's high availability features.

To briefly review: a Cassandra cluster is a collection of Cassandra instances (or "nodes") working together, assembled like a ring. There is no notion of a "master". All nodes are peers.





A Cassandra database has a shared-nothing architecture with the data itself partitioned among the node.

The operator can choose to make copies of the data (called “replicas”) to increase availability; the copies are then placed onto different nodes.

Changes to the schema (keyspaces and tables) are global. To create a new table, you only need to execute a CREATE TABLE command on one node. The change is replicated in all the other nodes. Data changes that are made on one node are then distributed to the other replicas.

Check [here](#) for an overview of Cassandra’s architecture.

Use the right hardware for the nodes

Cassandra is no different than SQL Server, or any other database for that matter; it just requires the right hardware for a workload. The key advantage of Cassandra is that nodes can be installed on commodity servers.

However, do not mistake “commodity” as meaning the “cheapest possible server on the market”.

This was the mistake my company made when it set up its first Cassandra cluster using hardware left over from an old project. The disk drives were too slow and the result was poor performance and frequent node outages

One smart thing we did before buying the new hardware was to run extensive load tests on a loaner using iometer and [cassandra-stress](#).

Refer to the chapter in the Apache Cassandra document about [hardware choices - it provides incredibly useful information to help you make the right hardware choice](#).

Installing and configuring Cassandra

Cassandra does not have a GUI installer. You can use a package installer (RPM or YUM) or Docker instead. The good news is there are no prompts so writing an installation script is much easier. The package installer automatically installs Cassandra as a service and creates the necessary Linux users.

Cassandra can also be installed from a tarball—a method initially used by my company. This method may be attractive if you are not familiar with Linux (as was my case). The downside is that you cannot run Cassandra as a service unless you write your service file. My recommendation is to use either a package installer or Docker for installation.

Please, please do not run Cassandra on Windows. It is not supported by the Apache Cassandra project and it will be very difficult to find support from the community.


The cluster I inherited back in 2015 ran Cassandra 2.1 on Windows, which was not supported. When I ran into problems, I was on my own. Most of the consultants I tried to hire would politely decline when they learned we were running on Windows. The one piece of advice we did receive was “move to Linux”. I doubt the situation is any better today.

Configuring Cassandra

[Configuring Cassandra](#) can be a complex task, especially at first. Each node has configuration files for Cassandra, logging, cluster topology, environment variables, and the Java Virtual Machine. For package manager installs, configuration files are located in `/etc/cassandra/conf`.

The good news is that if you are starting with a new cluster, you only need to set the [main runtime properties](#) and the directory locations if you decide to change the default paths. As your Cassandra matures you will likely have to change some “tuning” properties.

Avoid 2 major mistakes I made. One was changing properties without completely understanding their function and effect (in my early days, one set of changes decreased performance). This led to a rather uncomfortable discussion with my boss.



Second, I did not know the advantages of using a rack-aware topology. (Hint: this uses the `cassandra-rackdc.properties` file). A number of later problems could have been avoided if I had known to do so.

In the beginning, you will need advice on tuning Cassandra. The `cassandra.yaml` and `jvm.options` files have a number of notes and recommendations for production usage. More detailed information can be found in the Apache Cassandra [documentation](#).

Be wary of using articles you find on the web by asking yourself a few questions:

- Is the article referring to current versions of Cassandra?
- Are their workloads and environment similar to mine?
- Is the author a trustworthy source?

Cassandra is a Java-based application which means that it runs on the Java Virtual Machine (JVM). Configuring the JVM is a complex topic. I started with zero knowledge, and even after 7 years I am still learning. My advice is to set only the heap size until you have time to learn or consult with an expert. Talking to experienced Java developers is a good starting point.

