

UNIT- III

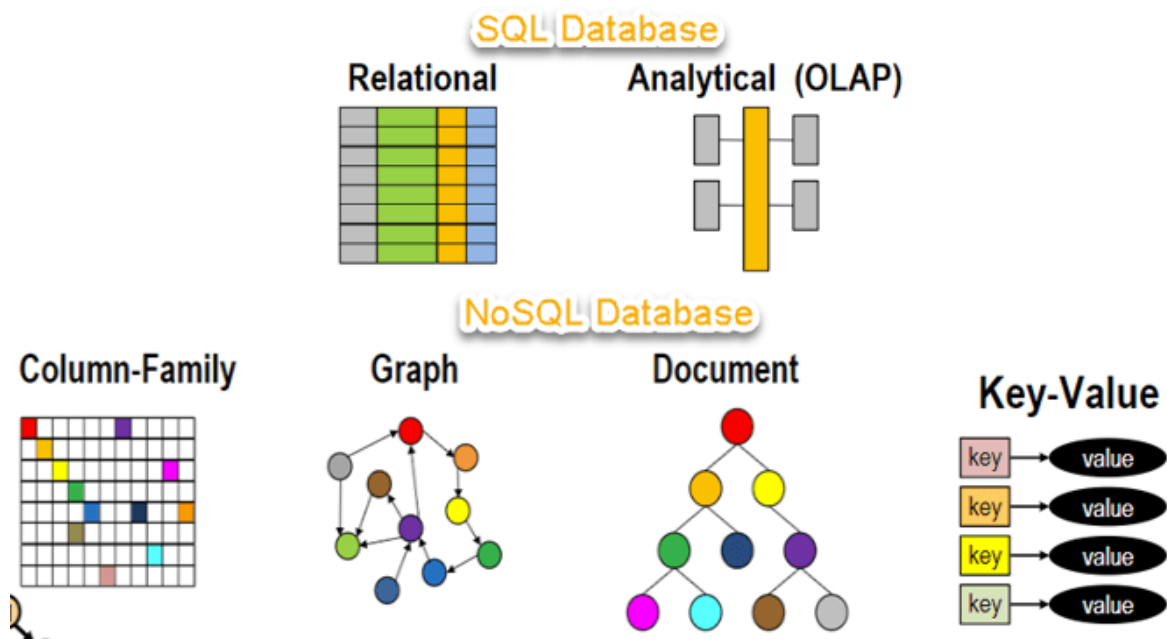
Introduction to NoSQL - MongoDB – Spark – Cassandra - Cassandra Data Model – Data Design – Cassandra Architecture – Read and Write Data – Clients – Integrate with Hadoop. Introduction - Importance of Effective Data Visualization - Introduction to Tableau - Choosing the Right Chart Type - Using the Color Effectively Reducing Clutter - Dashboard Creation and Formatting.

3.1 Introduction to NoSQL

NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

NoSQL database stands for “Not Only SQL” or “Not SQL.” Though a better term would be “NoREL”, NoSQL caught on. Carl Stroz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured, and polymorphic data.



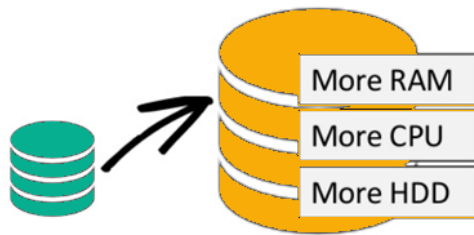
NoSQL

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

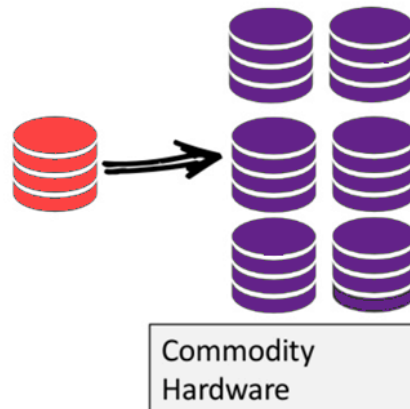
To resolve this problem, we could “scale up” our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute the database load on multiple hosts whenever the load increases. This method is known as “scaling out.”

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):



NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

Features of NoSQL

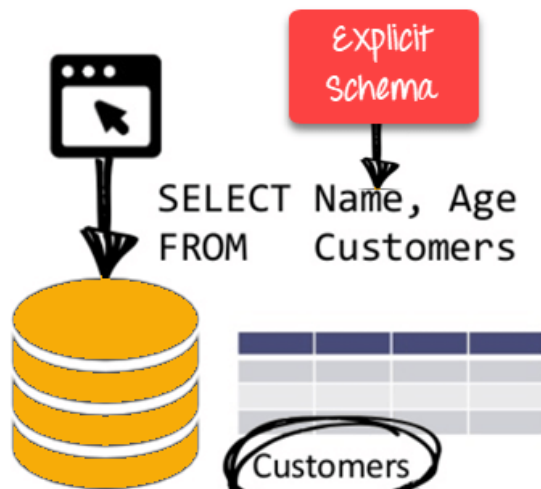
Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

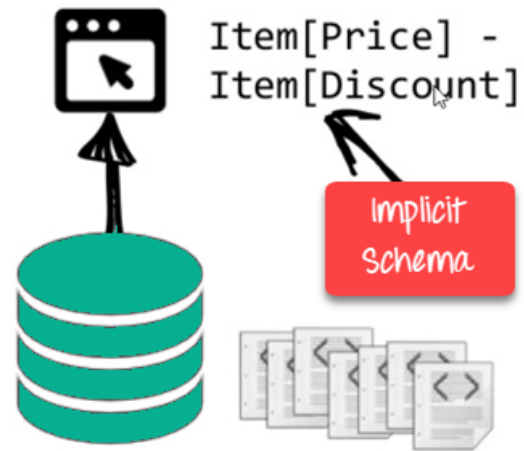
Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

RDBMS:



NoSQL DB:



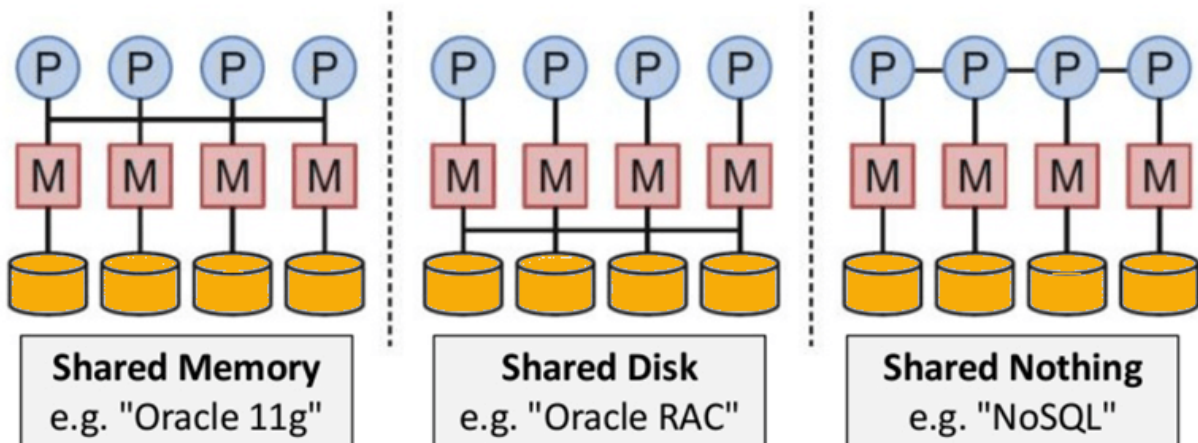
NoSQL is Schema-Free

Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based NoSQL query language
- Web-enabled databases running as internet-facing services

Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency
- Shared Nothing Architecture. This enables less coordination and higher distribution.



