medium.com

Introduction to Apache Cassandra - the “Lamborghini” of the NoSQL World

DataStax

10-13 minutes

Authors: David Jones-Gilardi, Aleksandr Volochnev, Stefano Lottini

Welcome to the first post in our six-part series on Apache Cassandra®. As an infinitely scalable database, Cassandra is widely regarded as the Lamborghini of the NoSQL database world. In this post, we’ll introduce you to NoSQL databases, the CAP theorem, and explain how Cassandra works.

Apache Cassandra® is a distributed NoSQL database used by the vast majority of Fortune 100 companies. By helping companies like Apple, Facebook, and Netflix process large volumes of fast-moving data in a reliable, scalable way, Cassandra has become essential

for the mission-critical features we rely on today.

In this post, which is based on our video tutorial introducing Apache Cassandra, we will:

- Discuss NoSQL databases, and the power of purpose-built databases
- Introduce Cassandra, a peer-to-peer database
- Explain the Consistency, Availability, and Partition Tolerance (CAP) theorem (i.e. the law of distributed systems)
- Demonstrate how to structure data with tables and partitions
- Share hands-on exercises you can complete on GitHub

From SQL to NoSQL: Why NoSQL was invented

Relational database management systems (RDBMS) dominated the market for decades. Then, with the rise of Big Tech like Apple, Facebook, and Instagram, the global datasphere skyrocketed 15-fold in the last decade. And, RDBMS simply weren't ready to cope with the new data volume, nor new performance requirements.

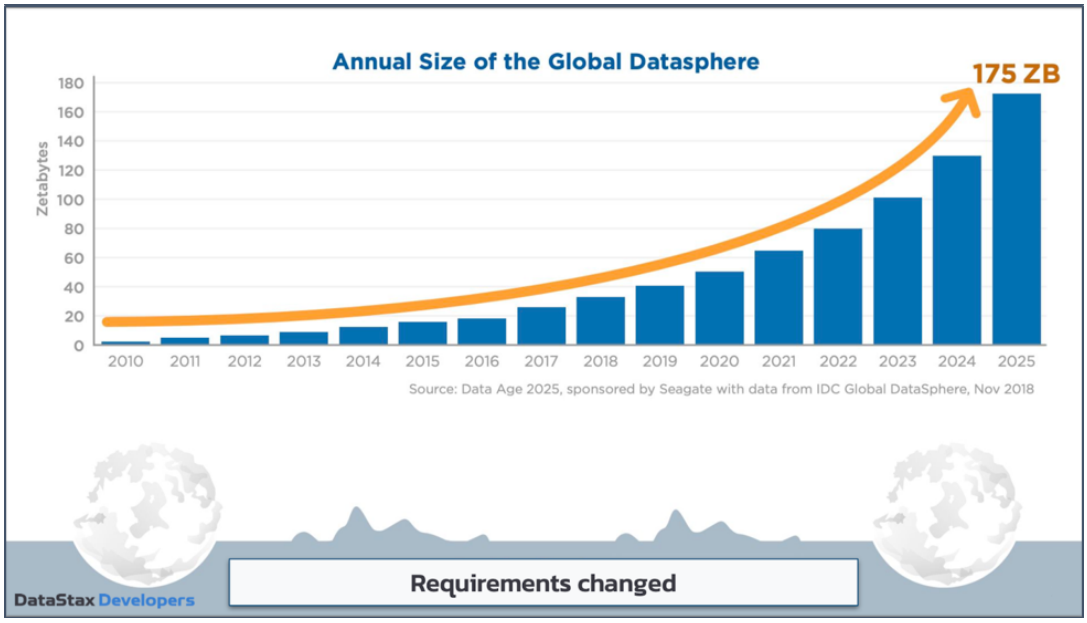


Figure 1. Skyrocketing data needs.

NoSQL was not only invented to cope with massive volumes of data, but also to tackle the challenge of both **velocity** (speed requirements) and **variety** (all the different types of data and data relations in the market).

