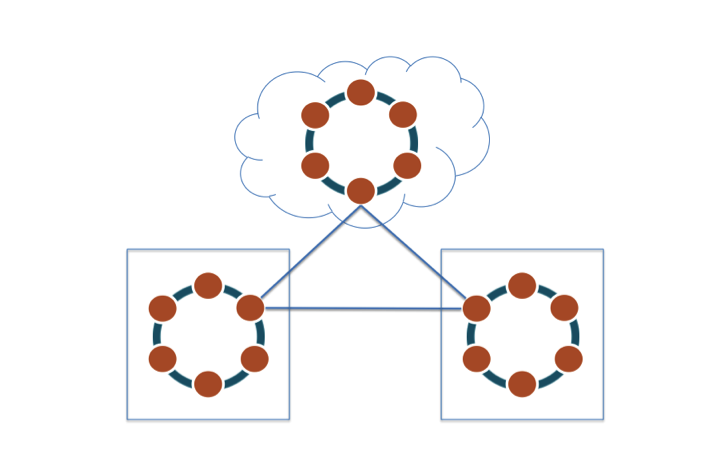
A brief introduction to the features and architecture of Apache Cassandra.



Introduction

The article provides an overview of non-relational databases and Apache Cassandra™, and describes a number of top-level Cassandra concepts that will help you understand how it operates and manages data. Subsequent ebooks in this series will expand on the material provided here, providing you with all the information necessary to build successful online applications that use Cassandra as the underlying data store. If you are new to Cassandra, then what follows should greatly help you understand what it’s all about.

What is NoSQL?

Many of today’s online applications have database requirements that exceed the capabilities of legacy relational databases. The need for very very low latency, heretofore unknown levels of scale, continuous uptime, global distribution of data, the ability both write and read data anywhere, and reducing both software and operational costs, all have given birth to the non-relational database category.

Non-relational databases use new and potentially unfamiliar architectures and data models. To satisfy the requirements of modern online applications, these new databases must make trade-offs as dictated by the [CAP theorem](https://en.wikipedia.org/wiki/CAP_theorem). These trade-offs have far-reaching consequences, among them a sacrifice of traditional relational database features like automatic table joins and ACID transactions. In the majority of cases, these trade-offs are well worth it.

How Does Cassandra Differ From a Relational Database?

Although the non-relational databases in the market today provide different features and benefits, a database like Cassandra differs from a typical relational database in the following ways:

Table 1. Table A quick comparison of RDBMS and a NoSQL database like Cassandra.

| **Relational Database** | **Cassandra** |
| --- | --- |
| Handles moderate incoming data velocity | Handles high incoming data velocity |
| Data arriving from one/few locations | Data arriving from many locations |
| Manages primarily structured data | Manages all types of data |
| Supports complex/nested transactions | Supports simple transactions |
| Single points of failure with failover | No single points of failure; constant uptime |
| Supports moderate data volumes | Supports very high data volumes |
| Centralized deployments | Decentralized deployments |
| Data written in mostly one location | Data written in many locations |
| Supports read scalability (with consistency sacrifices) | Supports read and write scalability |
| Deployed in vertical scale up fashion | Deployed in horizontal scale out fashion |

What is Apache Cassandra?

Apache Cassandra is a massively scalable open source non-relational database that offers 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, was open sourced in 2008, and became a top-level Apache project in 2010.

[START COURSE: INSTALL AND CONFIGURE APACHE CASSANDRA](https://academy.datastax.com/courses/installing-and-configuring-cassandra)

Key Cassandra Features and Benefits

Cassandra provides a number of key features and benefits for those looking to use it as the underlying database for modern online applications:

* Massively scalable architecture – a masterless design where all nodes are the same, which provides operational simplicity and easy scale-out.
* Active everywhere design – all nodes may be written to and read from.
* Linear scale performance – the ability to add nodes without going down produces predictable increases in performance.
* Continuous availability – offers redundancy of both data and node function, which eliminate single points of failure and provide constant uptime.
* Transparent fault detection and recovery – nodes that fail can easily be restored or replaced.
* Flexible and dynamic data model – supports modern data types with fast writes and reads.
* Strong data protection – a commit log design ensures no data loss and built in security with backup/restore keeps data protected and safe.
* Tunable data consistency – support for strong or eventual data consistency across a widely distributed cluster.
* Multi-data center replication – cross data center (in multiple geographies) and multi-cloud availability zone support for writes/reads.
* Data compression – data compressed up to 80% without performance overhead.
* CQL (Cassandra Query Language) – a SQL-like language that makes moving from a relational database very easy.