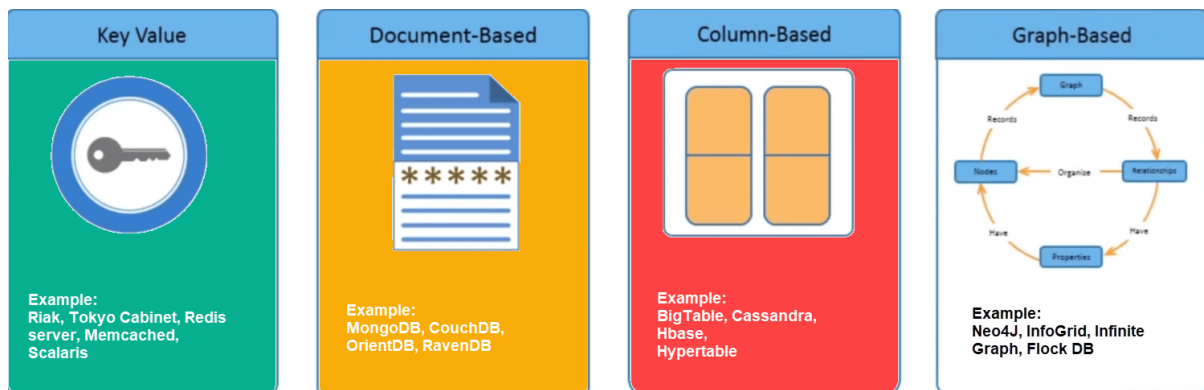
NoSQL is Shared Nothing.

Types of NoSQL Databases

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented



Key Value Pair Based

Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.

Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

For example, a key-value pair may contain a key like “Website” associated with a value like “Guru99”.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.

Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column-based NoSQL database

They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

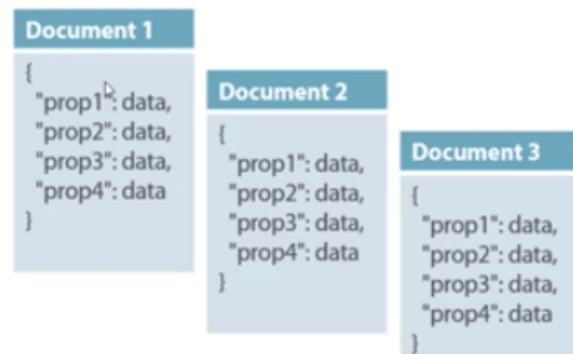
Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, HBase, and Hypertable are NoSQL query examples of the column-based database.

Document-Oriented:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data



Relational Vs. Document

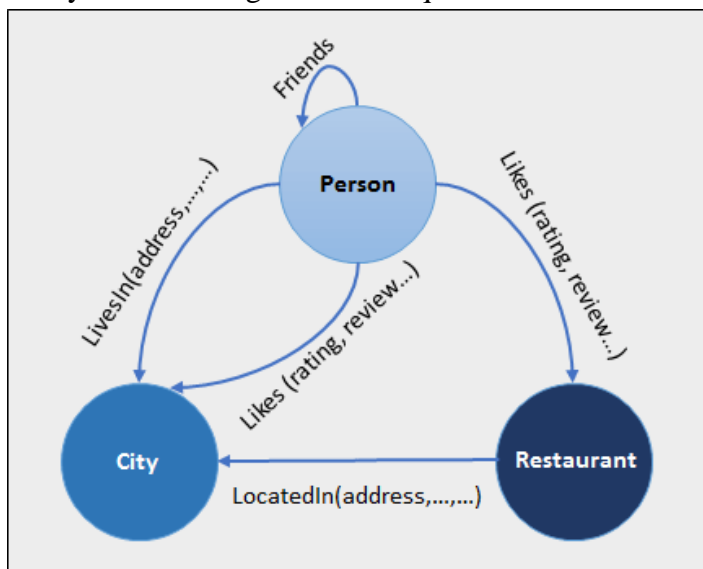
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

Graph-Based

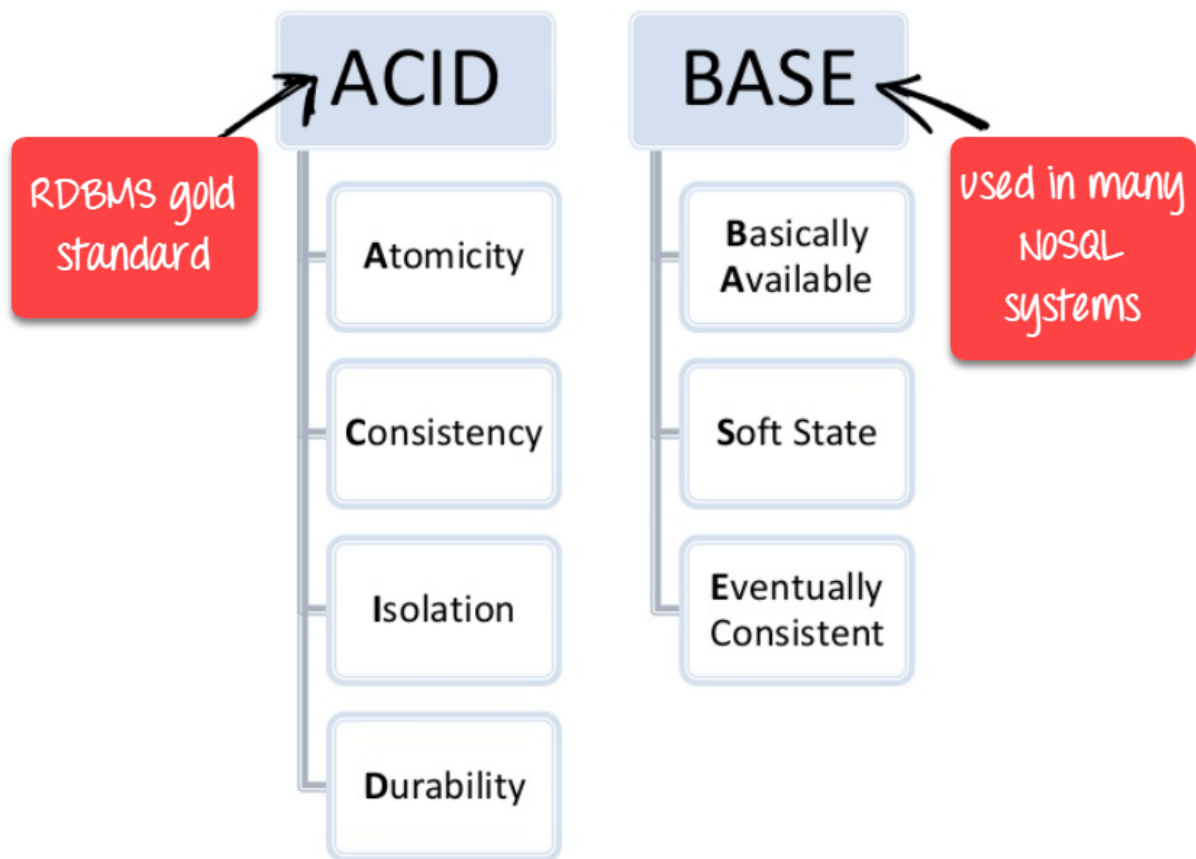
A graph-type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge have a unique identifier.



Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for social networks, logistics, spatial data.

Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.



Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible
- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities

- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open-source options so not so popular for enterprises.

MongoDB

MongoDB is a document-oriented NoSQL database used for high-volume data storage. Instead of using tables and rows as in traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and functions which is the equivalent of relational database tables. MongoDB is a database that came to light around the mid-2000s.

MongoDB Features

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. Scalability –MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with millions of documents within the database.

MongoDB Example

The below example shows how a document can be modeled in MongoDB.

