

Database Architects and Administrators Guide to NoSQL

As a database administrator (DBA) or architect, your job is to help define, develop, manage, and guard your company's single most important asset—data.

The meteoric rise of modern cloud applications—applications that create and leverage real-time value and run at epic scale—has brought about a change in data management with an unprecedented transformation to the decades-old way that databases have been designed and operated. Requirements from cloud applications have pushed beyond the boundaries of the relational database management system (RDBMS) and have introduced a new type of database—NoSQL. The latest [*IDC, Morgan Stanley analyst report predicts*](#) that NoSQL databases will dominate in “New Stack” software development spending.

As a DBA or architect, you may naturally be skeptical of new database systems, having seen database engines such as object-oriented and OLAP databases come and go. Why should NoSQL be any different?

This guide was created to help answer all these questions and more. In the following pages, you'll learn exactly what NoSQL is, why it's needed, how it works, what it should be used for, and (just as importantly) when it shouldn't be used.

You'll also learn how all the key areas of database administration work—database design, creation, security, object management, data replication, backup/recovery, monitoring and tuning, data migrations, and more—and how they are carried out in a NoSQL database like Apache Cassandra.

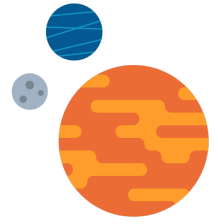
When you're finished, you'll discover how being an administrator or architect for NoSQL platforms like Cassandra is easy. You'll also understand why having NoSQL database skills makes you even more valuable today.

In fact, you may be interested to learn that salaries for DBAs and architects who possess NoSQL and other big data skills are significantly higher than the average RDBMS salary. Further, the [2019 Dice Tech Report](#) found the rewards of learning Cassandra to be greater than other NoSQL or relational databases, and Cassandra placed among the top five technologies to know.

Now, let's get started.



Why NoSQL?



The RDBMS has been the *de facto* standard for managing data since it was first made available from IBM in the mid-1980s.

The RDBMS really exploded in the '90s with Oracle, Sybase, Microsoft SQL Server, and other similar databases appearing in the data centers of nearly every enterprise—databases you likely use today.

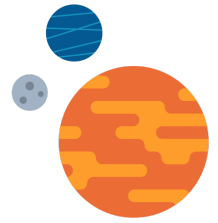
With the first wave of web applications, open source relational database management systems such as MySQL and Postgres emerged and became a standard at many companies that desired alternatives to expensive proprietary databases sold by vendors such as Oracle.

However, it wasn't long before things began to change, and the application and data center requirements of key internet players like Amazon, Facebook, and Google began to outgrow the RDBMS for certain types of applications. The need for more flexible data models that supported agile development methodologies and the requirements to consume large amounts of fast-incoming data from millions of cloud application users around the globe—while maintaining high levels of performance and uptime—necessitated the introduction of a new data management platform.

Enter NoSQL.

Today, with every company utilizing modern cloud applications, the data problems originally encountered by the internet giants have become common issues for every company, including yours. This means that you and your team of DBAs must realize

that it is no longer a question of if you will be deploying and managing NoSQL database systems, but when and how much of your company's data will eventually be stored on NoSQL platforms.



This chapter introduces the basics of NoSQL and then dives into a perspective on the most scalable and performant NoSQL database in the market today, Cassandra.

Types of NoSQL Databases

There are different types of NoSQL databases, with the primary difference characterized by the underlying data model and method for storing data. The main categories of NoSQL databases are:

- ➔ **Tabular** - Also known as wide-column or wide-row stores, these databases store data in rows and users are able to perform some query operations via column-based access. A wide-row store offers very high performance and a highly scalable architecture. Examples include: Cassandra, HBase, and Google Bigtable.
- ➔ **Key/Value** - These NoSQL databases are some of the least complex as all of the data within consists of an indexed key and a value. Examples include: Amazon DynamoDB and Oracle NoSQL database. Some tabular NoSQL databases, like Cassandra, can also service key/value needs.
- ➔ **Document** - Expands on the basic idea of key-value stores where “documents” are more complex, in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, oftentimes stored as JSON. Examples include: MongoDB and CouchDB. Note that some RDBMS and NoSQL databases outside of pure document stores are able to store and query JSON documents.
- ➔ **Graph** - Designed for highly complex and connected data, which outpaces the relationship and JOIN capabilities of an RDBMS. Graph databases are often exceptionally good at finding commonalities and anomalies among large data sets. Examples include: DataStax Enterprise Graph, Neo4j, JanusGraph, and Amazon Neptune.

One trend that is starting to emerge in both the NoSQL and RDBMS markets is the “multi-model” database. Most database management systems are organized around a single data model that determines how data can be organized, stored, and manipulated. By contrast, a multi-model database is designed to support multiple data models against a single, integrated backend.

The value supplied by multi-model databases is that an enterprise doesn’t have to utilize multiple data management providers for applications that need to store parts of the system’s data in different data models, and thus the requirement to shard the application across different database platforms is removed.

What are the Advantages of NoSQL Over an RDBMS?

While there are hundreds of different “Not Only SQL” (NoSQL) databases offered today, each with its own particular features and benefits, what you should know that a NoSQL database generally differs from a traditional RDBMS in the following ways:

- ➔ **Data model** - while an RDBMS primarily handles structured data in a rigid data model, a NoSQL database typically provides a more flexible and fluid data model and can be more adept at serving the agile development methodologies used for modern cloud applications. Note that one misconception about NoSQL data models is that they do not handle structured data, which is untrue. Lastly, as mentioned above, some NoSQL engines are designed to support multiple data models against a single backend.
- ➔ **Architecture** - Whereas an RDBMS database is normally architected in a centralized, scale-up or sharding, master-slave fashion, NoSQL systems such as Cassandra operate in a distributed, scale-out, masterless manner (i.e. there is no “master” node, all nodes are equal). However, some NoSQL databases like MongoDB, DynamoDB, Azure Cosmos DB, and HBase are master-slave or multi-master by design.
- ➔ **Data distribution model** - Because of their master-slave architectures, an RDBMS distributes data to slave machines that can act as read-only copies of the data and/or failover for the primary machine. By contrast, a NoSQL database like Cassandra that distributes data evenly to all nodes making up a database cluster and enables both reads and writes on all machines. Furthermore, the replication model of an RDBMS (including master-to-master) is not designed well for wide-scale, multi-geographical replication and synchronization of data between different locales and cloud availability zones, whereas Cassandra’s replication was built from the ground up to handle such things.
- ➔ **Availability model** - An RDBMS typically uses a failover design where a master fails over to a slave machine, whereas a NoSQL system like Cassandra is masterless and provides redundancy of both data and function on each node so that it offers continuous availability with no downtime versus simple high availability in the way an RDBMS does.

- ➔ **Scaling and Performance model** - An RDBMS typically scales vertically by adding extra CPU, RAM, etc., to a centralized machine, whereas a NoSQL database like Cassandra scales horizontally by adding extra nodes that deliver increased scale and performance in a linear manner.

There's little doubt that relational database management systems will be around for a long time and are exactly the right kind of database for handling centralized applications that require sophisticated transaction handling. But it's also true that NoSQL databases are clearly better to support widely distributed cloud applications and their specific use cases.

Deciding Between an RDBMS and NoSQL

How do you decide when to use an RDBMS and when to use a specific type of NoSQL database? In short, an RDBMS is great for centralized applications that need ACID transactions and whose data fits well within the relational data model. The following chart provides a general comparison between the characteristics that point toward an RDBMS vs. those that signal a NoSQL database may be a better choice:

RDBMS	NOSQL DATABASE LIKE CASSANDRA
Master-slave architecture	Masterless architecture
Relational, structured data	Multi-model (tabular, key/value, document, graph)
Moderate velocity data	High velocity data (time-series data from devices, sensors, etc.)
Always strongly consistent	Tunable consistency (eventual to strong)
Complex/nested transactions	Lightweight transactions
Protect uptime via failover/log shipping	Protect uptime via distributed and fault-tolerant architecture
High availability	Continuous availability
Scale up for more users/data	Scale out for more users/data
Maintain data volumes with purge	High data volumes; retain forever; horizontal scalability without boundaries
Transaction workloads	Mixed workloads of transactions and analytics

FIGURE 1 Top ten determining requirements – RDBMS versus NoSQL

Looking at the data model requirements is another tactic to use when evaluating an RDBMS versus. NoSQL. Certain NoSQL databases require the denormalization of data and aren't concerned with the relationships between data entities whereas others are built to handle complex and very intense data relationship scenarios:

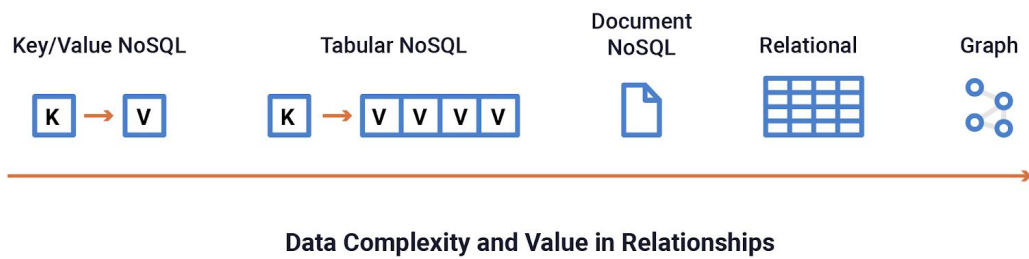


FIGURE 2 The data model continuum by data complexity and connectedness.

