

Introduction to MongoDB

Objective

- What is NoSQL?
- Architecture of NoSQL:
Sharding and Replica Sets
- Taxonomy of NoSQL
- CAP Theorem
- NoSQL vs RDBMS
- Benefits and Drawbacks
of NoSQL
- NoSQL BASE

Introduction to NoSQL

- NoSQL stands for Not only SQL or Non relational databases that provides a mechanism for storing and retrieval of data.
- NoSQL databases are used in real-time web applications and big – data driven applications.
- The data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL.

Issues with RDBMS

- Hard to Scale
- Resource Intensive

Relation Database

Scale Vertically

Vertical scaling is possible to some extent

NoSQL Databases

Scale Horizontally

Horizontal scaling has no limits

Architecture of NoSQL.

Sharding

- MongoDB achieves scaling through a technique known as “sharding”.
- It is the process of writing data across different servers to distribute the read and write load and data storage requirements.

Replication

- Replication serves as a very important feature in order to protect data loss from one server, increase the availability of data, and give protection from failures.

Classification of NoSQL

NoSQL has major 4 types

- Document Databases
- Key-Value store Databases
- Column-Oriented Databases
- Graph Databases

Document Databases

- Document stores are arguably the most popular of NoSQL databases.
- A document database stores data in JSON, BSON, or XML documents.
- Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.
- In a document database, documents can be nested. Particular elements can be indexed for faster querying.

Document Databases

```
{
  first_name: "Mary",
  last_name: "Jones",
  cell: "516-555-2048",
  city: "Long Island",
  year_of_birth: 1986,
  location: {
    type: "Point",
    coordinates: [-73.9876, 40.7574]
  },
  profession: ["Developer", "Engineer"],
  apps: [
    { name: "MyApp",
      version: 1.0.4 },
    { name: "DocFinder",
      version: 2.5.7 }
  ],
  cars: [
    { make: "Bentley",
      year: 1973 },
    { make: "Rolls Royce",
      year: 1965 }
  ]
}
```


Key Value Store Database

- 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.

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

Column Oriented Database

- Column databases store each column separately, allowing for quicker scans.
- In a column-oriented database it's easy to add another column because none of the existing columns are affected by it.

Graph Database

- The last big NoSQL database type is the most complex one, geared toward storing relations between entities in an efficient manner.
- When the data is highly interconnected, such as for social networks, graph databases are the answer.

CAP Theorem

Three distributed system characteristics to which the CAP theorem refers.

- **Consistency** - The data should remain consistent even after the execution of an operation.

This means once data is written, any future read request should contain that data.

- **Availability** – The data should always be available and responsive. It should have no downtime.

- **Partition Tolerance** - Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

NOTE: A *partition* is a communications break within a distributed system

CAP Theorem

Out of three guarantees, no system can provide more than 2 guarantees. Since in the case of a distributed systems, the partitioning of the network is must, the tradeoff is always between consistency and availability.

- **CP database:** A CP database delivers consistency and partition tolerance at the expense of availability.
- **AP database:** An AP database delivers availability and partition tolerance at the expense of consistency.
- **CA database:** A CA database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

SQL VS NoSQL

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP (consistency, availability, partition tolerance)

Why do NoSQL databases scale so well?

- They do not have any costly relationships. Every Item in database stand its own.
- This further tells how data is stored, in Key – Value stores. Each item in database has two fields **KEY** and **VALUE**

KEY	VALUE
19876567	"Macbook Air"
1983452	{ name: "Iphone 12", price: 500000, description: "Mobile" }