1. The `_id` field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.

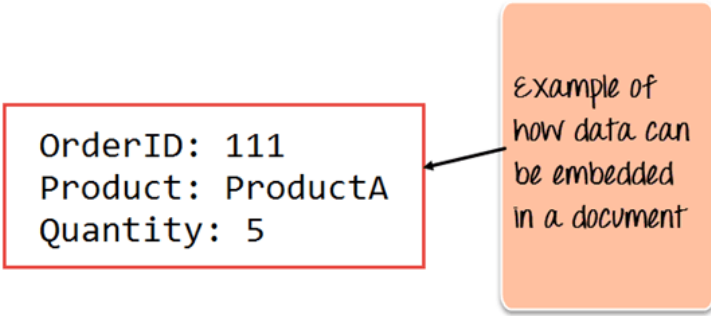

```

{
  _id : <ObjectId> ,

  CustomerName : Guru99 ,

  Order:
    {
      OrderID: 111
      Product: ProductA
      Quantity: 5
    }
}

```



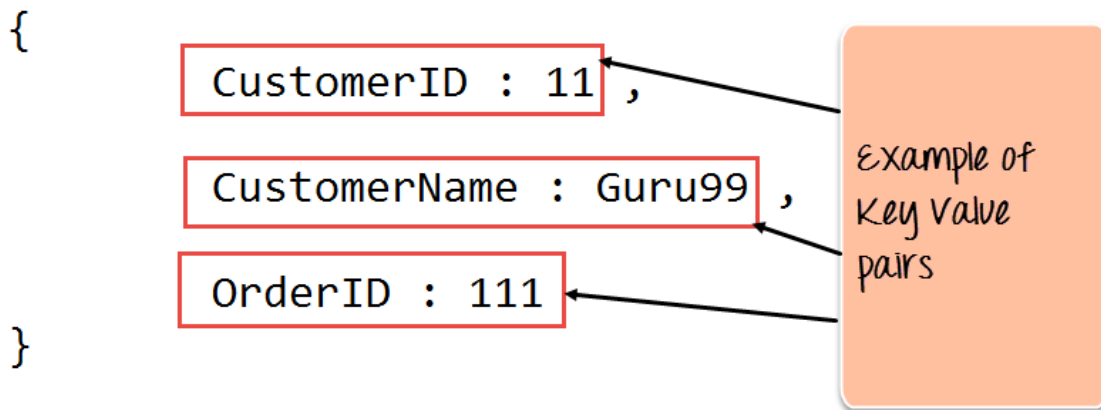
Key Components of MongoDB Architecture

Below are a few of the common terms used in MongoDB

1. **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

_Id	CustomerID	CustomerName
563479cc8a8a4246bd27d784	11	Guru99
563479cc7a8a4246bd47d784	22	Trevor Smith
563479cc9a8a4246bd57d784	33	Nicole

2. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.
3. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
4. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
5. **Document** – A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
6. **Field** – A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases. The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.



7. **JSON** – This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the `_id` field and a normal collection field. The `_id` field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

Use of bigdata

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries – MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. Indexing – Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. Replication – MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. Load balancing – MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

Difference between MongoDB & RDBMS

Below are some of the key term differences between MongoDB and RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contain Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB.

Apart from the differences in the terms, a few other differences are shown below

1. Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
2. RDBMS requires that data be normalized first so that it can prevent orphan records and duplicates Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes. As databases start to grow, performance can start becoming an issue. Again, this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

SPARK

Apache Spark is a general-purpose & lightning-fast cluster computing system. It provides a high-level API. For example, Java, Scala, Python, and R. Apache Spark is a tool for Running Spark Applications. Spark is 100 times faster than Bigdata Hadoop and 10 times faster than accessing data from disk. Spark is written in Scala but provides rich APIs in Scala, Java, Python, and R. It can be integrated with hadoop and can process existing Hadoop HDFS data.

History Of Apache Spark

Apache Spark was introduced in 2009 in the UC Berkeley R&D Lab, later it becomes AMP Lab. It was open-sourced in 2010 under a BSD license. In 2013 spark was donated to Apache Software Foundation where it became a top-level Apache project in 2014.

Need of Spark

In the industry, there is a need for a general-purpose cluster computing tool as:

- Hadoop MapReduce can only perform batch processing.
- Apache Storm / S4 can only perform stream processing.
- Apache Impala / Apache Tez can only perform interactive processing

- Neo4j / Apache Giraph can only perform graph processing

Hence in the industry, there is a big demand for a powerful engine that can process the data in real-time (streaming) as well as in batch mode. There is a need for an engine that can respond in sub-second and perform in-memory processing.

Apache Spark Definition says it is a powerful open-source engine that provides real-time stream processing, interactive processing, graph processing, in-memory processing as well as batch processing with very fast speed, ease of use, and standard interface. This creates the difference between Hadoop vs Spark and also makes a huge comparison between Spark vs Storm.

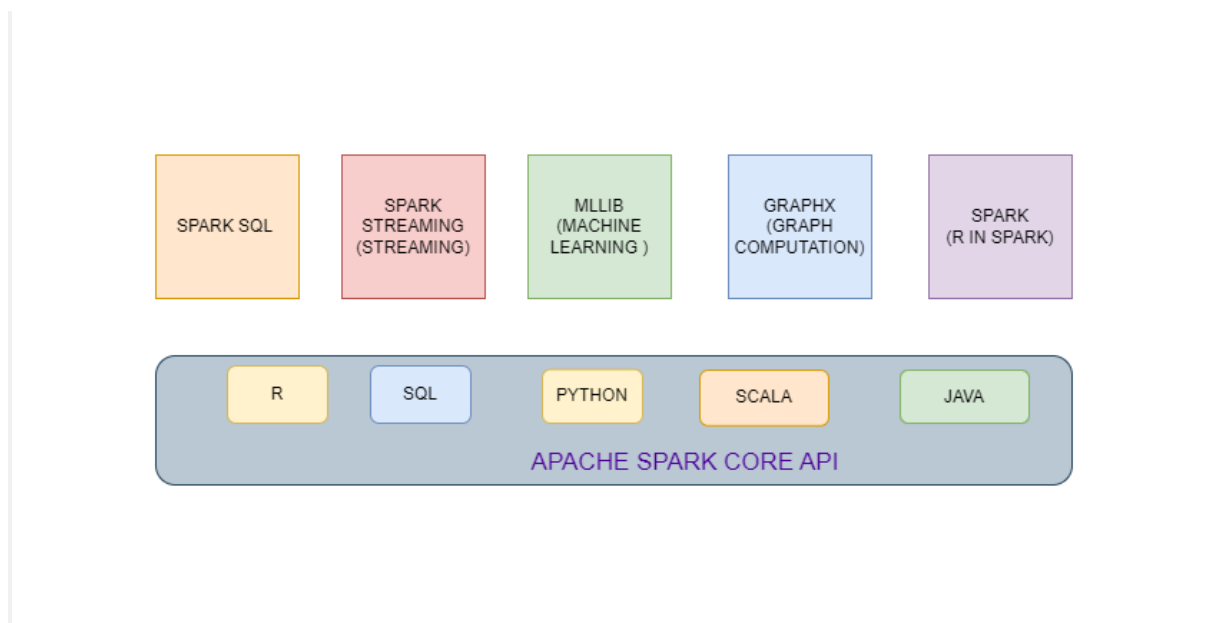
Features of Apache Spark

Apache Spark has following features.

- Speed – Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- Supports multiple languages – Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high level operators for interactive querying.
- Advanced Analytics – Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms

Apache Spark Components

Apache Spark puts the promise for faster data processing and easier development. How Spark achieves this? To answer this question, let's introduce the Apache Spark ecosystem which is the important topic in Apache Spark introduction that makes Spark fast and reliable. These components of Spark resolves the issues that cropped up while using Hadoop MapReduce.



i. Spark Core

It is the kernel of Spark, which provides an execution platform for all the Spark applications. It is a generalized platform to support a wide array of applications.

ii. Spark SQL

It enables users to run SQL/HQL queries on the top of Spark. Using Apache Spark SQL, we can process structured as well as semi-structured data. It also provides an engine for Hive to run unmodified queries up to 100 times faster on existing deployments.

iii. Spark Streaming

Apache Spark Streaming enables powerful interactive and **data analytics** application across live streaming data. The live streams are converted into micro-batches which are executed on top of spark core.

iv. Spark MLlib

It is the scalable machine learning library which delivers both efficiencies as well as the high-quality algorithm. Apache Spark MLlib is one of the hottest choices for **Data Scientist** due to its capability of in-memory data processing, which improves the performance of iterative algorithm drastically.

v. Spark GraphX

Apache Spark **GraphX** is the graph computation engine built on top of spark that enables to process graph data at scale.

vi. SparkR

It is an R package that gives a lightweight frontend to use Apache Spark from R. It allows data scientists to analyze large datasets and interactively run jobs on them from the R shell. The main idea behind SparkR was to explore different techniques to integrate the usability of R with the scalability of Spark.

Spark Architecture

The Spark follows the master-slave architecture. Its cluster consists of a single master and multiple slaves.