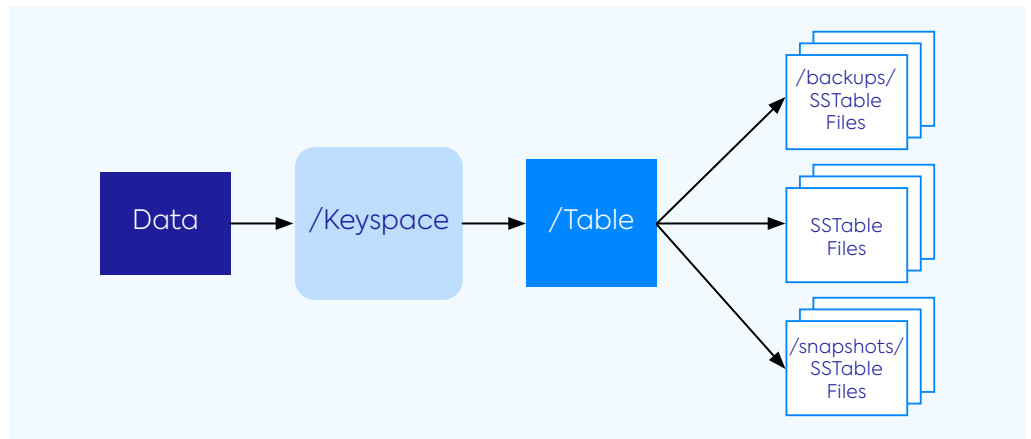
Data layout is folder based

By default, the data is located in `/var/lib/cassandra`. You can change the location of data by setting the property **“`data_file_directories`” in the `cassandra.yaml` file**. It is not necessary to use RAID on the data disks because data is replicated across the cluster based on the replication factor you’ve chosen.

Creating keyspaces and tables will create folders in the data directory. At the top level are the keyspace directories which have the same name as the keyspace. Keyspace tables are stored in subdirectories under the keyspace. The table folders take the name of the table plus a unique identifier (see diagram on next page)

Table data is stored on disk in [Sorted String Table](#) (SSTables) which are immutable. An SSTable is composed of multiple components stored in separate OS files.

For more information about the data directory, see the Cassandra documentation of the [Data Directory Structure](#).



Design based on 'Directory Structure for Cassandra Data' from the Apache Software Foundation found here: <https://cassandra.apache.org/doc/latest/cassandra/operating/backups.html#data-directory-structure>

Operating Cassandra


Cassandra clusters require constant monitoring and management—specifically, that means monitoring and managing the individual nodes.

There is no GUI management tool with Apache Cassandra. The main tool for managing and monitoring a Cassandra cluster is [nodetool](#), a command line utility. Nodetool operates only on the local node. For example, you must run nodetool on each node in order to make a change or run a command across the cluster.

Apache Cassandra has nothing equivalent to the SQL Server Agent. If you need a special task, you write it using a scripting language or Java. Scheduling tasks are done with the cron command.

On day 1 of taking over my company's Cassandra cluster, I had a one-page "cheat sheet" and a batch file that backed up the nodes every 24 hours. My first task was to write scripts to perform common tasks such as backups, restores, and repairs (more on repairs below).

Monitoring is important with any database, but it is absolutely necessary with Cassandra. You **cannot** operate a Cassandra cluster without monitoring. We built our solution with Zabbix for gathering and displaying metrics and Splunk for parsing and analyzing log files. Today, there are 2 good open source tools for monitoring: [Prometheus](#) for collecting metrics and [Grafana](#) for displaying metrics.



While it might be possible to manually deploy and manage a 3-node cluster it is not feasible to do so with larger clusters. You need a tool that can distribute commands in parallel across many servers. For that, we used [Salt](#), an open source configuration management and orchestration tool. Once I had developed the required state files, I could deploy or modify a Cassandra cluster in minutes.

It is also important to know a scripting language, such as Bash or Python, or Java for creating tasks and tools. Many companies have contributed tools to Github—more on that below.

For detailed information, see [Operating Cassandra](#) in the Apache documentation.

Backup and restores

First of all, do not waste time, as I did, looking for the equivalent of RESTORE DATABASE. There isn't one. More on this later.

Second, backups are strange to a SQL Server DBA.

Cassandra supports 2 types of backups: snapshots and incremental backups. You can create a snapshot with [nodetool snapshot](#) command. Certain actions such as truncating or dropping a table will automatically trigger a snapshot.


Incremental backups must be enabled in the `cassandra.yaml` file. Once enabled, incremental backups are automatic. The operator has no control over the process except to disable it.

Both types of backups create a copy of a table's SSTable files using hard links and then store the copies in subdirectories under each table directory.

You **cannot** change the destination for either type of backup. You must create a task to move snapshots and backups to a remote location.

There is no BACKUP CLUSTER command. Each node must be backed up individually.

A restore in Cassandra is the reverse of the backup process: You copy the snapshot files and the incremental backup files are used, back to the original data location on the nodes.