RDBMS and graph databases are at the high end of the data model continuum where the relationships between data are concerned and are somewhat similar in their base characteristics:

	RDBMS	GRAPH DB
An identifiable “something” to keep track of	Entity	Vertex
A connection or reference between two objects	Relationship	Edge
A characteristic of an object	Attribute	Property

One of the key differences between a graph database and an RDBMS is how relationships between entities/vertexes are prioritized and managed. While an RDBMS uses mechanisms like foreign keys to connect entities in a secondary fashion, edges in a graph database are of first-order importance. As such, a graph database is more scalable and performant than an RDBMS when it comes to complex data that is highly connected (e.g., millions or billions of relationships).

Unlike most other ways of displaying data, graphs are foundationally designed to express relatedness. Graph databases can uncover patterns that are difficult to detect when using traditional representations, such as RDBMS tables.

Below are suggestions for when to use a graph database versus an RDBMS:

RDBMS	GRAPH
Simple to moderate data complexity	Heavy data complexity
Hundreds of potential relationships	Hundreds of thousands to millions or billions of potential relationships
Moderate JOIN operations with good performance	Heavy to extreme JOIN operations required
Infrequent to no data model changes	Constantly changing and evolving data model
Static to semi-static data changes	Dynamic and constantly changing data
Primarily structured data	Structured and unstructured data
Nested or complex transactions	Simple transactions
Always strongly consistent	Tunable consistency (eventual to strong)
Moderate incoming data velocity	High incoming data velocity (e.g., sensors)
High availability (handled with failover)	Continuous availability (no downtime)
Centralized application that is location dependent (e.g., single location), especially for write operations and not just read	Distributed application that is location independent (multiple locations involving multiple data centers and/or clouds) for write and read operations

A NoSQL Example – Apache Cassandra

Now that you have a background on how NoSQL differs from an RDBMS, let's look a little more closely from a DBA and architect's point of view at how a NoSQL database like Cassandra functions and discuss the above characteristics in detail.

Cassandra is a massively scalable open source NoSQL database. It delivers continuous availability, linear scale performance, operational simplicity, and easy data distribution across multiple data centers and cloud availability zones. Cassandra, was originally developed at Facebook and sports a design combining capabilities from Amazon's Dynamo and Google's Bigtable architectures. It was open sourced in 2008.

What Makes Cassandra Ideal for Modern Cloud Applications

Cassandra provides a number of key features and benefits to facilitate the development and management of cloud applications:

- **Massively scalable architecture** - Cassandra has a masterless design where all nodes are the same, providing operational simplicity and easy scale-out capabilities.
- **Multi-data center replication** - Cassandra provides strong cross data center (in multiple geographies), multi-cloud availability zone support for writes/reads.
- **Active everywhere design** - All Cassandra nodes may be written to and read from no matter where they are located.
- **Linear scale performance** - Online node additions produce predictable increases in performance. For example, if two nodes produce 200K transactions/sec, four nodes will deliver 400K transactions/sec, and eight nodes, 800K transactions/sec.
- **Continuous availability** - DSE offers redundancy of both data and function, which supply no single point of failure and constant uptime.
- **Transparent fault detection and recovery** - Nodes that fail can easily be restored or replaced.
- **Flexible and dynamic data models** - Primarily supports the tabular data model, but also accommodates JSON well.
- **Strong data protection** - A commit log design ensures no data loss for incoming transactions. Also, built-in security with easy backup/restore keeps data protected.
- **Basic transaction support with tunable data consistency** - Cassandra supports the atomicity, isolation, and durability of transactions (including batch) with strong or eventual data consistency supplied across a widely distributed cluster.
- **Data compression** - Data compressed up to 80% without performance overhead helps save on storage costs.
- **CQL (Cassandra Query Language)** - A SQL-like language that makes moving from an RDBMS easy.

Top Use Cases

While Cassandra is a general purpose NoSQL database used for a variety of different applications in all industries, there are a number of use cases where the database excels over most any other option. These include:

- ➔ **Cloud-native applications** - Applications that require the wide distribution of data, no downtime, predictable performance no matter the location of the user, and easy scale make good targets for Cassandra.
- ➔ **Personalization** - Customization of an experience tailored to a particular individual including personal UX features, languages in response bots, or other automated, intelligent, and timely interactions with each customer. DataStax provides personalization in real-time at the moment of interaction within the application (e.g., in commerce), and personalization ensures webpages, searches, checkout processes, and core aspects of the shopping experience are tailored for each user, making it easier, more enjoyable, and faster.
- ➔ **Supply chain management** - Successful supply chains have an end-to-end understanding of their entire network while being able to connect with multiple partners and perform real-time and efficient operations.
- ➔ **Product catalogs and retail apps** - For retailers that need durable shopping cart protection, fast product catalog input and lookups, and similar retail application support, Cassandra is a good choice.
- ➔ **Customer 360** - This is a single view of all customer touchpoints across all mediums. Cassandra drives a system of engagement across channels and systems of record.
- ➔ **Other time-series-based applications** - Because of Cassandra's fast write capabilities and wide-row design, it is well-suited for any time-series-based application.
- ➔ **User activity tracking and monitoring** - Media, gaming, and entertainment companies use Cassandra to track and monitor the activity of users' interactions with their movies, music, games, website, and online applications.
- ➔ **Messaging** - Cassandra serves as the database backbone for numerous mobile phone, telecommunication, cable/wireless, and messaging providers' applications.
- ➔ **Security** - Whether understanding network attacks with vector analysis or detecting intrusions, common security use cases consume a massive amount of data and must successfully surface that data in real time to identify and prevent or stop threats.
- ➔ **Social media analytics and recommendation engines** - Online companies, websites, and social media providers use Cassandra to ingest, analyze, and provide analysis and recommendations to their customers.

- ➔ **Internet of Things (IoT)** - Cassandra is good for consuming and analyzing lots of fast, incoming data from devices, sensors, and similar mechanisms that exist in many different locations.
- ➔ **Fraud detection** - Solutions monitor and analyze data transactions to identify out-of-the ordinary patterns to act in real time, which is required in a broad array of industries and use cases.

Architecture Overview

The architecture of Cassandra allows the database to scale and perform with no downtime. Rather than using a legacy RDBMS master-slave or a manual and difficult-to-maintain sharded design, Cassandra has a masterless “ring” distributed architecture that is elegant, and easy to set up and maintain.

In Cassandra, all nodes are the same; there is no concept of a master node, with all nodes communicating with each other via a gossip protocol.

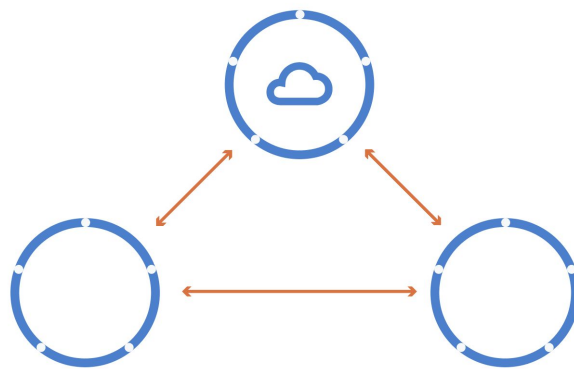


FIGURE 3 Cassandra sports a masterless “ring” architecture.

Cassandra’s built-for-scale architecture allows it to handle large amounts of data and thousands of concurrent users/operations per second, across multiple data centers, as easily as it can manage much smaller amounts of data and user traffic. To add more capacity, you simply add new nodes in an online fashion to an existing cluster.

Cassandra’s architecture also means that, unlike other master-slave or sharded systems, it has no single point of failure and therefore offers true continuous availability and uptime.