Other than tabular databases like Cassandra, we've also seen the rise of other types of NoSQL databases, such as:

- **Time-series databases** (e.g. Prometheus)
- **Document databases** (e.g. MongoDB)
- **Graph databases** (e.g. DataStax Graph)
- **Ledger databases** (e.g. Amazon QLDB)
- **Key/value databases** (e.g. Amazon DynamoDB)

What makes Cassandra so powerful?

Known for its performance at scale, Cassandra is regarded as the Lamborghini of the NoSQL database world: it is essentially infinitely scalable. There's no leader node, and Cassandra is a peer-to-peer system.

For example, at Netflix, Cassandra runs **30 million ops/second** on its most active single cluster and 98% of streaming data is stored on Cassandra. Apple runs 160,000+ Cassandra instances with thousands of clusters.

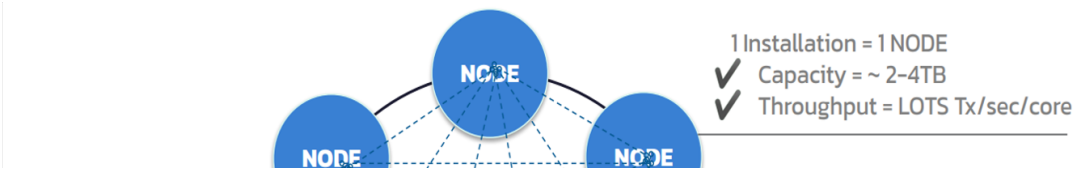
There are eight features that makes Cassandra powerful:

1. **Big data ready:** Partitioning over distributed architecture makes the database capable of handling data of any size: at petabyte scale. Need more volume? Add more nodes.
2. **Read-write performance:** A single node is very performant, but a cluster with multiple nodes and data centers brings throughput to the next level. Decentralization (leaderless architecture) means that every node can deal with any request, read or write.

3. **Linear scalability:** There are no limitations on volume or velocity and no overhead on new nodes. Cassandra scales with your needs.
4. **Highest availability:** Theoretically, you can achieve 100% uptime thanks to replication, decentralization and topology-aware placement strategy.
5. **Self-healing and automation:** Operations for a huge cluster can be exhausting. Cassandra clusters alleviate a lot of headaches because they are smart — able to scale, change data replacement, and recover — all automatically.
6. **Geographical distribution:** Multi-data center deployments grant an exceptional capability for disaster tolerance while keeping your data close to your clients, wherever they are in the world.
7. **Platform agnostic:** Cassandra is not bound to any platform or service provider, which allows you to build hybrid-cloud and multi-cloud solutions with ease.
8. **Vendor independent:** Cassandra doesn't belong to any of the commercial vendors but is offered by a non-profit open-source Apache Software Foundation, ensuring both open availability and continued development.

How does Cassandra work?

In Cassandra, all servers are created equal. Unlike traditional architecture, where there's a leader server for write/read and follower servers for read-only, leading to a single point of failure, Cassandra's leader-less (peer to peer) architecture distributes data across multiple nodes within clusters (also known as data centers or rings).



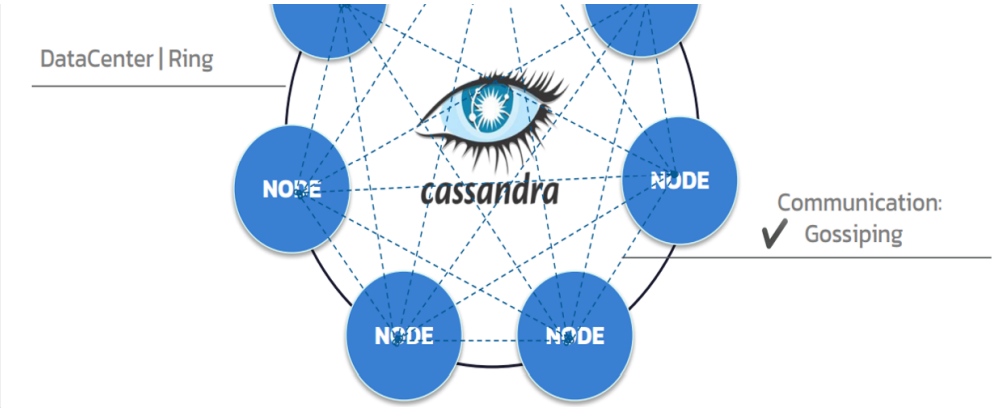


Figure 2. Apache Cassandra structure.