Top Use Cases

While Cassandra is a general purpose non-relational database that can be used for a variety of different applications, there are a number of use cases where the database excels over most any other option. These include:

* Internet of things applications – Cassandra is perfect for consuming lots of fast incoming data from devices, sensors and similar mechanisms that exist in many different locations.
* Product catalogs and retail apps – Cassandra is the database of choice for many retailers that need durable shopping cart protection, fast product catalog input and lookups, and similar retail app support.
* User activity tracking and monitoring – many media and entertainment companies use Cassandra to track and monitor the activity of their users’ interactions with their movies, music, website and online applications.
* Messaging – Cassandra serves as the database backbone for numerous mobile phone and messaging providers’ applications.
* Social media analytics and recommendation engines – many online companies, websites, and social media providers use Cassandra to ingest, analyze, and provide analysis and recommendations to their customers.
* Other time-series-based applications – because of Cassandra’s fast write capabilities, wide-row design, and ability to read only the columns needed to satisfy queries, it is well suited time series based applications.

Architecture Overview

Cassandra’s architecture is responsible for its ability to scale, perform, and offer continuous uptime. Rather than using a legacy master-slave or a manual and difficult-to-maintain sharded design, Cassandra has a masterless “ring” architecture that is elegant, easy to set up, and easy to maintain.

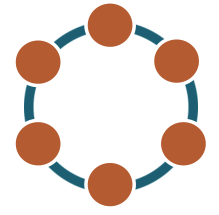


Figure 1. Cassandra surports a masterless "ring" architecture

In Cassandra, all nodes play an identical role; there is no concept of a master node, with all nodes communicating with each other via a distributed, scalable protocol called "gossip."

Cassandra’s built-for-scale architecture means that it is capable of handling large amounts of data and thousands of concurrent users or operations per second—​even 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 to an existing cluster without having to take it down first.

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

Writing and Reading Data

Cassandra is well known for its impressive performance in both reading and writing data.

Data is written to Cassandra in a way that provides both full data durability and high performance. Data written to a Cassandra node is first recorded in an on-disk 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 written to an immutable file on disk called an *SSTable*. Buffering writes in memory in this way allows writes always to be a fully sequential operation, with many megabytes of disk I/O happening at the same time, rather than one at a time over a long period. This architecture gives Cassandra its legendary write performance.

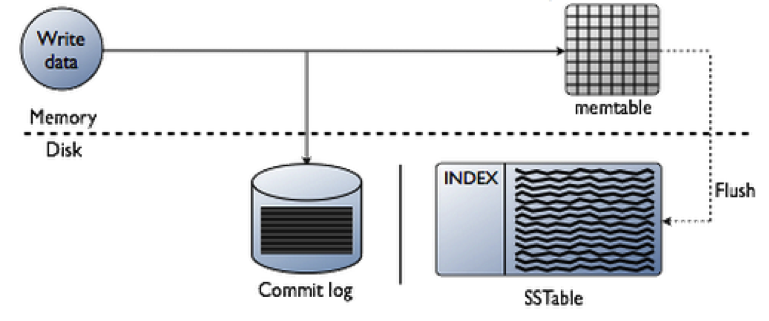


Figure 2. The Cassandra Write Path

Because of the way Cassandra writes data, many SSTables can exist for a single Cassandra logical data table. A process called *compaction* occurs on a periodic basis, coalescesing 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 an in-memory data structure called a [*Bloom filter*](http://en.wikipedia.org/wiki/Bloom_filter) that checks the probability of an SSTable having the needed data. The Bloom filter can tell very quickly whether the file probably has the needed data, or certainly does not have it. If answer is a tentative yes, Cassandra consults another layer of in-memory caches, then fetches the compressed data on disk. If the answer is no, Cassandra doesn’t trouble with reading that SSTable at all, and moves on to the next.