Writing and Reading Data

One of Cassandra's hallmarks is its fast I/O operation capability for both writing and reading data. Data is written to Cassandra in a way that provides both full data durability and high performance. From a high-level perspective, data written to a Cassandra node is first recorded in a commit log and then written to a memory-based structure called a memtable. When a memtable's size exceeds a configurable threshold, the data is flushed to disk and written to an SStable (sorted strings table), which is immutable.

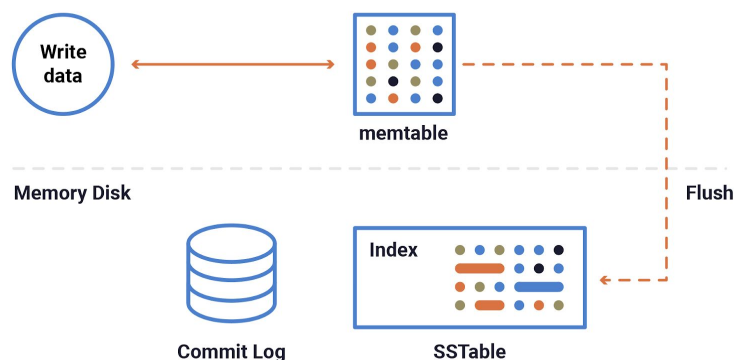


FIGURE 4 The Cassandra write path.

Because of the way Cassandra writes data, many SStables can exist for a single Cassandra table/column family. A process called compaction for a node occurs on a periodic basis that coalesces multiple SStables into one for faster read access.

Reading data from Cassandra involves a number of processes that can include various memory caches and other mechanisms designed to produce fast read response times.

For a read request, Cassandra consults a bloom filter that checks the probability of a table having the needed data. If the probability is good, Cassandra checks a memory cache that contains row keys and either finds the needed key in the cache and fetches the compressed data on disk, or locates the needed key and data on disk and then returns the required result set.

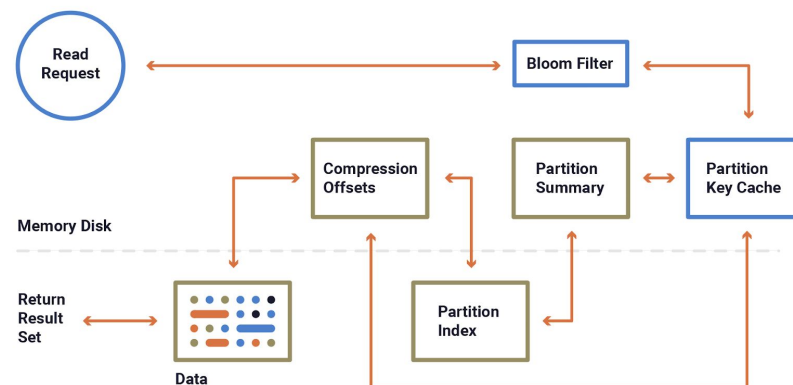


FIGURE 5 The Cassandra read path

Data Distribution and Replication

While the prior section provides a general overview of read and write operations in Cassandra, the actual I/O activity that occurs is somewhat more sophisticated, due to the database's masterless architecture. Two concepts that impact read and write activity are the chosen data distribution and replication models.

Automatic Data Distribution

While an RDBMS and some NoSQL databases necessitate manual and developer-driven methods for distributing data across multiple machines that make up a database (i.e., sharding), Cassandra automatically distributes and maintains data across a cluster so you don't have to.

Cassandra uses a partitioner to determine how data is distributed across the nodes that make up a database cluster. A partitioner is a hashing mechanism that takes a table row's primary key, computes a numerical token for it, and then assigns it to one of the nodes in a cluster.

While Cassandra has multiple partitioners that can be chosen, the default partitioner is one that randomizes data across a cluster and ensures an even distribution of all data.

Cassandra also automatically maintains the balance of data across a cluster even when existing nodes are removed or new nodes are added to a system.

Replication Basics

Unlike many other database management systems, replication in Cassandra is very straightforward and simple to configure and maintain. Most Cassandra users agree that the replication model is one of the features that helps the database stand out from other RDBMS or NoSQL options.

A running Cassandra database cluster can have one or more keyspaces, which are analogous to a Microsoft SQL Server, MySQL, or Oracle pluggable database. It is at the keyspace level that replication is configured, allowing different keyspaces to have different replication models.

Cassandra is able to replicate data to multiple nodes in a cluster, which helps ensure reliability, continuous availability, and fast I/O operations. The total number of data copies that are replicated is referred to as the replication factor (RF). For example, a replication factor of 1 means that there is only one copy of each row in a cluster, whereas a replication factor of 3 means three copies of the data are stored across the cluster.

Once a keyspace and its replication have been created, Cassandra automatically maintains that replication even when nodes are removed, added, or go down and become unavailable for receiving data requests. This equates to no replication babysitting.

Cassandra's replication is both simple to configure and powerful in that it supports a wide range of replication capabilities such as replicating data to different hardware racks (reducing database downtime due to hardware failures) and multiple data centers in different geographic locations on premises as well as the cloud.

Multi-Data Center and Cloud Support

A very popular aspect of Cassandra's replication is its support for multiple cloud regions and availability zones. Many users deploy Cassandra in a multi-region and availability zone manner to ensure constant uptime for their applications and to supply fast read/write data access in localized regions.

You can easily set up replication so that data is replicated across on premises or multi-clouds with users being able to read and write to any data center they choose and the data being automatically synchronized across all centers.

You can also choose how many copies of your data exist in each data center (e.g., two copies in data center 1; three copies in data center 2). Hybrid deployments of part on-premises data centers and part cloud are also supported.

Using Cassandra in Production Environments

As a DBA or architect, you have a responsibility to ensure that the database software you use will work and perform as expected in production environments. To provide that type of guarantee, most NoSQL databases have a commercial software vendor that offers a production-certified version of the database, which oftentimes possess various enterprise features that the open source version does not.

For Cassandra, DataStax provides DataStax Enterprise (DSE) as the commercial software offering. As a DBA or architect, you should be aware that DSE provides the following benefits over the open source version of Cassandra that help you manage, secure, and optimize your database systems:

- A production-certified version of Cassandra that is heavily tested and ready for enterprise environments.
- Multi-model database capabilities with support for the key-value, tabular, JSON / document, and graph data models, all of which inherit the capabilities of Cassandra and additional commercial functionality that follows.
- Advanced security with external security software support, encryption, and data auditing.
- Workload isolation and data replication that ensures OLTP, analytics, and search workloads do not compete with each other for data or compute resources.
- Integrated analytics powered by Apache Spark, including integration with external Hadoop and Spark platforms.
- Integrated enterprise search on stored data.

- Around-the-clock expert support
- Advanced replication that handles data distribution among different clusters in a hub-and-spoke fashion.
- Tiered storage that provides automatic movement of data between different storage media (e.g., SSDs, spinning disks).
- Multi-instance functionality that assists with running multiple instances of the software on single, large servers.
- Automatic management services that transparently automate numerous database maintenance and performance monitoring/management tasks.
- Visual management and monitoring tools that work from any device (laptop, tablet, smart phone).
- In-memory database option for both OLTP and analytic workloads.
- Certified software updates.

NoSQL and Hadoop: A Comparison

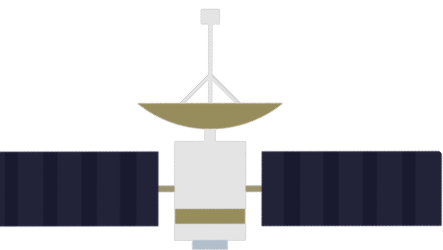
You've no doubt heard about Apache Hadoop™ and perhaps your company is already using it to handle various new data warehousing projects. Are you wondering how Hadoop differs from NoSQL?

Hadoop is an open source software project that enables the distributed processing of large data sets, and uses a scale-out architecture that stores and processes data across many machines. Hadoop is an ecosystem umbrella term that encompasses many different software components.

In general, Hadoop is not a database, but is instead a framework primarily devoted to handling modern data warehousing and analytics “data lake” use cases. Hadoop does offer a NoSQL database as part of its framework (HBase), but it is used mostly for data warehousing situations.

To summarize, the main differences between the two is Hadoop is not a database system but a software ecosystem that allows for massively parallel computing and NoSQL, on the other hand, was created specifically with a database framework in mind. By contrast, a NoSQL database like Cassandra is an operational / transactional database used for cloud applications.

FEATURES	HADOOP	CASSANDRA
Usage	Batch processing data	Real-time processing of data
Architecture	Master-slave	Distributed
Data access	MapReduce	Cassandra Query Language, similar to SQL
Compression	~ 10-15%	~ 80%
Indexing	Challenging	Simple
Storage method	3	Number of nodes in data center
Communication method	Access control through master node	Gossip protocol





This section takes a look at the core Cassandra's data model, what data objects are used for managing data, CQL (Cassandra Query Language), and how transactions are handled in the database.