The Spark architecture depends upon two abstractions:

Resilient Distributed Dataset (RDD)

Directed Acyclic Graph (DAG)

Resilient Distributed Datasets (RDD)

The Resilient Distributed Datasets are the group of data items that can be stored in-memory on worker nodes. Here,

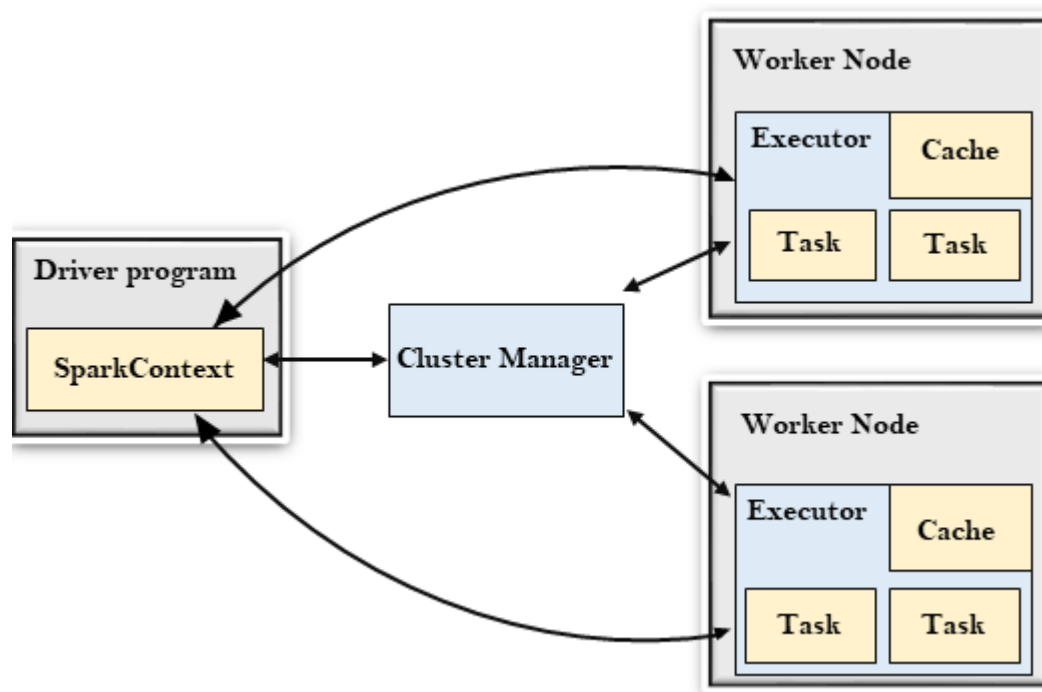
Resilient: Restore the data on failure.

Distributed: Data is distributed among different nodes.

Dataset: Group of data.

Directed Acyclic Graph (DAG)

Directed Acyclic Graph is a finite direct graph that performs a sequence of computations on data. Each node is an RDD partition, and the edge is a transformation on top of data. Here, the graph refers the navigation whereas directed and acyclic refers to how it is done.



Architecture of graph

Driver Program

The Driver Program is a process that runs the `main()` function of the application and creates the `SparkContext` object. The purpose of `SparkContext` is to coordinate the spark applications, running as independent sets of processes on a cluster.

To run on a cluster, the `SparkContext` connects to a different type of cluster managers and then perform the following tasks: -

- It acquires executors on nodes in the cluster.
- Then, it sends your application code to the executors. Here, the application code can be defined by JAR or Python files passed to the `SparkContext`.
- At last, the `SparkContext` sends tasks to the executors to run.

Cluster Manager

- The role of the cluster manager is to allocate resources across applications. The Spark is capable enough of running on a large number of clusters.
- It consists of various types of cluster managers such as Hadoop YARN, Apache Mesos and Standalone Scheduler.
- Here, the Standalone Scheduler is a standalone spark cluster manager that facilitates to install Spark on an empty set of machines.

Worker Node

- The worker node is a slave node
- Its role is to run the application code in the cluster.

Executor

- An executor is a process launched for an application on a worker node.
- It runs tasks and keeps data in memory or disk storage across them.
- It read and write data to the external sources.
- Every application contains its executor.

Task

A unit of work that will be sent to one executor.

Cassandra

Apache Cassandra is an open-source wide column NoSQL database created by the Apache Software Foundation (ASF). It was an early entrant in the NoSQL arena. Cassandra quickly became a popular database for non-relational data projects. It provides IT organizations with an easy way to scale data across multiple nodes and datacenters with fault tolerance and data redundancy.

Components of Cassandra

The key components of Cassandra are as follows –

- Node – It is the place where data is stored.
- Data center – It is a collection of related nodes.
- Cluster – A cluster is a component that contains one or more data centers.
- Commit log – The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- Mem-table – A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- SSTable – It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- Bloom filter – These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Data Model

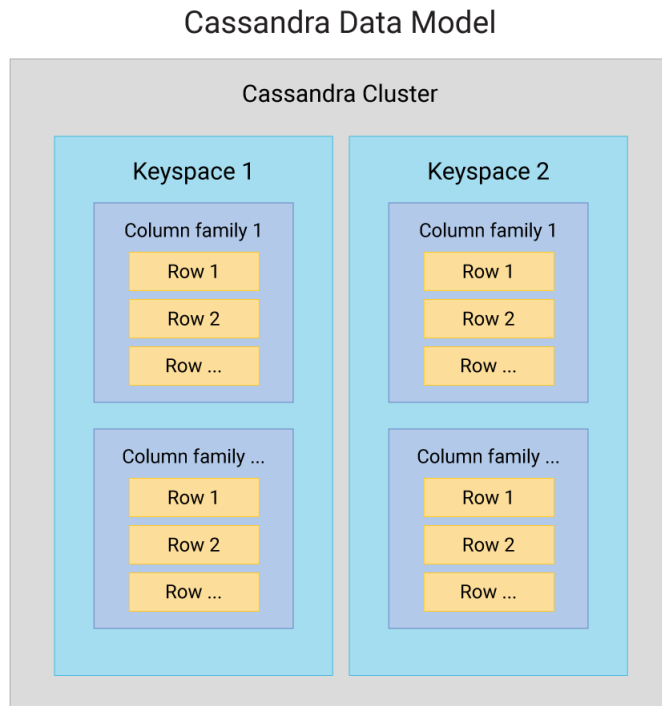
Apache Cassandra data model components include keyspaces, tables, and columns:

- Cassandra stores data as a set of rows organized into tables or column families
- A primary key value identifies each row
- The primary key partitions data
- You can fetch data in part or in its entirety based on the primary key

Keyspaces: At a high level, the Cassandra NoSQL data model consists of data containers called keyspaces. Keyspaces are similar to the schema in a relational database. Typically, there are many tables in a keyspace.

Tables: Tables, also called column families in earlier iterations of Cassandra, are defined within the keyspaces. Tables store data in a set of rows and contain a primary key and a set of columns.

Columns: Columns define data structure within a table. There are various types of columns, such as Boolean, double, integer, and text.



A Cassandra database is distributed across machines operating in tandem. Cassandra assigns data to nodes in the outermost container in a ring cluster: the Keyspace. Each node contains a replica that takes charge in case of failure handling.

Data modeling involves identifying items to be stored or entities and the relationships between them. Data modeling in Cassandra follows a query-driven approach, organizing data based on specific queries.

The design of Cassandra's database is based on fast read and write requirements, so the speed of data retrieval improves with schema design. Queries select data from tables; query patterns define example user phrases and schema defines how table data is arranged.

In contrast, relational databases write the queries that will be made only after normalizing data based on the relationships and tables designed. Unlike the query-driven Cassandra approach, data modeling in relational databases is table-driven. The database expresses relationships between tables in queries as table joins.

The Cassandra data model offers tuneable consistency, or the ability for the client application to choose how consistent the requested data must be for any given read or write operation. Tuning consistency is a factor in latency, but it is not part of the formal data modeling process.

Cassandra Keyspaces

In Cassandra, a Keyspace has several basic attributes:

Column families: Containers of rows collected and organized that represent the data's structure. There is at least one column family in each keyspace and there may be many.