A node represents a single instance of Cassandra, and each node stores a few terabytes of data. Nodes “gossip” or exchange state information about itself and other nodes across the cluster for data consistency. When one node fails, the application contacts another node, ensuring 100% uptime.

In Cassandra, data is replicated. The **replication factor** (RF) represents the number of nodes used to store your data. If RF = 1, every partition is stored on one node. If RF = 2, then every partition is stored on two nodes, and so on. The industry standard is a **replication factor of three**, though there are cases that call for using more or fewer nodes.

For a more detailed explanation, watch our Introduction to Cassandra tutorial to understand how data replication works on Cassandra.

The CAP theorem: is Cassandra AP or CP?

The famous “CAP” theorem states that a distributed database system can **only guarantee two out of these three characteristics in case of a failure scenario**: Consistency, Availability, and Partition Tolerance:

1. **Consistency**: This means “no stale data.” A query returns the most recent value. If one of the servers returns outdated information, then your system is inconsistent.

2. **Availability:** This basically means “uptime.” If servers fail but still gives a response, then your system is available**.**
3. **Partition Tolerance:** This is the ability of a distributed system to survive “network partitioning.” Network partitioning means part of the servers cannot reach the second part.

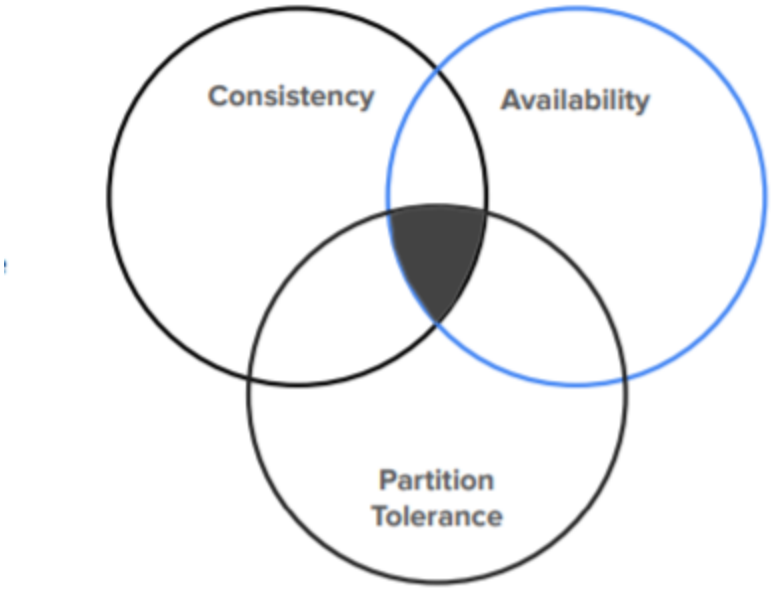


Figure 3. CAP Theorem governing databases.

Any database system, including Cassandra, has to guarantee partition tolerance: It must continue to function during data losses or system failures. To achieve partition tolerance, databases have to either prioritize consistency over availability “CP,” or availability over consistency or “AP”.

Cassandra is usually described as an “AP” system, meaning it errs on the side of ensuring data availability even if this means sacrificing consistency. But that’s not the whole picture. **Cassandra is configurably consistent: You can set the Consistency Level you need and tune it to be more AP or CP according to your use case**. If you want to take a deeper dive, you’ll find a more detailed explanation in our video tutorial.

How does Cassandra structure and distribute data?

Cassandra's innate architecture can handle and distribute massive amounts of data across thousands of servers without experiencing downtime. Each Cassandra node and even each Cassandra driver knows data allocation in a cluster (it's called token-aware), so your application can contact just about any server and receive fast answers.