Data Model Overview

Achieving success with Cassandra almost always comes down to getting two things right:

- The data model
- The selected hardware, especially the storage subsystem

Cassandra is a wide-column/tabular database that uses a query-driven approach designed to quickly capture and query data. There are no concepts of foreign keys, referential integrity, or joins in Cassandra (note that using Spark with Cassandra provides join capability through SparkSQL).

Although Cassandra has objects that resemble an RDBMS (e.g., tables, primary keys, indexes), data should not be modeled in a legacy entity-relationship-attribute fashion as is done with a relational database. Modeling data in Cassandra is done by understanding what questions you will need to ask the database upfront, whereas in an RDBMS, you are likely not used to addressing such things until after all entities, relationships, and attributes are documented.

Unlike an RDBMS that penalizes the use of many columns in a table, Cassandra is highly performant with tables that have hundreds of columns. You may be used to highly normalized, third normal form (3NF) models that you translate into a set of physical tables and their accompanying indexes and such. With Cassandra, you will oftentimes instead have wide-row tables with some data duplication between tables.

Creating your physical objects, however, still looks very much like what you carry out in an RDBMS. For example, a new table defining users for an application might look like the following:

```
CREATE TABLE users (
    username      varchar,
    firstname     varchar,
    lastname      varchar,
    email         list<varchar>,
    password      varchar,
    created_date  timestamp,
    PRIMARY KEY   (username)
);
```

Cassandra Objects

The basic objects you will use in Cassandra include:

- ➔ **Keyspace** - A container for data tables and indexes; analogous to a database in many relational database management systems. It is also the level at which replication is defined.
- ➔ **Table** - Somewhat like an RDBMS table only much more flexible and capable of handling all modern data types.
- ➔ **Primary key** - Used to uniquely identify a row in a table and also distribute a table's rows across multiple nodes in a cluster.
- ➔ **Index** - Similar to an RDBMS index in that it speeds read operations.
- ➔ **User** - A login account used to access data objects.

Cassandra Query Language

Earlier versions of Cassandra solely used an interface called Thrift to create database objects and manipulate data. Today, the CQL has become the primary interface used for interacting with a Cassandra database cluster.

CQL very closely resembles SQL (Structured Query Language) used by all relational database management systems. Because of this similarity, your learning curve will be greatly reduced.

DDL (e.g., CREATE, ALTER, DROP), DML (INSERT, UPDATE, DELETE, TRUNCATE), and query (SELECT) operations are all supported in the manner to which you are accustomed.

CQL data types also reflect RDBMS syntax with numerical (int, bigint, decimal, etc.), character (ascii, varchar, etc.), date (timestamp, etc.), unstructured (blob, etc.), and specialized data types (JSON, etc.) being supported.

To learn more about CQL, visit the [DataStax Documentation website](#).

Transaction Management

While Cassandra does not offer complex/nested transactions in the same way that your legacy RDBMS offers ACID transactions, it does offer the “AID” portion of ACID, in that data written is atomic, isolated, and durable. The “C” of ACID does not apply to Cassandra, as there is no concept of referential integrity or foreign keys.

With respect to data consistency, Cassandra offers tunable data consistency across a database cluster. This means you can decide exactly how strong (e.g., all nodes must respond) or eventual (e.g., just one node responds, with others being updated eventually) you want data consistency to be for a particular transaction, including transactions that are batched together. This tunable data consistency is supported across single or multiple data centers, and you have a number of different consistency options from which to choose.

Moreover, consistency can be handled on a per operation basis, meaning you can decide how strong or eventually consistent it should be per the SELECT, INSERT, UPDATE, and DELETE operations. For example, if you need a particular transaction available on all nodes throughout the world, you can specify that all nodes must respond before a transaction is marked complete. On the other hand, a less critical piece of data (e.g., a social media update) may only need to be propagated eventually, so in that case, the consistency requirement can be greatly relaxed.

Cassandra also supplies “lightweight transactions” (or compare and set). Using and extending the Paxos consensus protocol (which allows a distributed system to agree on proposed data modifications without the need for any one “master” database or two-phase commit), Cassandra offers a way to ensure a transaction isolation level, similar to the serializable level offered by an RDBMS, for situations that need it.

Database Query and Management Tools

As a DBA or architect coming from the RDBMS world, you likely use many command line and visual tools for interacting with the databases you manage. The same kind of tools are available to you with Cassandra.

Various command line utilities are provided for handling administration functions (e.g., the nodetool utility), loading data, and using CQL to create and query database objects (the CQL shell, which is much like Oracle’s SQL*Plus or the MySQL shell).



FIGURE 6 DSE OpsCenter, used for visual database administration.

In addition, graphical tools are provided for running CQL commands against database clusters (e.g., DataStax DevCenter, DataStax Studio) and visually creating/managing/monitoring your clusters (DSE OpsCenter).

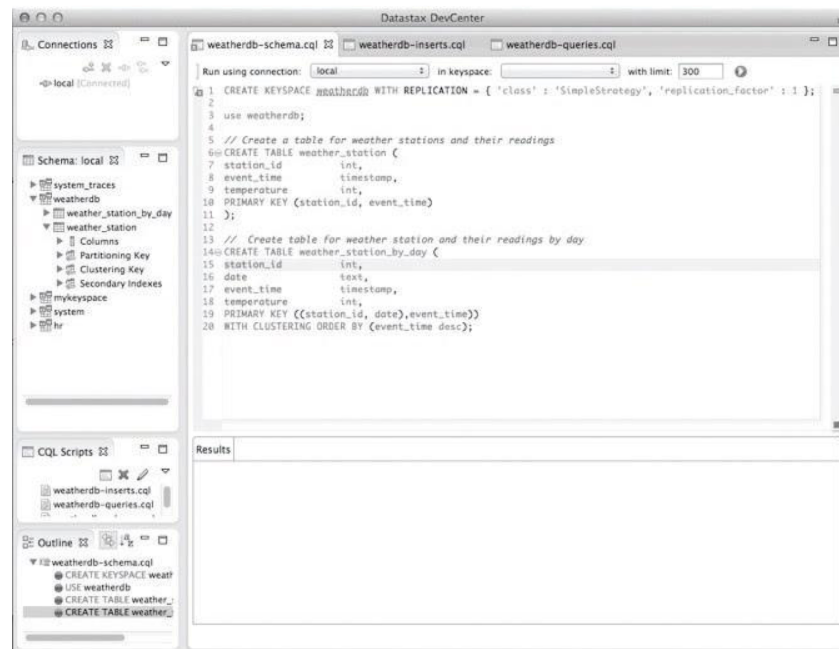
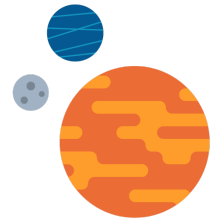


FIGURE 7 DataStax Studio, used for visually querying databases.

Security Management



As a DBA or architect, data security is one of your top priorities, implementing a global security measures to ensure data protection compliance like Personally Identified Information (PII) and Sensitive Personal Information (SPI) is a challenge for any database. One of the myths of NoSQL databases like Cassandra is that they don't offer the security mechanisms needed for enterprise production environments.

In this section, we'll review Cassandra's security capabilities and highlight DSE Advanced Security features, for more information read this [whitepaper](#). In addition, DSE Security Guide provides guidance and a [checklist](#) that fortifies your DSE database against potential harm.

Similar to relational database, NoSQL database security compliances can be broken down into a few main areas: authentication and authorization; encryption for "at rest data" and "data in transit"; auditing for logging and forensics.

Cassandra provides internal-based authentication that allows you to easily create users who can be authenticated to Cassandra database clusters. You'll find the authentication framework extremely familiar - it uses the RDBMS style CREATE/ALTER/DROP USER commands to create/manage with passwords that will then be internally handled by Cassandra. To simplify user provisioning process a default superuser, 'cassandra', is supplied to initially enable the security authentication definition process, and as a best practice it's recommended to disable or remove default account after custom superuser account has been established.

For more advanced user authentication, you can leverage DSE Unified Authentication to manage external, 3rd party security protocols like Kerberos, LDAP, and Active Directory. Authorization and Access Control Management

Cassandra utilize the very familiar GRANT/REVOKE security paradigm - something you should have no problem using. Control over DDL, DML, and SELECT operations are all handled via the granting and revoking of user privileges. Note that a GRANT may be done with or without the GRANT OPTION, which allows the user receiving the grant to provide the same privileges on that object to other users just as how it occurs in the RDBMS world.