Replication factor: The number of cluster machines that receive identical copies of data.

Replica placement strategy: Analogous to a load balancing algorithm, this is simply the strategy for placement of replicas in the ring cluster. There are rack-aware strategies and datacenter-shared strategies.

Cassandra Primary Keys

Each Cassandra table must have a primary key, a set of columns. (This is why tables were called column families in past iterations of Cassandra.) The primary key shapes the table's data structure and determines the uniqueness of a row.

There are two parts to the Cassandra primary key:

Partition key: The primary key is the required first column or set of columns. The hashed partition key value determines where in the cluster the partition will reside.

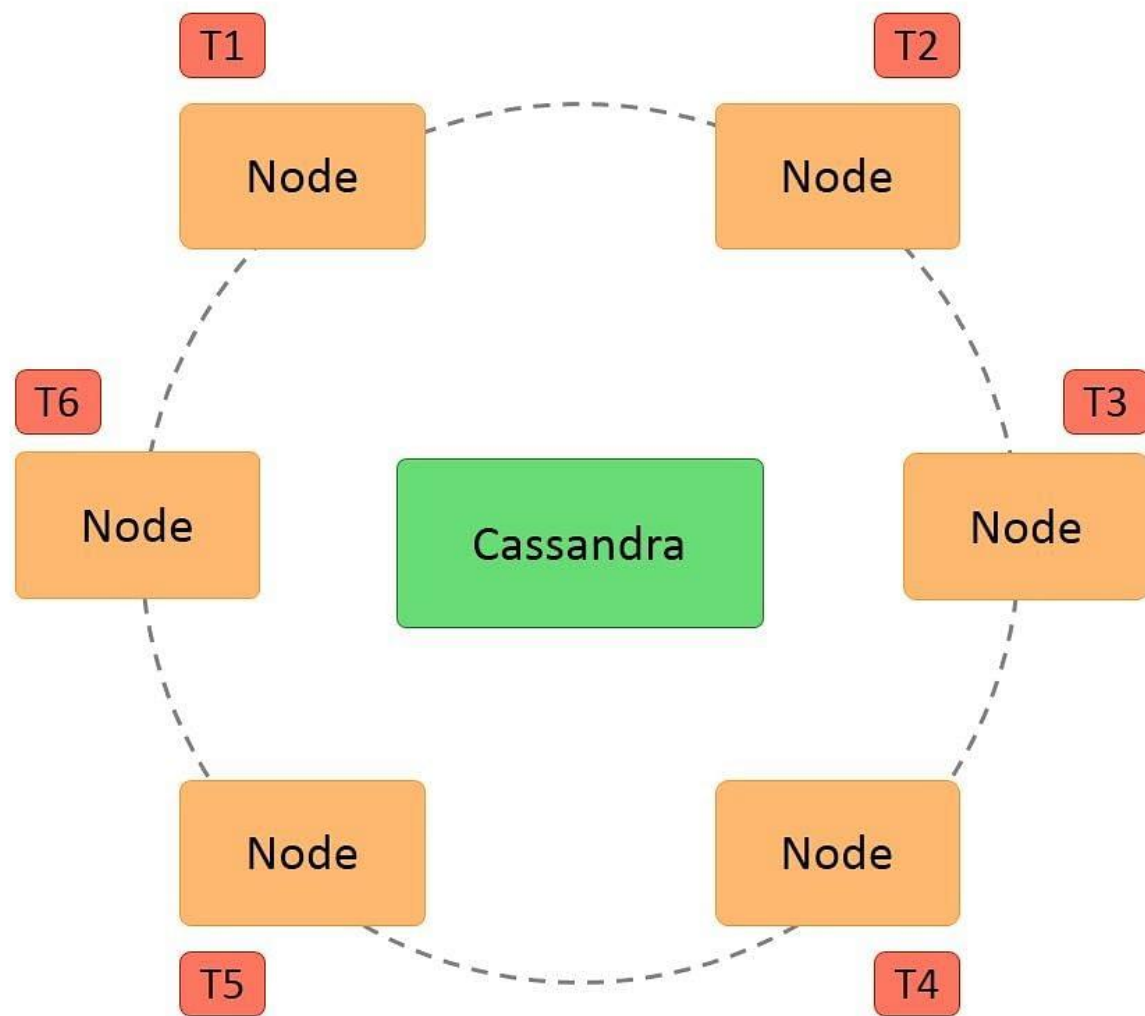
Clustering key: Also called clustering columns, clustering keys are optional columns after the partition key. The clustering key determines the order of rows sort themselves into within a partition by default.

Cassandra Architecture

Some of the features of Cassandra architecture are as follows:

- Cassandra is designed such that it has no master or slave nodes.
- It has a ring-type architecture, that is, its nodes are logically distributed like a ring.
- Data is automatically distributed across all the nodes.
- Similar to HDFS, data is replicated across the nodes for redundancy.
- Data is kept in memory and lazily written to the disk.
- Hash values of the keys are used to distribute the data among nodes in the cluster.

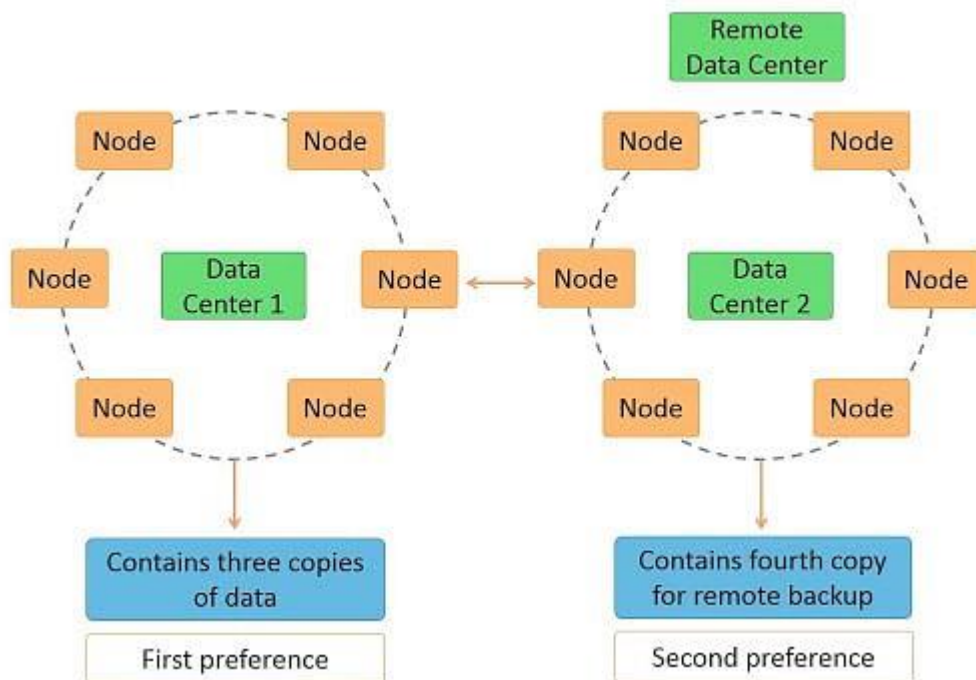
A hash value is a number that maps any given key to a numeric value. For example, the string 'ABC' may be mapped to 101, and decimal number 25.34 may be mapped to 257. A hash value is generated using an algorithm so that the same value of the key always gives the same hash value. In a ring architecture, each node is assigned a token value, as shown in the image below:



Additional features of Cassandra architecture are:

- Cassandra architecture supports multiple data centers.
- Data can be replicated across data centers.

You can keep three copies of data in one data center and the fourth copy in a remote data center for remote backup. Data reads prefer a local data center to a remote data center.



Effects of the Architecture

Cassandra architecture enables transparent distribution of data to nodes. This means you can determine the location of your data in the cluster based on the data. Any node can accept any request as there are no masters or slaves. If a node has the data, it will return the data. Else, it will send the request to the node that has the data.

You can specify the number of replicas of the data to achieve the required level of redundancy. For example, if the data is very critical, you may want to specify a replication factor of 4 or 5.