Other than having `nodetool snapshot` and `nodetool clearsnapshot`, you are on your own. Incremental backups lack any tools.

It might be possible to perform backups and restores manually on a small cluster but eventually, you will need to create some tools. After 2 weeks of creating snapshots manually for every node, I wrote a simple bash script to manage snapshots and incremental backups.

If you wish to avoid the work of creating your own tooling, Instaclustr provides a free command line tool, [Esop](#), which can be downloaded from GitHub.

New operational tasks

So far, all the tasks we have discussed share similarities between SQL and NoSQL. When you start to work with Cassandra, you will be dealing with some new ones.

Repairs

This will be a new, not much fun, operational task.

In a distributed environment such as Cassandra, replicas can become out of sync if a node should fail or a write is dropped. A repair is a process to synchronize data between replicas. Cassandra does not automatically run repairs. It is run by the operator via the `nodetool repair` command.

Repairs must be run regularly on all nodes in the cluster so an automated process is essential.

Repairs can take a long time to complete and can impact performance while running.

I cobbled together bash scripts and cron jobs to schedule and run repairs. Luckily, you do not need to do this. There is an open source tool called [cassandra-reaper](#) that provides a simple web-based GUI for scheduling and managing repairs. I highly recommend it.

Tombstones

Another task that proved difficult was managing tombstones. In Cassandra, a `DELETE` command does not remove data. Instead, it inserts a special record called a tombstone that contains information about the data which was “deleted”. The tombstones, and the

referenced data, are eventually purged after a specified time has elapsed (the actual process is more complicated but this is not the place to discuss it).

What you must keep in mind is that too many tombstones can increase read latency and, in extreme cases, cause queries to fail.

It took me a while to understand tombstones and how to manage them. However, I did not have available this excellent article on [Managing Tombstones](#).

Replacing catalog system stored procedures

Cassandra has no exact analog to [Catalog Stored Procedures](#). The equivalent function is provided by nodetool and CQL commands. Virtual tables, which first appeared in Cassandra 4.0, expose metrics and metadata similar to Dynamic Management Views.

On my first day working with Cassandra, I made a reference table to help me keep track of how routine functions like Catalog Stored Procedures were done with Cassandra:

Function	SQL Server Catalog Stored Procedure	Cassandra Nodetool Command	Cassandra CQL
Return cluster information		status describecluster ring	
Return server information	sp_server_info	info	
List databases	sp_databases sp_helpdb		desc keyspaces
List tables in current database/ keyspace	sp_tables	cfstats	desc tables
List columns in table	sp_columns		desc table
List users in database	sp_helprolemember sp_helprole		list roles
List active users	sp_who2		The clients virtual table (V4 only)

A mapping of system tables

Just as SQL Server has system databases to store metadata, Cassandra has system keyspaces.

SQL Server System Database	Cassandra Equivalent System Keyspace	Notes
Master	system system_auth	
information_schema	system_schema	
Resource	system_views system_virtual_tables	Cassandra 4.0 and above

Support for Apache Cassandra on the web

Throughout my time as a Cassandra DBA, I've found many resources on the web to be incredibly helpful.

The official Apache Cassandra documentation can be found [here](#).


Support for Cassandra is provided by community forums such as Stack Exchange, Slack, and mailing lists. See [Cassandra Community](#) for more information.

Instaclustr has an extensive [content library](#) of free articles related to Cassandra.

How Instaclustr Can Help You Make the Transition

Instaclustr [Consulting Services](#) offers tailored packages that will help make your first Cassandra project a success. Several of the consultants are former SQL Server DBAs who understand the challenges you face.

[Training Services](#) provides open enrollment workshops on both Cassandra Development and Operations. We can also provide custom workshops for companies.



The [Instaclustr Managed Platform for Apache Cassandra®](#) is an SOC 2 certified, fully managed service hosted in the cloud or on-prem, customized, and optimized so you can focus on scaling your applications.

Whether on-prem or in the cloud, Instaclustr offers 24x7 [Expert and Dedicated Support](#) for Apache Cassandra and related data infrastructure. Contact us to discuss any specific Cassandra requirements you may have, including security needs and SLAs.

Conclusion

There are certainly more differences than presented here. I selected the ones which posed the greatest challenges in my transition from an experienced SQL Server DBA to a new, and sometimes lost, Cassandra DBA.

Cassandra is no better or worse than SQL Server—it is just different. Each has its place in the database world.