The following mechanisms are used to grant resource authorizations.

Role-Based Access Control (RBAC)

One of the major security concerns for DBA and architects is who has access to the data. A smart DBA would take full advantage of Cassandra's powerful and flexible role based access control (RBAC), which can greatly reduce and simplify the security workload.

Under this method, administrators can bundle related privileges together by granting them to roles, which in turn can then be assigned to a specific database user. Once roles are defined and assigned we can grant or revoke permissions on resources.

When combined with auditing capabilities available with DSE, administrators can define specific administrative actions per role, and then log all those actions. As a result, you are able to enforce end-to-end operational control while role-based auditing greatly reduces the audit trail. Read our [blog](#) to learn how to implement Role Based Access Control.

RLAC (Row Level Access Control)

RLAC provides authorization to data within a table by matching a filter applied to a text-based partition key. It provides fine-grain user access control down to the row level so that only authorized users are able to view or modify subsets of data. RLAC support comes in handy when you have multi-tenant SaaS applications in a single DSE platform.

Proxy Auth

DSE Proxy Management allows roles to log in and execute CQL queries as other roles. This is particularly useful for a secure middleware like web servers; the web server can log in once and proxy execute queries as its clients, keeping the audit log intact and leveraging DSE role-based access control.

Separation of duties

While roles can define who has access to the data, regulatory mandates require enterprises to have segregation of duties to insure that no individual person, role or group have the ability to execute all parts of a transaction or process. DSE helps with the implementation of separation of duties by providing separate paths for administrative access and data access. This separation gives administrators full control of the database administration capabilities but restricts their access to view the data itself.

Encryption In-Flight

This refers to the encryption of data as it moves over a network between nodes. In a distributed environment like DSE, network traffic is constant. First, Cassandra includes an optional encrypted 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.

Next, node-to-node encryption can be used as well to ensure data is protected as it is transferred between nodes in a database cluster.

Encryption At-Rest

This refers to the data that is stored on persistent storage such as disk drives in an encrypted format. Encryption at-rest protects against data exposure in the event of the physical theft of a device or an environment where storage resources might be re-used such as public clouds. DSE Transparent Data Encryption (TDE) protects data at rest from being stolen or used in an unauthorized manner. You can encrypt tables using a local encryption key file or a remotely stored and managed Key Management Interoperability Protocol (KMIP) encryption key. An important note is TDE encrypts only specific sensitive files as opposed to entire filesystems. This increases the security of files that are moved around and ensure that they aren't improperly decrypted.

DSE TDE provides encryption for:

- Entire tables (except partition keys, which are always stored in plain text)
- SSTables containing data, including system tables
- Search indexes
- File-based hints
- Commit logs
- Sensitive properties in `dse.yaml` and `cassandra.yaml`

Encryption is transparent to all end user activities; data may be read, inserted, updated, etc, with nothing having to change on the application end. Click [here](#) for more information on Transparent data encryption (TDE).

Data Auditing

Database audit log is a standard tool for enterprises to capture critical information on data access and database changes, this information can be reviewed later to ensure compliance with regulatory, security and operational policies. Auditing database activity ensures that companies meet regulatory compliance laws such as SOX, HIPAA, PCI and GDPR et al. Security compliance means companies want to know what user activities took place on a particular node or entire cluster. Data auditing allows for a "who looked at what/when, who changed what/when" type of documentation that many large-scale enterprises need to have in order to comply with various internal or external security policies.

The granularity of activities that can be audited include:

- All activity (DDL, DML, queries, errors)
- DML only
- DDL only
- Security changes (e.g., assigning/revoking privileges, dropping users)
- Queries only
- Errors only (e.g., login failures)

Read this to learn more about [auditing database activity](#).

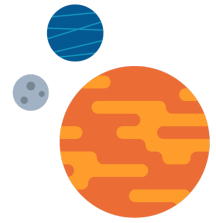
Traditionally in DSE auditing was controlled by database objects you want to monitor, this can provide a lot of information on an object but not a person's activity. Security administrators also want the ability to filter out the noise and only focus on role instead of just a database object. DSE has now enhanced auditing to audit changes and user activity by role. This would filter out audit logs generated by machines and let Security administrators focus on roles such as DBA access vs. machine access.

Private Schemas

Similar to RDBMS DBA and architects need to ensure that only authorized users have access to certain database objects. This gives administrators more control over schema visibility which would limit who can see what parts of a table definition, assisting in secure multi-tenant applications.



Managing Availability and Multiple Data Centers



Another key aspect of your job as a DBA or architect is to ensure the databases you manage are always available for the applications that use them. One thing you will like about Cassandra is that ensuring constant uptime is easy. There is no need for specialized, add-on log shipping software such as Oracle DataGuard.

Further, distributing data to multiple geographies and across various cloud providers is simple and straightforward with Cassandra.

How to Ensure Constant Uptime

As previously discussed, Cassandra sports a masterless architecture where all nodes are equal; and it has been built from the ground up with the understanding that outages and hardware failures will occur. To overcome those and similar issues, Cassandra delivers redundancy in both data and function to a database cluster with all nodes being the same.

Where data operations are concerned, any node in a cluster may be the target for both reads and writes. Should a particular node go down, there is no hiccup in the cluster at all, as any other node may be written to, with reads served from other nodes holding copies of the downed node's data.

To ensure constant access to data, you should configure Cassandra's replication to keep multiple copies of data on the nodes that comprise a database cluster. The number of data copies is completely up to you, with three being the most commonly used in production Cassandra environments.

Should a node go down, new or updated information is simply written to another node that keeps a copy of that data. When the downed node is brought back online, it automatically syncs with other nodes holding its data so that it is brought back up to date in a transparent fashion.

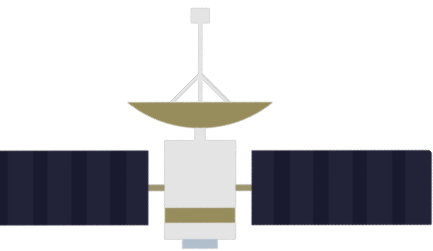
Multi-Data Center and Cloud Options

Cassandra is the leading distributed database for multi-data center and cloud support. Many production Cassandra systems consist of a database cluster that spans across multiple physical data centers, cloud regions and availability zones, or a combination of both. Should a large outage occur in a particular geographical region, the database cluster continues to operate as normal with the other data centers assuming the operations previously directed at the now downed data center or cloud zone. Once the downed data center comes back online, it syncs with the other data centers and makes itself current.

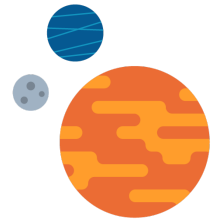


FIGURE 8 A single DSE cluster can span multiple data centers and the cloud.

An additional benefit of having a single cluster that spans multiple data centers and geographies is that data can be read and written to incredibly quickly in each location, thus keeping performance very high for the customers it serves in those locations.



Analyzing and Searching Data



Many applications have requirements that their underlying transactional database easily service analytic and search operations.

As a DBA or architect, you are likely familiar with analytic capabilities that can be run via SQL and full-text search options in an RDBMS, and might wonder how the same things are handled in Cassandra.

DataStax Enterprise (DSE) Analytics, built on a production-certified version of Apache Spark™, with enhanced capabilities like AlwaysOn SQL and highly available Spark resource manager, enables enterprises to build real-time, contextual applications to make highly relevant, in-the-moment decisions.

Be it to process streaming and historical data or to improve your ongoing BI reporting, DSE Analytics gives you the flexibility to transform all your meaningful data into action — at cloud scale.

Real Time and Batch Analytics

Because Cassandra has a distributed, shared-nothing architecture, the framework for running analytics on it compared to a centralized RDBMS will be different.

There are two options in DataStax Enterprise that allow you to run analytic operations easily on Cassandra data. You can run both real-time and batch (i.e., longer running) analytics on data via the platform's built-in components that utilize Apache Spark for analytics work.

The analytics capability in the platform provides you with a number of the SQL functions and abilities that you are used to in the RDBMS world (e.g., joins, aggregate functions). In addition, analytics can be run across multiple data centers and cloud availability zones. Built-in continuous availability options are also included.

External Hadoop and Spark Support

You also have the ability to connect the data in DataStax Enterprise to an external Hadoop and/or Spark cluster and run analytic queries on data that combines both the operational data in Cassandra with historical data stored in a Hadoop deployment such as Cloudera or Hortonworks (e.g., a single query can join a Cassandra table with a Hadoop object). If you have used RDBMS connection options such as Oracle's database links or Microsoft SQL Server's linked servers to integrate external database systems, the concept is somewhat similar.