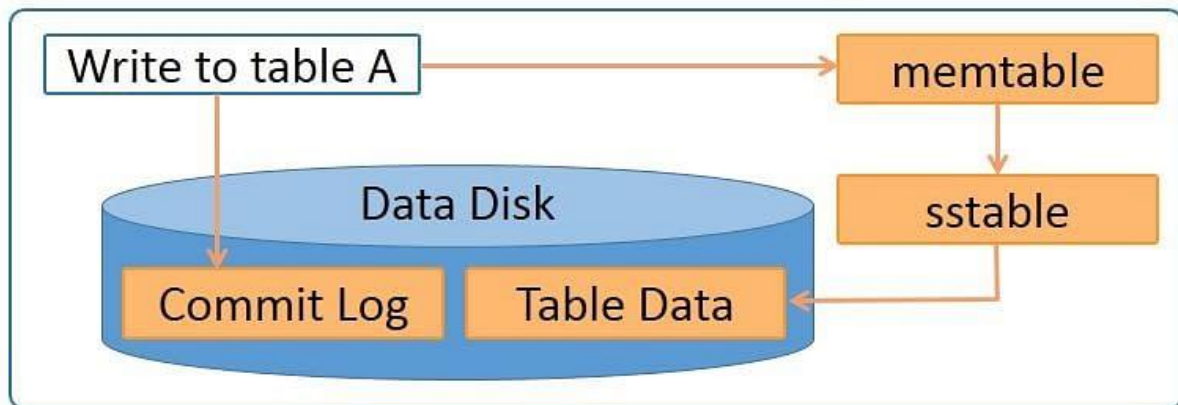
If the data is not critical, you may specify just two. It also provides tunable consistency, that is, the level of consistency can be specified as a trade-off with performance. Transactions are always written to a commit log on disk so that they are durable. Let us discuss the Cassandra write process in the next section of the Cassandra architecture tutorial.

Cassandra Write Process

The Cassandra write process ensures fast writes. Steps in the Cassandra write process are:

1. Data is written to a commitlog on disk.
2. The data is sent to a responsible node based on the hash value.
3. Nodes write data to an in-memory table called memtable.
4. From the memtable, data is written to an sstable in memory. Sstable stands for Sorted String table. This has a consolidated data of all the updates to the table.
5. From the sstable, data is updated to the actual table.
6. If the responsible node is down, data will be written to another node identified as tempnode. The tempnode will hold the data temporarily till the responsible node comes alive.

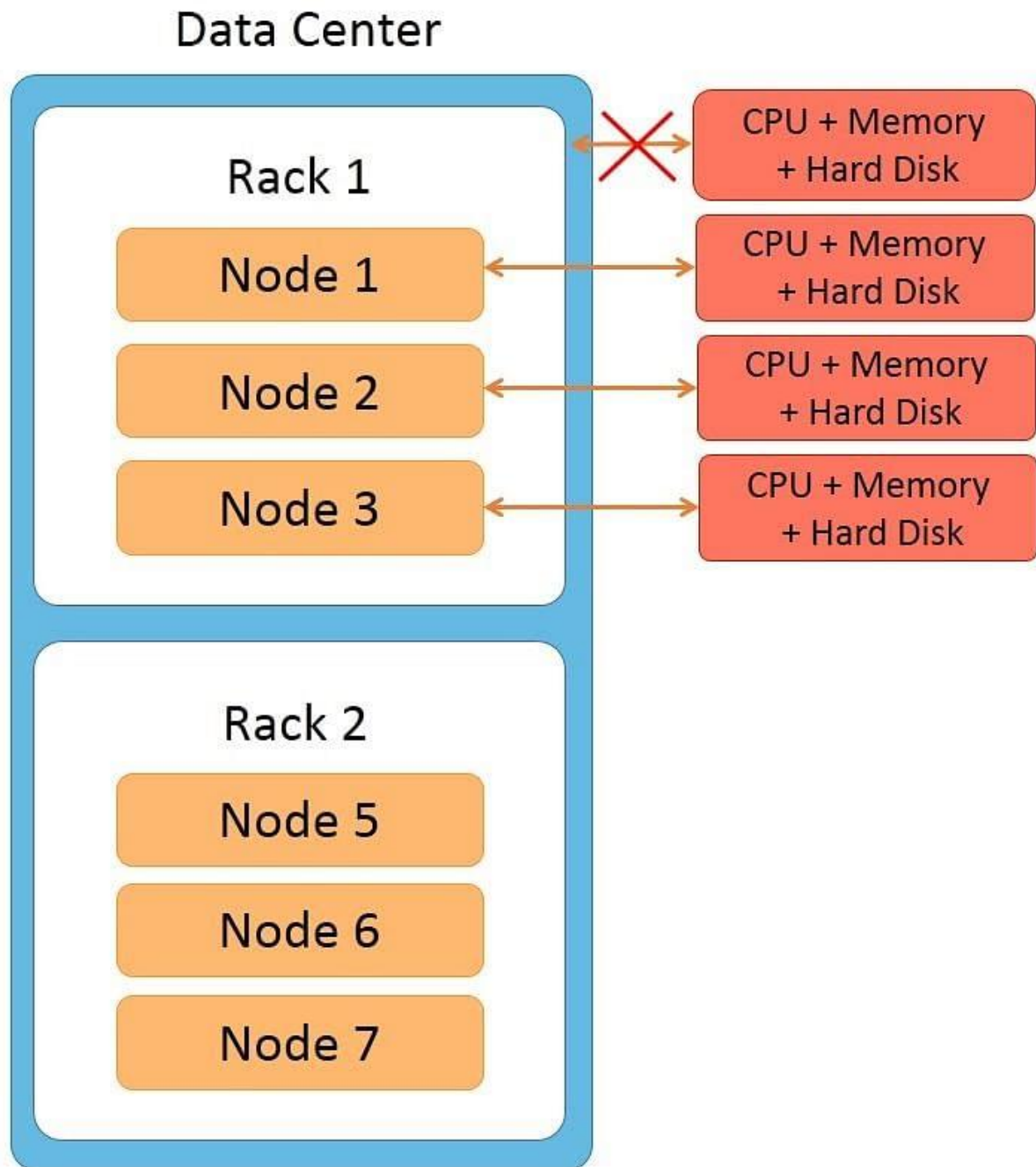
The diagram below depicts the write process when data is written to table A.



Data is written to a commitlog on disk for persistence. It is also written to an in-memory memtable. Memtable data is written to sstable which is used to update the actual table. Let us understand what rack is, in the next section of the cassandra architecture tutorial.

Rack

The term 'rack' is usually used when explaining network topology. A rack is a group of machines housed in the same physical box. Each machine in the rack has its own CPU, memory, and hard disk. However, the rack has no CPU, memory, or hard disk of its own.



Features of racks are:

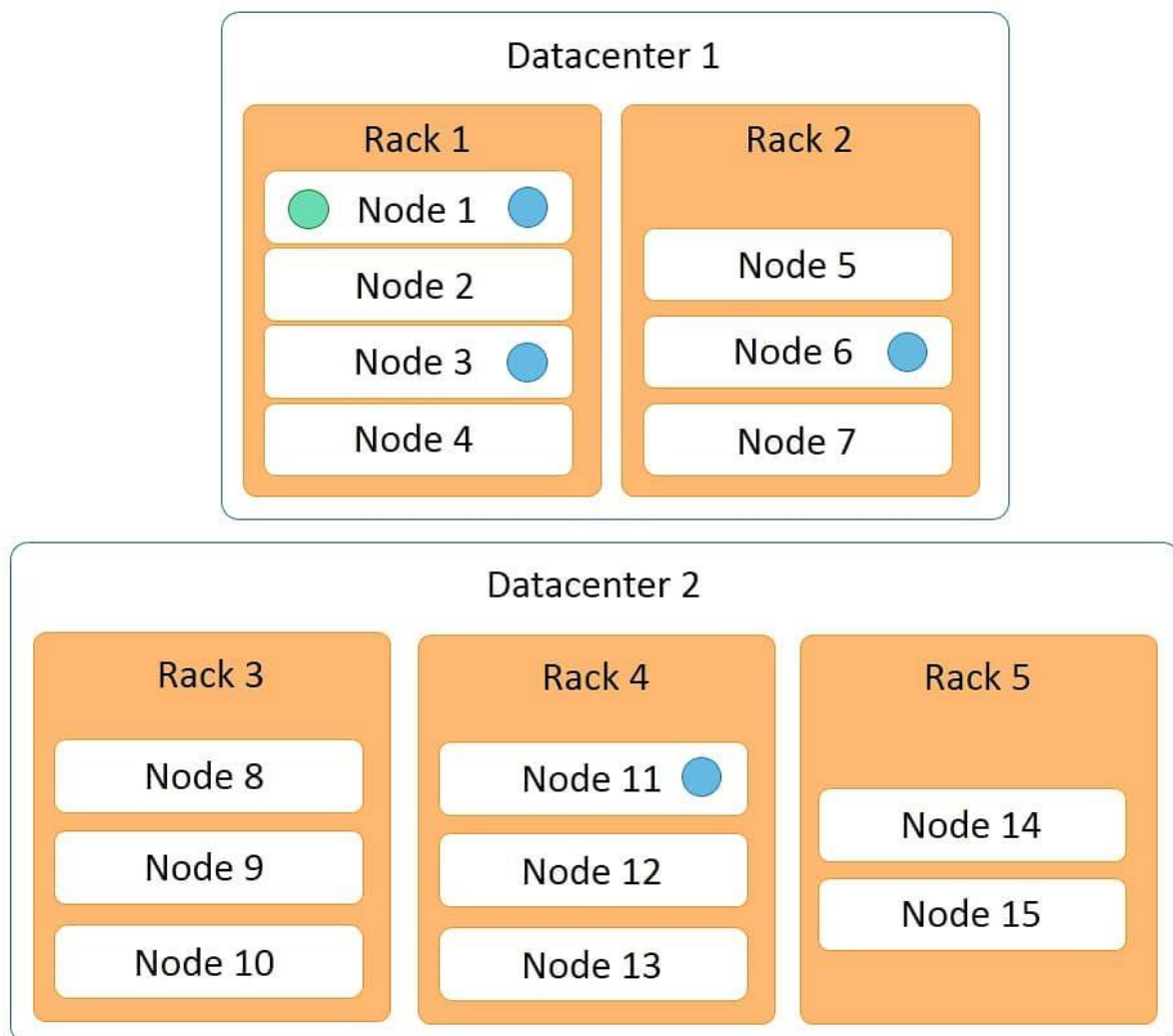
- All machines in the rack are connected to the network switch of the rack
- The rack's network switch is connected to the cluster.
- All machines on the rack have a common power supply. It is important to notice that a rack can fail due to two reasons: a network switch failure or a power supply failure.
- If a rack fails, none of the machines on the rack can be accessed. So it would seem as though all the nodes on the rack are down.

Cassandra Read Process

The Cassandra read process ensures fast reads. Read happens across all nodes in parallel. If a node is down, data is read from the replica of the data. Priority for the replica is assigned on the basis of distance. Features of the Cassandra read process are:

- Data on the same node is given first preference and is considered data local.
- Data on the same rack is given second preference and is considered rack local.
- Data on the same data center is given third preference and is considered data center local.
- Data in a different data center is given the least preference.

Data in the memtable and sstable is checked first so that the data can be retrieved faster if it is already in memory. The diagram below represents a Cassandra cluster.



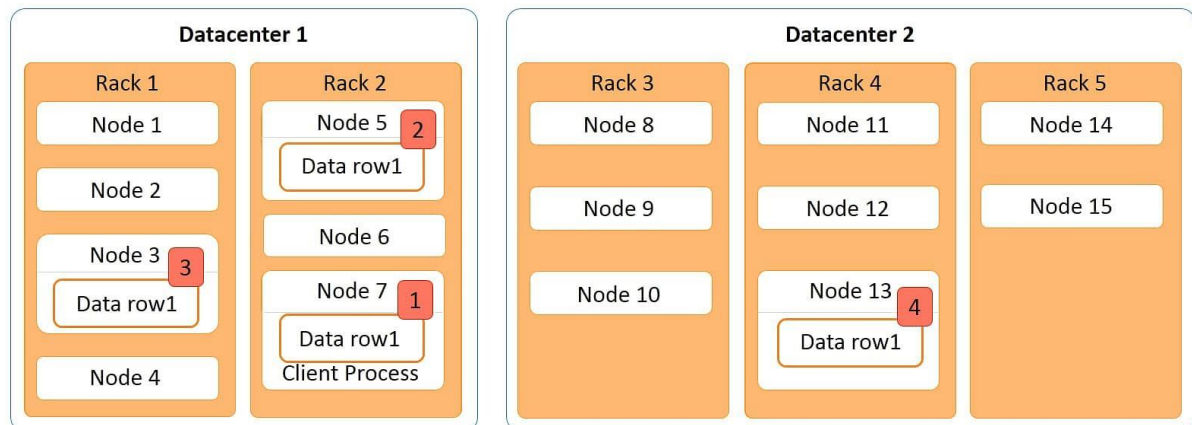
It has two data centers:

- data center 1
- data center 2

Data center 1 has two racks, while data center 2 has three racks. Fifteen nodes are distributed across this cluster with nodes 1 to 4 on rack 1, nodes 5 to 7 on rack 2, and so on. Let us discuss the example of Cassandra read process in the next section of the cassandra architecture tutorial.

Example of Cassandra Read Process

The Cassandra read process is illustrated with an example below.



The diagram below explains the Cassandra read process in a cluster with two data centers, five racks, and 15 nodes. In the image, place data row1 in this cluster. Data row1 is a row of data with four replicas.

- The first copy is stored on node 3
- The second copy is stored on node 5
- The third copy is stored on node 7

All these nodes are in data center 1. The fourth copy is stored on node 13 of data center 2. If a client process is running on data node 7 wants to access data row1; node 7 will be given the highest preference as the data is local here. The next preference is for node 5 where the data is rack local. The next preference is for node 3 where the data is on a different rack but within the same data center.

The least preference is given to node 13 that is in a different data center. So the read process preference in this example is node 7, node 5, node 3, and node 13 in that order. Let us focus on Data Partitions in the next section of the Cassandra architecture tutorial.

Clients — Integrate with Hadoop.

Introduction

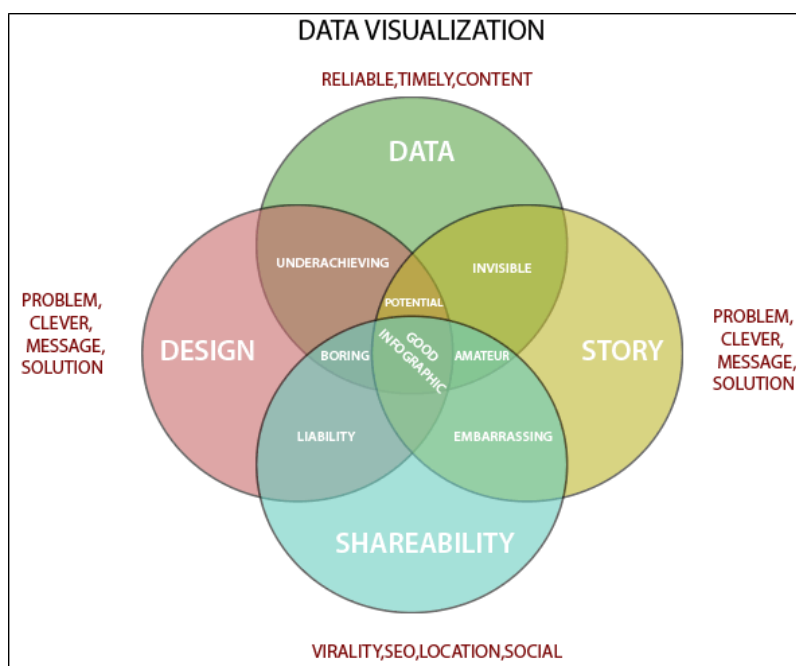
Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

Additionally, it provides an excellent way for employees or business owners to present data to non-technical audiences without confusion.

In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.

Importance of Effective Data Visualization

Effective data visualization is created by communication, data science, and design collide. Data visualizations did right key insights into complicated data sets into meaningful and natural.



Data visualization is important because of the processing of information in human brains. Using graphs and charts to visualize a large amount of the complex data sets is more comfortable in comparison to studying the spreadsheet and reports.