I am often asked if I prefer to work with one over the other, and for me, the answer is simple: I prefer Cassandra. The challenges it has presented have forced me to develop an entirely new set of skills and further challenged me to continue learning.

At this rate, I will happily be doing just that until the day I retire.

Glossary of terms

The terminology used with Cassandra can be strange at first as it is quite different from SQL. Here are some of the more common terms you will encounter.

Node - An instance of Cassandra.

Cluster - A collection of nodes working together.

Datacenter - A subdivision of a cluster. A cluster can have multiple datacenters which can be geographically separated. Data is replicated among the datacenters.

Keyspace - The Cassandra equivalent of a database. It serves as the namespace for the table. It is also used to define the replication factor of the keyspace for each datacenter.

Table - The fundamental object for storing data. Formerly known as a column family.

SSTable (Sorted String Table) - An immutable data file that Cassandra uses for persisting data on a disk. A table is composed of one or more SSTables. Each SSTable is composed of multiple components stored in separate files.

Commitlog - An append-only log of all mutations (data changes) local to a Cassandra node. Any data written to Cassandra will first be written to a commit log before being written to a memtable.

Compaction - A process of reconciling various copies of data spread across distinct SSTables. Cassandra performs compaction of SSTables as a background activity.

Compaction Strategy - A table-level option that determines how compaction is performed.

Consistency Level - The number of replicas that must successfully respond to a request.

CQL (Cassandra Query Language) - The API for querying a Cassandra database. Its syntax is very similar to SQL.

cqlsh - CQL command line shell is a command-line interface for interacting with Cassandra using CQL. It is the Cassandra equivalent of **sqlcmd**.

Gossip - The protocol that is used to discover the location and state information about the other nodes participating in a Cassandra cluster.

Memtable - A memory structure where Cassandra buffers write data. In general, there is one active memtable per table. Eventually, a memtable is flushed onto the disk and becomes an SSTable.

Mutation - A write operation that changes the data. In Cassandra, deletes are mutations.

Nodetool - A command-line utility for managing nodes.

Partition - A group of rows that share the same partition key. It is how Cassandra shards data. The location of the partition is a function of its hash value. For more information see [Apache Cassandra Data Partitioning](#).

Partitioner - Determines how data is distributed across the nodes in the cluster (including replicas). The default is Murmur3Partitioner which uniformly distributes data across the cluster based on MurmurHash hash values.

Repairs - The synchronization of data among replicas. Anti-entropy repairs are run by command ([nodetool repair](#)). Read repairs run in the background.

Replication Factor - The number of copies, called replicas, that will be made for each data row. Each of these replicas is stored on a different node to ensure fault tolerance. The replication factor is defined at the keyspace.

Tombstone - A special marker written whenever data is deleted, the NULL value is inserted, or a non-frozen collection is used. Excess tombstones can cause long GC pauses, latency, read failures, or out-of-heap errors. For more information see [Managing Tombstones](#) in Cassandra.

About Instaclustr

Instaclustr helps organizations deliver applications at scale through its managed platform for open source technologies such as Apache Cassandra®, Apache Kafka®, Apache Spark™, Redis™, OpenSearch®, PostgreSQL®, and Cadence®.

Instaclustr combines a complete data infrastructure environment with hands-on technology expertise to ensure ongoing performance and optimization. By removing the infrastructure complexity, we enable companies to focus internal development and operational resources on building cutting edge customer-facing applications at lower cost. Instaclustr customers include some of the largest and most innovative Fortune 500 companies.

© 2023 Instaclustr Copyright | Apache®, Apache Cassandra®, Apache Kafka®, Apache Spark™, and Apache ZooKeeper™ are trademarks of The Apache Software Foundation. Elasticsearch™ and Kibana™ are trademarks for Elasticsearch BV. Kubernetes® is a registered trademark of the Linux Foundation. OpenSearch is a registered trademark of Amazon Web Services. Postgres®, PostgreSQL® and the Slonik Logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission. Redis™ is a trademark of Redis Labs Ltd. *Any rights therein are reserved to Redis Labs Ltd. Cadence is a trademark of Uber Technologies, Inc. Any use by Instaclustr Pty Limited is for referential purposes only and does not indicate any sponsorship, endorsement or affiliation between Redis and Instaclustr Pty Limited. All product and service names used in this website are for identification purposes only and do not imply endorsement.