Searching Data

Some architects still shard their systems and use something like Cassandra for operational data management and a separate system and set of software for search operations as a DBA, you'd likely prefer to have everything under one roof.

DataStax Enterprise supplies DSE Search, which uses Apache Solr™ as its foundation to manage search tasks. With DSE Search, you don't have to shard your application and you have the typical search bases covered including full-text search, hit highlighting, faceted search, rich document (e.g., PDF, Microsoft Word) handling and geospatial search.

Search operations can scale out across multiple nodes so you can add more nodes dedicated to search tasks when the need arises. Multi-data center and cloud support is built in, as is redundancy for continuous availability.

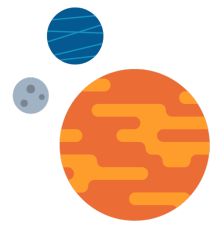
Workload Management for Analytics and Search

When enabling analytics and search on a database cluster, you have a number of configuration options available. If you choose, you can run transactional (OLTP), analytics and search operations on all nodes in a database cluster.

Another deployment methodology includes separating OLTP, analytics, and search workloads so that each runs on its own series of nodes. This strategy ensures that differing workloads do not compete with each other for either compute or data resources. Replication can be set up between all nodes so that data is transparently replicated to each set of nodes without manual intervention.

This translates into you not having to worry about complex ETL jobs that transfer data between different systems, as you might be used to doing with an RDBMS.

This also holds true if you are running graph database operations in a cluster – operational, analytical and search tasks can either be combined or separated across different nodes.



Backup and Recovery

One of your key responsibilities as a DBA is to ensure that proper backup and recovery procedures are in place should a database become corrupted or a large data loss occurs. This section describes how backup and recovery processes work on a NoSQL database like Cassandra.

Using Replication and Multi-Data Center for Backup and Recovery

Some administrators simply use Cassandra's built-in replication and multi-data center capabilities for backup. Because the functionality is native to Cassandra, there is no need for add-on software (e.g., Oracle DataGuard). Since replication is so easy to use, some DBA's just create one or more physical or virtual data centers for a cluster and utilize them for disaster recovery purposes.

While such a strategy can be satisfactory for some situations, it is important to note that it will not protect you in cases where large amounts of data are deleted, tables are dropped, and other similar unintended actions are carried out - such activities will be replicated and applied to the other data centers.

Cassandra Backup Options

Cassandra allows you easily backup all keyspaces in a cluster, certain selected keyspaces, or only desired tables in a keyspace. A backup is called a snapshot in DSE.

Snapshot-based Backups

Cassandra provides node-level snapshot via the `nodetool` command. Snapshots make a copy of all or part of the keyspaces and tables in a node and save it into a separate file. When you take a snapshot, Cassandra flushes data in the memtables into the disk (SStable) and makes a hard link to each SStable file. Each snapshot contains a `manifest.json` file that list the SStable files included in the snapshot.

You can take snapshots of your cluster via either a command line utility or visually through DataStax OpsCenter. While you can script your own backups via command line utilities, OpsCenter provides an easy way to design and schedule your backups.

Note that you can also customize backups in OpsCenter by writing and including scripts that run both before and after a backup.

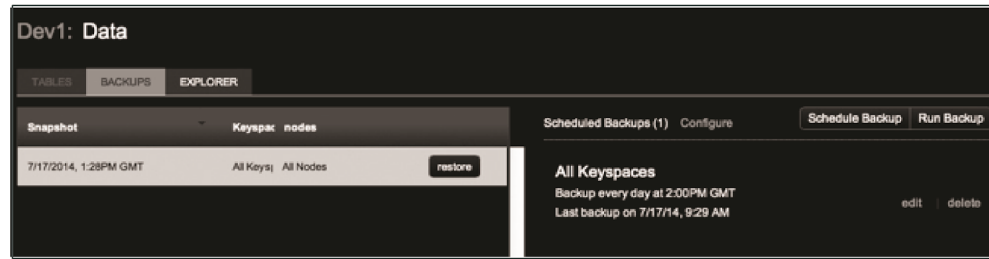


FIGURE 9 DataStax OpsCenter's backup interface.

Incremental Backups

Incremental (only new or changed data versus full) backups are also supported for Cassandra, although it's not exactly the same as with an RDBMS. An incremental backup will backup only changed SSTables vs change rows as in an RDBMS. In Cassandra, incremental backups contain only new SSTables files, making them dependent on the previously created snapshot.

Incremental Backups in Combination with Snapshot

You can combine both methods to create a better granularity of the backups. Data is backed up periodically via the snapshot and incremental backup files are used to obtain granularity between scheduled snapshots.

Commit Log Backup in Combination with Snapshot

This method is similar to incremental backup. Instead of backing up the newly added SSTable, commit logs are archived. Similar to the previous methods, snapshots provide the bulk of backup data, while archive of the commit log is used for point-in-time backup.

Restoring Data

Database recovery operations can be carried out with either command line utilities or visually through DataStax OpsCenter. Restores can be full, utilize incremental backups, and also be object-level if needed (e.g., you can only restore one backed up table versus all tables).

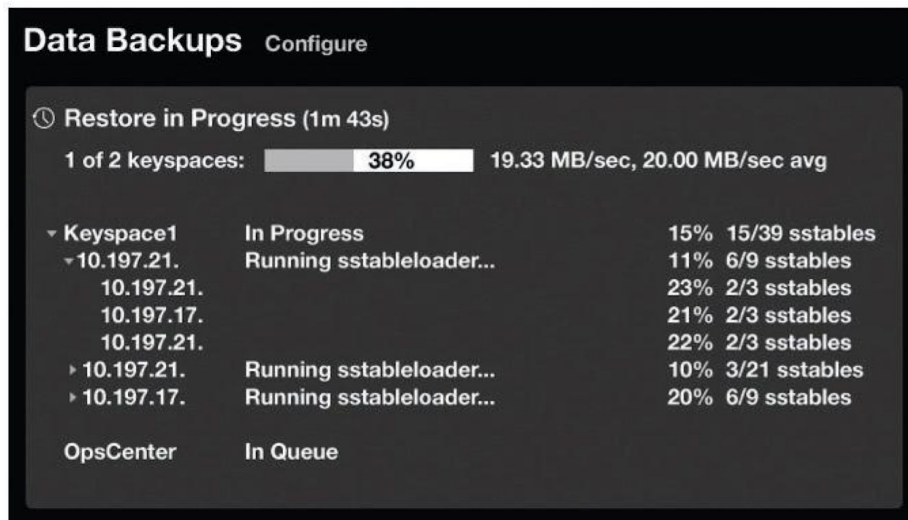
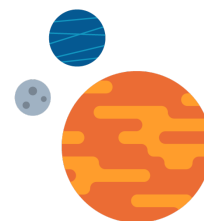


FIGURE 10 Restoring a keyspace with OpsCenter.

OpsCenter simplifies restore operations and handles restore tasks on all affected nodes.



Performance Management



Monitoring, troubleshooting, and tuning databases are a top priority for you as a DBA. This section details how you can carry out your performance management tasks on a NoSQL database like Cassandra.

Monitoring Basics

There are a number of command line utilities that enable you to get a status of your database clusters as well as general metrics for the network, objects and I/O operations both at a high level and low level (e.g., table) fashion. For example, the Cassandra `nodetool` utility lets you quickly determine the up/down status and current data distribution of a cluster:

```

root@robin-1: ~/ctest/dsc-cassandra-1.0.5/bin — ssh — 124x12
INFO 15:10:12,246 Node robin-1/50.57.220.16 state jump to normal
INFO 15:10:12,248 Bootstrap/Replace/Move completed! Now serving reads.
INFO 15:10:12,249 Will not load MX4J, mx4j-tools.jar is not in the classpath

root@robin-1:~/ctest/dsc-cassandra-1.0.5/bin# ./nodetool -h localhost ring
Address      DC           Rack         Status State   Load      Owns      Token
-----
datacenter1 rack1       Up           Normal  11.00 KB  25.00%    127605887595351923798765477786913079296
datacenter1 rack1       Up           Normal  6.71 KB   25.00%    42535295865117307932921825928971026432
datacenter1 rack1       Up           Normal  11.00 KB  25.00%    85070591730234615865843651857942052864
datacenter1 rack1       Up           Normal  9.05 KB   25.00%    127605887595351923798765477786913079296
root@robin-1:~/ctest/dsc-cassandra-1.0.5/bin#

```

FIGURE 11 Checking a cluster's status with the `nodetool` utility.