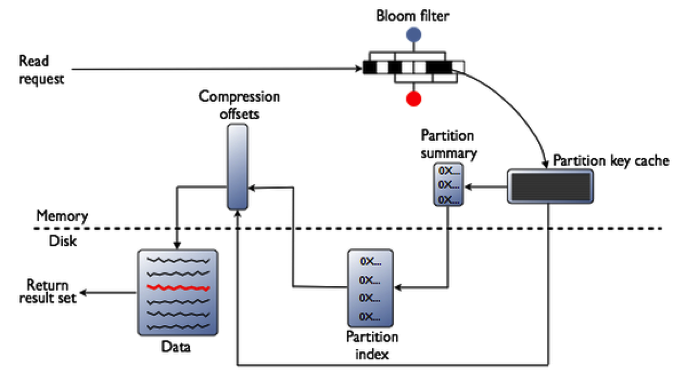


Figure 3. The Cassandra Read Path

Data Distribution and Replication

While the prior section provides a general overview of read and write operations in Cassandra in the context of a single node, the cluster-wide I/O activity that occurs is somewhat more sophisticated due to the database’s distributed, masterless architecture. Two concepts that impact read and write activity are the data distribution and replication models.

Automatic Data Distribution

Relational databases and some NoSQL systems require manual, developer-driven methods for distributing data across the multiple machines of a database cluster. These techniques are commonly referred to by the term "sharding." Sharding is an old technique that has seen some success in the industry, but is beset by inherent design and operational challenges. In contrast to this legacy architecture, Cassandra automatically distributes and maintains data across a cluster, freeing developers and architects to direct their energies into value-creating application features.

Cassandra has an internal component called a *partitioner*, which determines how data is distributed across the nodes that make up a database cluster. In short, 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 in a way that is predictable and consistent.

While the partitioner is a configurable property of a Cassandra cluster, 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 databases, Cassandra features a replication mechanism that is very straightforward and easy to configure and maintain. Most Cassandra users agree that its replication model is one of the features that makes the database stand out from other relational or non-relational options.

A Cassandra cluster can have one or more *keyspaces*, which are analogous to Microsoft SQL Server and MySQL *databases* or Oracle *schemas*. Replication is configured at the keyspace level, 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*. 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 fail.

Multi-Data Center and Cloud Support

A popular aspect of Cassandra’s replication is its support for multiple data centers and cloud availability zones. Many users deploy Cassandra in this 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 geographically diverse data centers, with users being able to read and write to any data center they choose and the data being automatically synchronized across all locations.

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

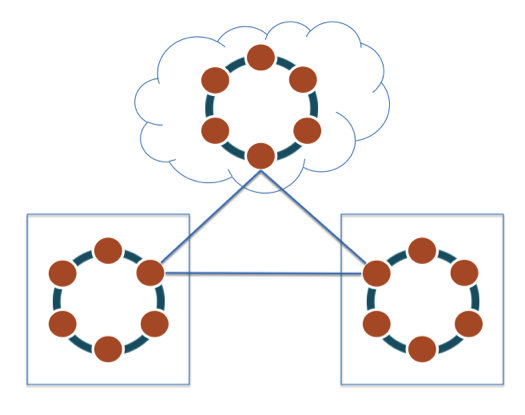


Figure 4. Cassandra supports multi-data-center and cloud deployments.

Data Management Concepts

This section provides a brief introduction to Cassandra’s data model, along with the data structures and query language Cassandra uses to manage data.

Data Model Overview

Cassandra is a wide-row-store database that uses a highly denormalized model designed to capture and query data performantly. Although Cassandra has objects that resemble a relational database (e.g., tables, primary keys, indexes, etc.), Cassandra data modeling techniques necessarily depart from the relational tradition. For example, the legacy entity-relationship-attribute data modeling paradigm is not appropriate in Cassandra the way it is with a relational database.

Success with Cassandra almost always comes down to getting two things right: the data model and the selected hardware—especially the storage subsystem. The topic of data modeling is of prime importance.

Unlike a relational database that penalizes the use of many columns in a table, Cassandra is highly performant with tables that have thousands or even tens of thousands of columns. Cassandra provides helpful data modeling abstractions to make this paradigm approachable for the developer.

Cassandra Data 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 databases. It is also the level at which replication is defined.
* Table – somewhat like a relational table, but capable of holding vastly large volumes of data. A table is also able to provide very fast row inserts and column level reads.
* Primary key – used to identity a row uniquely in a table and also distribute a table’s rows across multiple nodes in a cluster.
* Index – similar to a relational index in that it speeds some read operations; also different from relational indices in important ways.

Cassandra Query Language

Early versions of Cassandra exclusively used the programmatic *Thrift* interface to create database objects and manipulate data. While Thrift is still supported and maintained in Cassandra, the Cassandra Query Language (CQL) has become the primary API used for interacting with a Cassandra cluster today. This represents a substantial improvement in Cassandra’s usability.