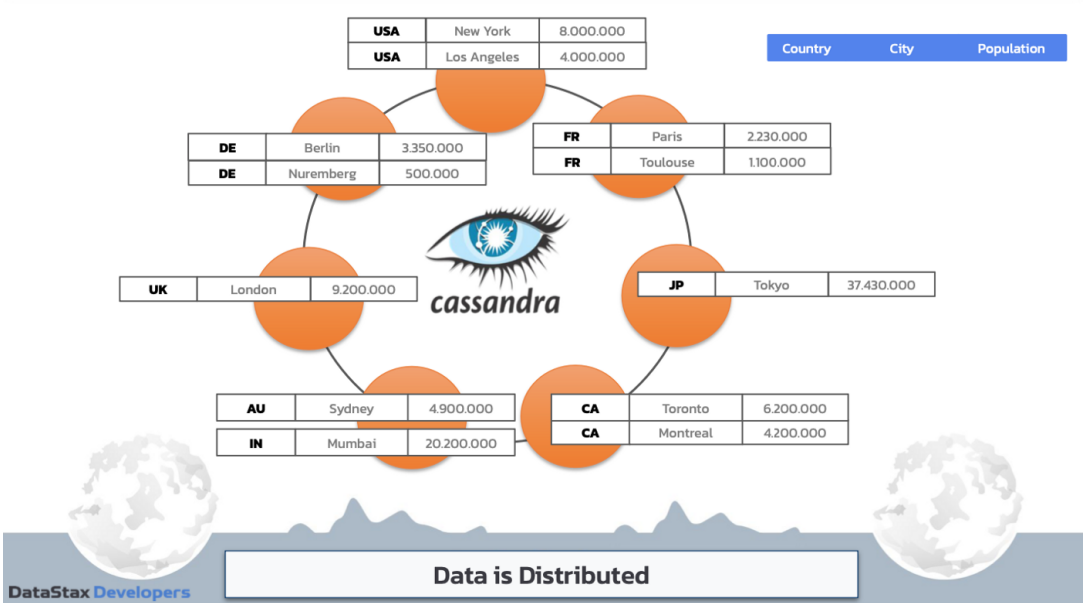
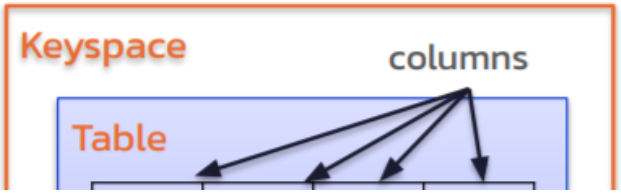


Figure 4. Data distribution across multiple nodes.

Cassandra uses key-based partitioning. The main components of Cassandra's data structure include:

- **Keyspace:** A container of data, similar to a schema, which contains several tables.
- **Table:** A set of columns, primary key, and rows storing data in partitions.
- **Partition:** A group of rows together with the same partition token (a base unit of access in Cassandra).
- **Row:** A single, structured data item in a table.