Advanced Command Line Performance Monitoring Tools

From a performance metrics standpoint, Cassandra delivers many different statistics that can be accessed in various ways. If you are coming from an RDBMS like Oracle or Microsoft SQL Server and are used to performance data dictionaries like Oracle's V\$ views or SQL Server's dynamic management tables, the most familiar interface for you is the one supplied by DataStax Enterprise's Performance Service.

The Performance Service collects, organizes, and maintains an in-depth diagnostic data dictionary for each cluster. It consists of various tables that can be accessed via any CQL utility (e.g., the CQL shell utility, DataStax DevCenter, and DataStax Studio) and gives you both high-level and detailed performance views of how well a cluster is running.

The Performance Service maintains the following levels of performance information:

- ➔ **System level** - Supplies general memory, network and thread pool statistics.
- ➔ **Cluster level** - Provides metrics at the cluster, data center and node level.
- ➔ **Database level** - Provides drill down metrics at the keyspace, table and table-per-node level.
- ➔ **Table histogram level** - Delivers histogram metrics for tables being accessed.
- ➔ **Object I/O level** - Supplies metrics concerning 'hot objects' and data on what objects are being accessed the most.
- ➔ **User level** - Provides metrics concerning user activity, top users (those consuming the most resources on the cluster) and more.
- ➔ **Statement level** - Captures queries that exceed a certain response time threshold along with all their relevant metrics.
- ➔ **Slow queries** - Identify and tune poorly performing queries

You can configure the service to collect nothing, all, or selected performance metrics for the above categories. Once the service has been configured and is running, statistics are populated in their associated tables and stored in a special keyspace (`dse_perf`). You can then query the various performance tables to get statistics such as the I/O metrics for certain objects:

```
cqlsh:dse_perf> use dse_perf;  
cqlsh:dse_perf> expand on;  
cqlsh:dse_perf> select * from object_io;
```

@ Row 1

node_ip	127.0.0.1
keyspace_name	weatherdb
table_name	weather_station
last_activity	2014-06-24 08:24:11-0400
memory_only	False
read_latency	104
total_reads	6
total_writes	33
write_latency	51

```
cqlsh:dse_perf> expand on;
cqlsh:dse_perf> select * from object_io;
```

@ Row 2

node_ip	127.0.0.1
keyspace_name	weatherdb
table_name	weather_station_by_day
last_activity	2014-06-24 08:24:11-0400
memory_only	False
read_latency	0
total_reads	0
total_writes	30
write_latency	45

Visual Database Monitoring

In addition to monitoring your database clusters from the command line, you can also easily check the health of all clusters you're managing visually by using DataStax OpsCenter. OpsCenter gives you both global, at-a-glance dashboards that help you understand how all clusters under your control are doing, as well as drill down capabilities into each cluster and its individual nodes.

A global dashboard helps you understand how well all clusters are running and if there are any alerts or issues for one or more clusters that need your attention:

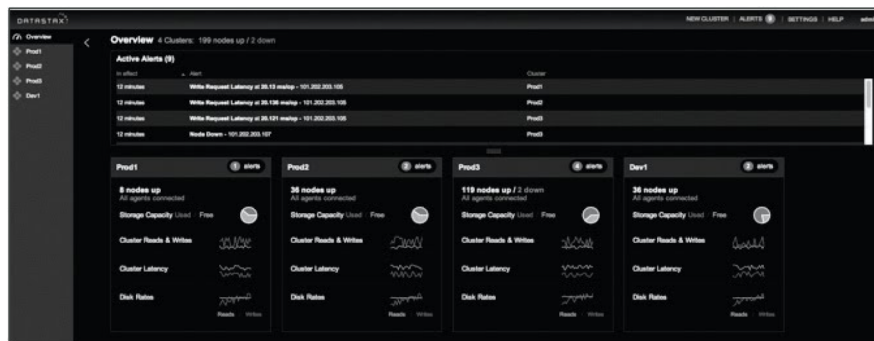


FIGURE 12 Checking OpsCenter's global cluster dashboard.

From the global dashboard, you can drill down into each individual cluster and create customized monitoring dashboards for the performance metrics you care about the most:



FIGURE 13 Examining performance metrics for a single database center.

You can also create proactive alerts that notify you far in advance of a problem actually occurring in one of your clusters:

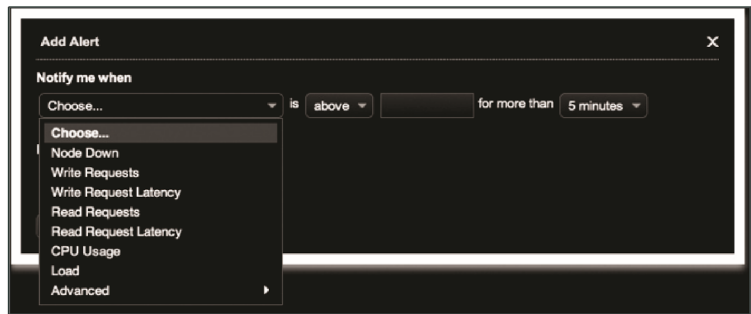


FIGURE 14 Creating an alert in OpsCenter

In addition, you can utilize built-in expert services like the Best Practice service that will scan your clusters and provide expert advice on how to configure and tune things for better uptime and performance:

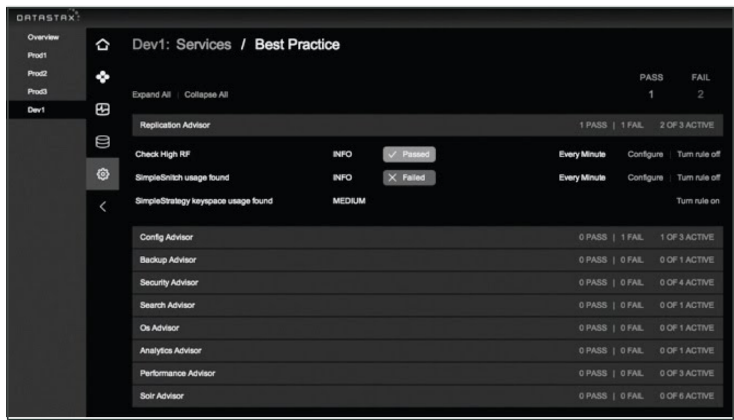


FIGURE 15 OpsCenter's Best Practice service.

These and other capabilities in OpsCenter help monitor and tune database clusters via any Web browser (laptop, tablet, smart phone) no matter if they are in your own data center or are running on one of the cloud providers.

Finding and Troubleshooting Problem Queries

As a DBA or architect, you're sometimes called upon to locate a database's worst running queries that slow the performance of the system as a whole. You'll find this isn't hard to do with Cassandra.

First, you can use the DataStax Enterprise Performance Service to automatically capture long-running queries (based on response time thresholds you specify) and then query a performance table that holds those statements:

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

@ Row 1

node_ip	127.0.0.1
date	2014-06-24 00:00:00-0400
start_time	c45fa7c0-fb9c-11e3-b5b2-e76149b7842c
commands	select *from weather_station where station_id=2 and date="2014-01-01";
duration	183
parameters	null
source_ip	127.0.0.1
table_names	weather_station
username	anonymous

In addition, there is a background query tracing utility available that you can use on an ad-hoc basis. You can choose to trace all statements coming into a database cluster or only a percentage of them, and then look at the results. The trace information is stored in the `systems_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, and much more.

You can also use the tracing utility much in the same way you do an EXPLAIN PLAN on an RDBMS query. For example, to understand how a Cassandra cluster will satisfy a single CQL INSERT statement, you would enable the trace utility from the CQL command shell, issue your query, and review the diagnostic information provided:

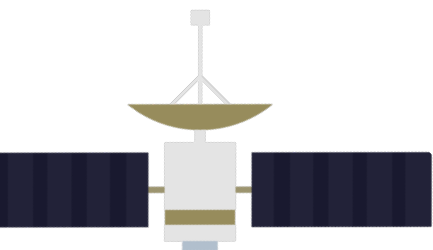
```
cqlsh>tracing on;
Now tracing requests.
cqlsh:foo>INSERT INTO test (a, b) VALUES (1, 'example');
Tracing session 4ad36250-1eb4-11e2-0000-fe8ebeced9f9
```

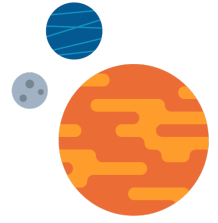
Activity	Time Stamp	Source	Source_Elapsed
execute_cqL3_query	00:2:37, 015	127.0.01	0
execute_cqL3_query	00:2:37, 015	127.0.01	0
Parsing statement	00:2:37, 015	127.0.01	81
Preparing statement	00:2:37, 015	127.0.01	273
Determining replicas for mutation	00:2:37, 015	127.0.01	540
Sending message to /127.0.0.2	00:2:37, 015	127.0.01	779
Message received from /127.0.0.1	00:2:37, 016	127.0.02	63
Applying mutation	00:2:37, 016	127.0.02	220
Acquiring switchLock	00:2:37, 016	127.0.02	250
Appending to commitlog	00:2:37, 016	127.0.02	577
Adding to memtable	00:2:37, 016	127.0.02	378
Enqueuing response to /127.0.0.1	00:2:37, 016	127.0.02	710
Sending message to /127.0.0.1	00:2:37, 016	127.0.02	888
Message received from /127.0.0.2	00:2:37, 017	127.0.01	2334
Processing response from /127.0.0.2	00:2:37, 017	127.0.01	2550
Request complete	00:2:37, 017	127.0.01	2581

With Cassandra's tracing capabilities, OpsCenter's visual monitoring, DataStax Enterprise's Performance service and general command line monitoring tools, you will have most, if not all, of the typical performance tools at your disposal with Cassandra as you do today with your favorite RDBMS.