Data visualization is an easy and quick way to convey concepts universally.

- Data visualization can identify areas that need improvement or modifications.
- Data visualization can clarify which factor influence customer behavior.
- Data visualization helps you to understand which products to place where.
- Data visualization can predict sales volumes.

Data visualization tools have been necessary for democratizing data, analytics, and making data-driven perception available to workers throughout an organization. They are easy to operate in comparison to earlier versions of BI software or traditional statistical analysis

software. This guide to a rise in lines of business implementing data visualization tools on their own, without support from IT.

Introduction to Tableau

Tableau is the fastly growing and powerful data visualization tool. Tableau is a business intelligence tool which helps us to analyze the raw data in the form of the visual manner; it may be a graph, report, etc.

Example: - If you have any data like Big Data, Hadoop, SQL, or any cloud data and if you want to analyze that given data in the form of pictorial representation of data, you can use Tableau.

Data analysis is very fast with Tableau, and the visualizations created are in the form of worksheets and dashboards. Any professional can understand the data created using Tableau.

Tableau software doesn't require any technical or any programming skills to operate. Tableau is easy and fast for creating visual dashboards.

Here are some reasons to use Tableau:

- Ultimate skill for Data Science
- User-Friendly
- Apply to any Business
- Fast and Easy
- You don't need to do any Coding
- Community is Huge
- Hold the power of data
- It makes it easier to understand and explain the Data Reports

Features of Tableau

Data Blending: Data blending is the most important feature in Tableau. It is used when we combine related data from multiple data sources, which you want to analyze together in a single view, and represent in the form of a graph.

Example: Assume, we have Sales data in relational database and Sales Target data in an Excel sheet. Now, we have to compare actual sales with target sales, and blend the data based on common dimensions to get access. The two sources which are involved in data blending referred to as primary data and secondary data sources. A left join will be created between the primary data source and the secondary data source with all the data rows from primary and matching data rows from secondary data source to blend the data.

Real-time analysis: Real-Time Analysis makes users able to quickly understand and analyze dynamic data, when the Velocity is high, and real-time analysis of data is complicated. Tableau can help extract valuable information from fast moving data with interactive analytics.

The Collaboration of data: Data analysis is not isolating task. That's why Tableau is built for collaboration. Team members can share data, make follow up queries, and forward easy-to-digest visualizations to others who could gain value from the data. Making sure everyone understands the data and can make informed decisions is critical to success.

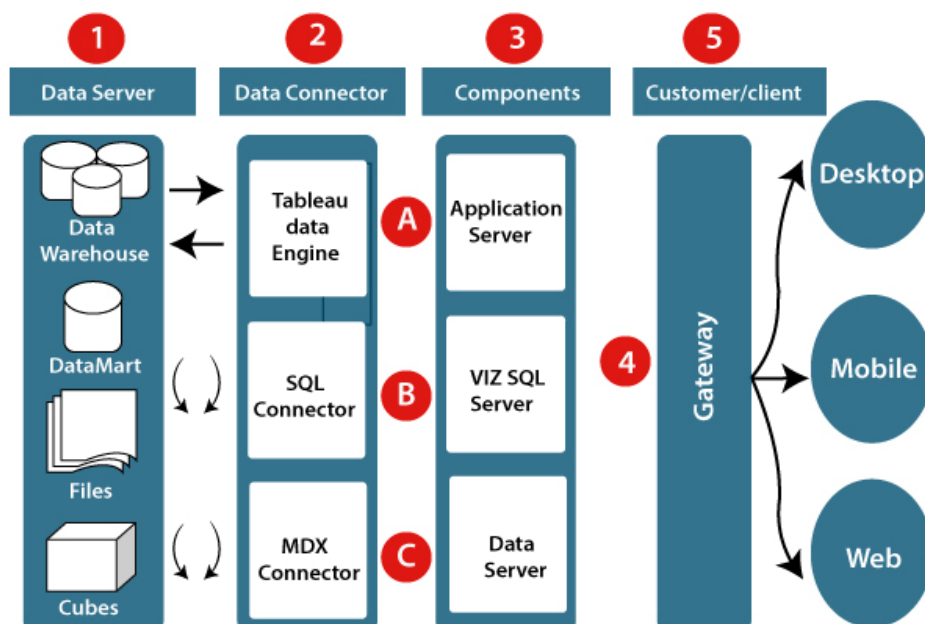
Tableau Architecture

Tableau Server is designed to connect many data tiers. It can connect clients from Mobile, Web, and Desktop. Tableau Desktop is a powerful data visualization tool. It is very secure and highly available.

It can run on both the physical machines and virtual machines. It is a multi-process, multi-user, and multi-threaded system.

Providing such powerful features requires unique architecture.

The different layers used in Tableau server are given in the following architecture diagram:-



1. Data server:- The primary component of Tableau Architecture is the Data sources which can connect to it.

Tableau can connect with multiple data sources. It can blend the data from various data sources. It can connect to an excel file, database, and a web application at the same time. It can also make the relationship between different types of data sources.

2. Data connector:- The Data Connectors provide an interface to connect external data sources with the Tableau Data Server.

Tableau has in-built SQL/ODBC connector. This ODBC Connector can be connected with any databases without using their native connector. Tableau desktop has an option to select both extract and live data. On the uses basis, one can be easily switched between live and extracted data.

- Real-time data or live connection: Tableau can be connected with real data by linking to the external database directly. It uses the infrastructure existing database by sending dynamic multidimensional expressions (MDX) and SQL statements. This feature can be used as a linking between the live data and Tableau rather than importing the data. It makes optimized and a fast database system. Mostly in other enterprises, the size of the database is large, and it is updated periodically. In these cases, Tableau works as a front-end visualization tool by connecting with the live data.
- Extracted or in-memory data: Tableau is an option to extract the data from external data sources. We make a local copy in the form of Tableau extract file. It can remove millions of records in the Tableau data engine with a single click. Tableau's data engine uses storage such as ROM, RAM, and cache memory to process and store data. Using filters, Tableau can extract a few records from a large dataset. This improves performance, especially when we are working on massive datasets. Extracted data allows the users to visualize the data offline, without connecting to the data source.

3. Components of Tableau server: Different types of components of the Tableau server are:

- Application server
- VizQL server
- Data server

A. Application server: The application server is used to provide the authorizations and authentications. It handles the permission and administration for mobile and web interfaces. It gives a guarantee of security by recording each session id on Tableau Server. The administrator is configuring the default timeout of the session in the server.

B. VizQL server: VizQL server is used to convert the queries from the data source into visualizations. Once the client request is forwarded to the VizQL process, it sends the query directly to the data source retrieves information in the form of images. This visualization or image is presented for the users. Tableau server creates a cache of visualization to reduce the load time. The cache can be shared between many users who have permission to view the visualization.

C. Data server: Data server is used to store and manage the data from external data sources. It is a central data management system. It provides data security, metadata management, data connection, driver requirements, and data storage. It stores the related details of data set like calculated fields, metadata, groups, sets, and parameters. The data source can extract the data as well as make live connections with external data sources.

4. Gateway: The gateway directed the requests from users to Tableau components. When the client sends a request, it is forwarded to the external load balancer for processing. The gateway works as a distributor of processes to different components. In case of absence of external load balancer, the gateway also works as a load balancer. For single server configuration, one gateway or primary server manages all the processes. For multiple server configurations, one physical system works as a primary server, and others are used as worker servers. Only one machine is used as a primary server in Tableau Server environment.

5. Clients: The visualizations and dashboards in Tableau server can be edited and viewed using different clients. Clients are a web browser, mobile applications, and Tableau Desktop.

- Web Browser: Web browsers like Google Chrome, Safari, and Firefox support the Tableau server. The visualization and contents in the dashboard can be edited by using these web browser.
- Mobile Application: The dashboard from the server can be interactively visualized using mobile application and browser. It is used to edit and view the contents in the workbook.
- Tableau Desktop: Tableau desktop is a business analytics tool. It is used to view, create, and publish the dashboard in Tableau server. Users can access the various data source and build visualization in Tableau desktop.

- Choosing the Right Chart Type - Using the Color Effectively Reducing Clutter - Dashboard Creation and Formatting