CQL resembles the standard SQL used by all relational databases. Because of that similarity, the learning curve for those coming from the relational world is reduced. DDL (CREATE, ALTER, DROP), DML (INSERT, UPDATE, DELETE, TRUNCATE), and query (SELECT) operations are all supported.

CQL datatypes also reflect RDBMS syntax with numerical (int, bigint, decimal, etc.), character (ascii, varchar, etc.), date (timestamp, etc.), unstructured (blob, etc.), and specialized datatypes (set, list, map, etc.) being supported.

Various CQL command line utilities like cqlsh and graphical tools like DataStax DevCenter can be used to interact with a Cassandra cluster, and client drivers for Cassandra (Java, C#, etc.) also support CQL for developing applications.

Transaction Management

While Cassandra does not support ACID transactions like most legacy relational databases, it does offer the “AID” portion of ACID. Writes to Cassandra are atomic, isolated, and durable. The “C” of ACID—​consistency—​does not apply to Cassandra, as there is no concept of referential integrity or foreign keys.

Cassandra offers tunable data consistency across a database cluster. This means you can decide whether you want strong or eventual consistency for a particular transaction. You might want a particular request to complete if just one node responds, or you might want to wait until all nodes respond. Tunable data consistency is supported across single or multiple data centers, and you have a number of different consistency options from which to choose.

Consistency is configurable on a per-query basis, meaning you can decide how strong or eventual consistency should be per SELECT, INSERT, UPDATE, and DELETE operation. For example, if you need a particular transaction to be 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 a compare-and-set mechanism. Using and extending the Paxos consensus protocol (which allows a distributed system to agree on proposed data modifications with a quorum-based algorithm, and 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 relational database.

Migrating Data to Cassandra

Moving data from an RDBMS or other database to Cassandra is fairly easy depending on the state of the existing data. The following options currently exist for migrating data to Cassandra:

* COPY command – The cqlsh utility provides a copy command that is able to load data from an operating system file into a Cassandra table. Note that this is not recommended for very large files.
* SSTable loader – this utility is designed for more quickly loading a Cassandra table with a file that is delimited in some way (e.g. comma, tab, etc).
* Sqoop – Sqoop is a utility used in Hadoop to load data from relational databases into a Hadoop cluster. DataStax supports pipelining data from a relational databases table directly into a Cassandra table in its production certified Cassandra platform (DataStax Enterprise).
* ETL tools – there are a variety of ETL tools 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).

Management and Administration Basics

Cassandra is designed to minimize the need for active management and maintenance. Even so, there are a number of Cassandra management tasks and considerations that you should keep in mind when developing and maintaining your clusters.

Security

Cassandra provides a familiar security paradigm for anyone coming from a relational database. Cassandra defaults to having no security enabled. You can enable security and create standard login accounts with passwords, and handle operational and data permission management via the familiar GRANT/REVOKE method of assigning and removing permissions.

Cassandra also provides encryption options for data being sent over the wire from a client to a database cluster as well as encryption for node-to-node communications.

Advanced security is available in DataStax Enterprise and includes external authentication capabilities (e.g. Kerberos), transparent data encryption for tables, and data auditing.

Backup and Recovery

Cassandra provides a number of backup options to ensure data protection and restoration should data loss occur. While some users of Cassandra rely on the database’s replication feature to supply a form of data backup to other locations, this plan does not protect against accidental deletion of data or dropping of database objects. Those actions will be replicated to any remote sites that participate in the cluster, so a regular backup plan remains an important part of any Cassandra deployment.

DataStax OpsCenter (a graphical management and monitoring tool for Cassandra and DataStax Enterprise) provides a visual way of scheduling and maintaining backups for a Cassandra database as well as doing graphical restores for a cluster, down to object level restores if desired. Standard command-line tools also provide the ability to do scripted snapshots of node data across a cluster.

Ensuring Data Consistency

Because of Cassandra’s shared-nothing distributed architecture, there are occasions when data inconsistencies occur in a cluster due to incidents like nodes becoming unavailable or disruptions in network connectivity. Cassandra takes steps automatically to reduce the impact of cluster "entropy" of this kind, but administrators must also periodically run maintenance operations to ensure that data is consistent across all nodes in the cluster.

Cassandra provides a scriptable operation called *repair* that you can run as needed to ensure that all nodes are consistent with respect to the data they contain. For production environments, DataStax Enterprise supplies an automated repair service that transparently handles this administration task for you.

Conclusion

Now that you have a general understanding of Cassandra’s core features and capabilities, you’re one step ahead of the game in your continued studies into its details and most successful use cases. Continue exploring DataStax Academy for more!