NoSQL Performance Benchmarks

When picking the NoSQL database it is critical to understand your use case and finding the right fit for your use case. Instead of conducting a formal proof of concept (PoC), we have provided the following [benchmark test](#) to provide a graphical, 'at a glance' view of how Cassandra stacks up against MongoDB, HBase and Couchbase. You can also find three [additional benchmarks](#); taking a look at write/read performance and performance at a scale.





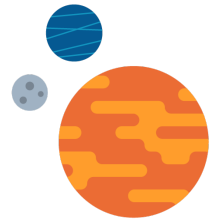
Moving data from an RDBMS or other database to Cassandra is generally quite easy.

The following options exist for migrating data to Cassandra:

- ➔ **COPY command** - CQL provides a copy command (very similar to Postgres) that is able to load data from an operating system file into a Cassandra table.
- ➔ **DataStax Bulk loader (dsbulk)** - This utility is designed for more quickly loading a DSE table with a file that is delimited in some way (e.g., comma, tab, etc.) Note: there is a separate bulk loader available for DSE Graph. For more information, take a moment to read our blog post on [Introducing DataStax Bulk Loader](#).
- ➔ **ETL tools** - There are a variety of ETL tools (e.g., Talend, Informatica and Streamsets) that support Cassandra as both a source and target data platform. Many of these tools not only extract and load data but also provide transformation routines that can manipulate the incoming data in many ways. A number of these tools are also free to use (e.g., Pentaho, Jaspersoft, Talend).
- ➔ **sstableloader** - Provides the ability to bulk load external data into a cluster.
- ➔ **DSE Analytics** – Can use Apache Spark to connect to a wide variety of data sources and save the data to DSE using either the older RDD or newer DataFrame method.
- ➔ **DataStax Apache Kafka Connector** – Synchronizes records from a Kafka topic with rows in one or more DSE database tables.



Strategies for Implementing NoSQL



This section provides basic checklists to use when evaluating a NoSQL database for production environments, guidelines for deciding when NoSQL should be deployed versus an RDBMS and what deployment scenarios are most common.

Evaluating NoSQL for Your Enterprise

Although not exhaustive, below are technical and business considerations designed to ask the right questions when evaluating whether a particular NoSQL database is suited for your production environment.

Technical Considerations

- Can the NoSQL database serve as the primary data source for the intended online application?
- How safe is the NoSQL database where the possibility of losing critical data is concerned? Are writes durable in nature by default so that data is safe?
- Is the NoSQL database fault tolerant (i.e., has no single point of failure) and is it capable of providing not just high availability, but continuous availability?
- Can the NoSQL database easily replicate data between the same and multiple data centers, as well as different cloud regions and availability zones?
- Does the NoSQL database offer read/write anywhere capabilities (i.e. can any node in the cluster be written to and read from)?
- Does the NoSQL database provide a robust security feature set?
- Does it support easy-to-create and manage backup and recovery procedures?
- Does the NoSQL database require or remove the need for special caching layers?
- Is the NoSQL database capable of dealing with data velocity, data variety and data complexity issues?
- Does the NoSQL database offer linear scalability where adding new nodes is concerned?
- Can new nodes be added and removed online (i.e., without business impact)?

- Does the NoSQL database support key platforms/developer languages?
- Does the NoSQL database provide an SQL-like query language?
- Can the NoSQL database run on commodity hardware with no special hardware requirements?
- Is the NoSQL database easy to implement and maintain for large deployments?
- Does the NoSQL database provide data compression that supplies real storage savings?
- Can analytic operations be run easily on the NoSQL database?
- Does the NoSQL database have automatic workload and data balancing?
- Can search operations and functions be easily and directly carried out on the NoSQL database?
- Can the NoSQL database provide workload isolation between online, analytic, and search operations in a single application?
- Does the database have solid command-line and visual tools for development, administration, and performance management?

Business Requirements

- Is the NoSQL solution backed by a commercial entity?
- Does the commercial entity provide enterprise 24x7 support and services?
- Does the NoSQL solution have professional training resources and online documentation?
- Does the NoSQL solution have referenceable customers across a wide range of industries that use the product in critical production environments?
- Does the NoSQL database have an attractive cost/pricing structure?
- If open source, does the NoSQL database have a thriving open source community?

Practical Guidelines for Selecting NoSQL vs. an RDBMS

How do you determine whether a NoSQL database like Cassandra or DSE Graph should be used for all or part of an application versus an RDBMS? Some basic questions to ask include:

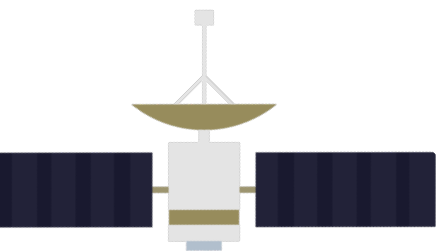
- Do you need a more flexible data model to manage data that goes beyond the RDBMS table/row data structure and instead includes a combination of structured, semi-structured and unstructured data?
- Do you find that complex JOIN operations are overwhelming your RDBMS and response times are slow because of them?
- Do you care more about the value derived from the relationships that form between the tables vs. the tables themselves?
- Do you need continuous availability with redundancy in both data and function across one or more locations versus simple failover for the database?

- Do you need a database that runs over multiple data centers / cloud availability zones?
- Do you need to handle high velocity data coming in via sensors, mobile devices, and the like, and have extreme write speed and low latency query speed?
- Do you need to go beyond single machine limits for scale-up and instead go to a scale-out architecture to support the easy addition of more processing power and storage capacity?
- Do you need to run different workloads (e.g. online, analytics, search) on the same data without needing to manually ETL the data to separate systems/machines?
- Do you need to manage a widely distributed system with minimal staff?

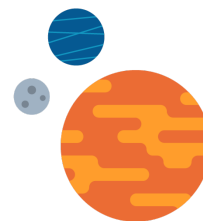
Deployment Considerations

From a practical perspective, as a DBA or an architect, how do you go about actually moving to NoSQL and implementing your first application? In general, there are three ways to deploy a NoSQL database like Cassandra:

1. **New applications:** Many begin with NoSQL by choosing a new application and starting from the ground up. Such an approach mitigates the issues of application rewrites, data migrations, etc.
2. **Augmentation:** Some choose to augment an existing system by adding a NoSQL component to it. This oftentimes happens with applications that have outgrown an RDBMS due to scale problems, the need for better availability, or other issues. Parts of the existing system continue to use the existing RDBMS whereas other components of the application are modified to utilize the NoSQL database.
3. **Full Rip-Replace:** For systems that simply are proving too costly from an RDBMS perspective to keep, or are breaking in major ways due to increases of user concurrency, data velocity, or data volume from cloud applications, a full replacement is done with a NoSQL database.



Conclusion



This guide has been designed to provide you with a preliminary understanding from a DBA or architect perspective on the basics of NoSQL, and how a NoSQL database like Apache Cassandra differs from an RDBMS like Oracle, SQL Server, and MySQL.

It has been written to supply you with an overview of how you will go about designing, managing, deploying, and monitoring Cassandra or multi-model database systems.

To find out more about Apache Cassandra and DataStax, and to obtain downloads of Apache Cassandra and DataStax Enterprise software, visit www.datastax.com. DataStax Enterprise is completely free to use in non-production environments, while production deployments require a software subscription be purchased.

© 2020 DataStax, All Rights Reserved. DataStax, Titan, and TitanDB are registered trademarks of DataStax, Inc. and its subsidiaries in the United States and/or other countries.

Apache, Apache Cassandra, and Cassandra are either registered trademarks or trademarks of the Apache Software Foundation or its subsidiaries in Canada, the United States, and/or other countries.