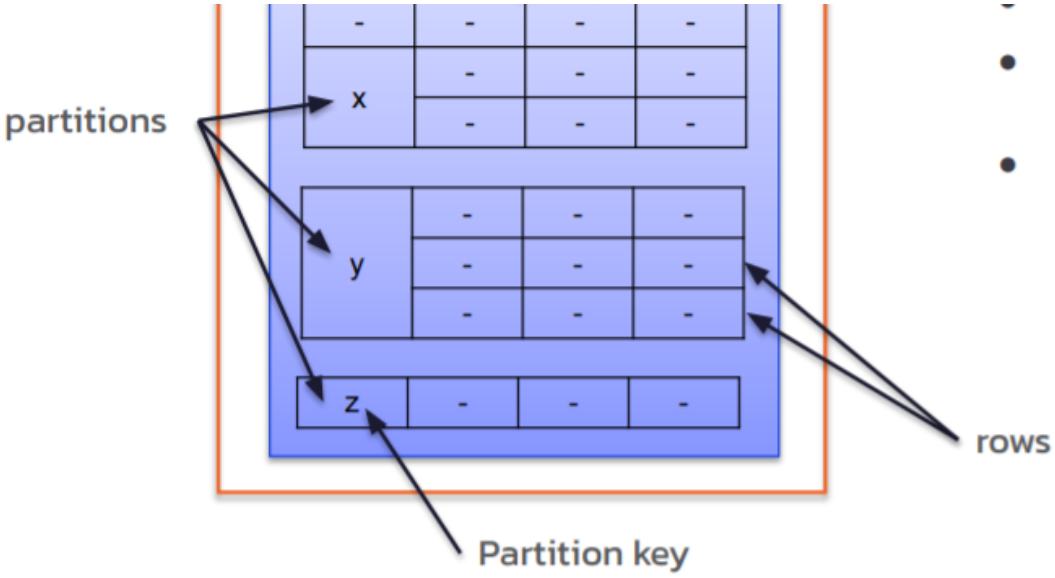


Figure 5. Overall data structure on Cassandra.

Cassandra stores data in partitions, representing a set of rows in a table across a cluster. Each row contains a partition key — one or more columns that are hashed to determine how data is distributed across the nodes in the cluster.

Why partitioning? Because this makes scaling so much easier! Big Data doesn't fit in a single server. Instead, it's split into chunks that are easily spread over dozens, hundreds or even thousands of servers, adding more if needed.

Once you set a partition key for your table, a partitioner transforms the value in the partition key to tokens (also called hashing) and assigns every node with a range of data called a token range.

Cassandra then distributes each row of data across the cluster by the token value automatically. If you need to scale up, just add a new node, and your data gets redistributed according to the new token range assignments. On the flip side, you can also scale down hassle-free.

Data architects need to know how to create a partition that returns queries accurately and quickly before they create a data model. Once you've set a primary key for your table, it cannot be changed.

Instead, you'll need to create a new table and migrate all the new data. To gain a better understanding of how to create a good partition, watch our video tutorial to step through a real-life example.

Hands-on exercises with Cassandra

Now, let's create your own Cassandra database in the cloud with DataStax's Astra DB. Sign up for an Astra DB account and choose the free tier with 80 gigabytes. Set up your database with database name, keyspace name, provider, and region. Our GitHub and video tutorial will walk through:

1. Creating your Astra DB instance
2. Creating tables
3. Executing CRUD (create, read, update, delete) operations

Whether you join our live workshop or go at your own pace, the repository will show you the most important fundamentals and basics of the powerful distributed NoSQL database Apache Cassandra for every developer who wants to try to learn a new database: creating tables and CRUD operations.

Conclusion

We hope you enjoyed this initial introduction to Cassandra with basic exercises to get you started. Check out Part 2 for more practical applications of Cassandra on topics like advanced data modeling, then head on over to Part 3 to learn about benchmarking your database performance.

In the meantime, if you'd like to learn more about Cassandra, check out our Apache Cassandra course on DataStax Academy and join our community to discuss NoSQL with experts in the field. You can also peruse our post on how to deploy a machine learning model

using Cassandra.

Follow the DataStax Tech Blog for more developer stories. Check out our YouTube channel for tutorials and DataStax Developers on Twitter for the latest news about our developer community.

Resources

1. Apache Cassandra: Open-source NoSQL Database
2. DataStax Astra DB: Multi-cloud DBaaS built on Apache Cassandra
3. Astra DB Sign Up
4. DataStax Apache Cassandra Course
5. YouTube Tutorial: Introduction to Apache Cassandra
6. Introduction to Apache Cassandra GitHub
7. DataStax Enterprise Graph
8. DataStax Academy
9. Real-World Machine Learning with Apache Cassandra and Apache Spark (Part 1)
10. Build CRUD Operations with Node JS and Python on DataStax Astra DB
11. Data Age 2025: The Datasphere and Data-readiness From Edge to Core