Because of this simple design NoSQL databases Scale better

Advantages of NoSQL

- Elastic scalability
- Big data applications
- Economy

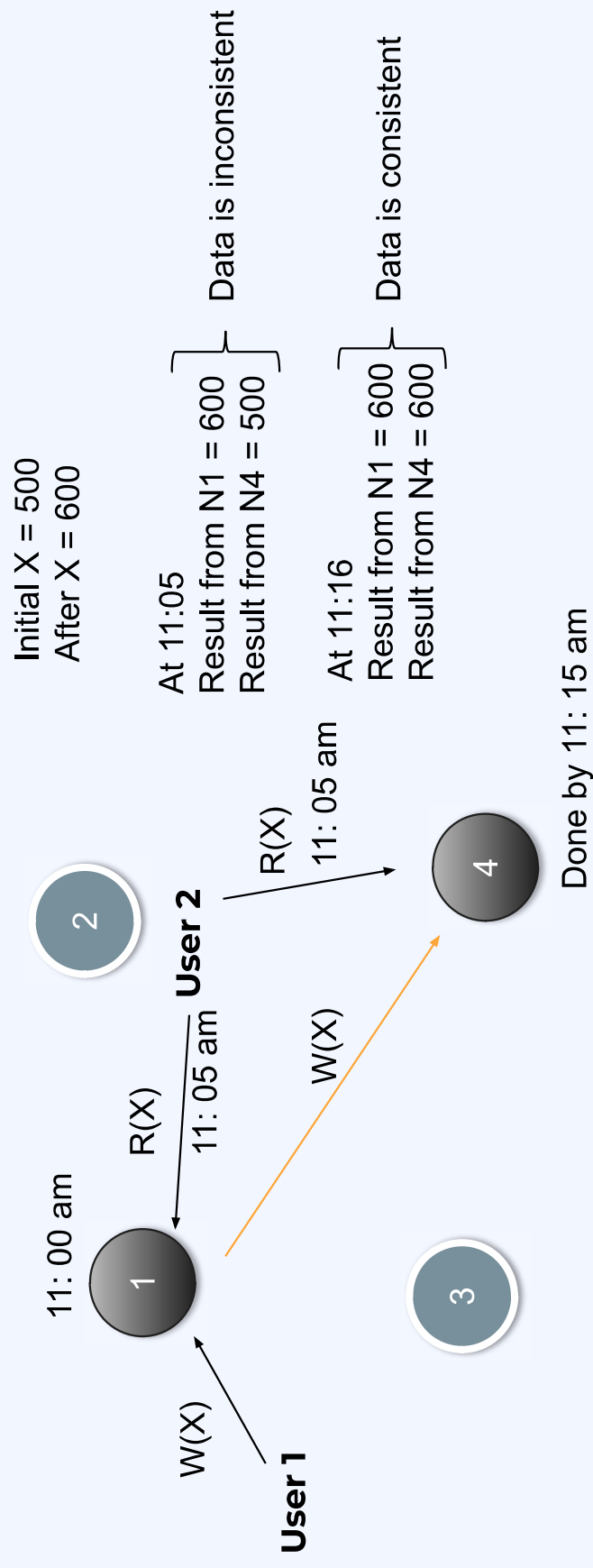
NoSQL BASE

B -> Basically Available - Basically, available means DB is available all the time as per CAP theorem

S -> Soft State - Soft state means even without an input, the system state may change

E -> Eventual Consistency - Eventual consistency means that the system will become consistent over time

NosQL BASE – Eventual Consistency



Introduction to MongoDB

- **MongoDB** is a document-oriented NoSQL database used for high volume data storage
- Instead of using tables and rows as in the 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 which is the equivalent of relational database tables.

Introduction to MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key provided by itself)

Features of MongoDB

- 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.
- 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.
- 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.
- The data model available within MongoDB allows you to represent hierarchical relationships to store arrays, and other more complex structures more easily.

Key Components of MongoDB Architecture

- **_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.
- **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 MS SQL.
- **Cursor** – This is a pointer to the result of a query. Clients can iterate through a cursor to retrieve results.
- **Database** - This is a container for collections like in RDMS wherein it is a container for tables.
- **Document** - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
- **JSON** – This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data.

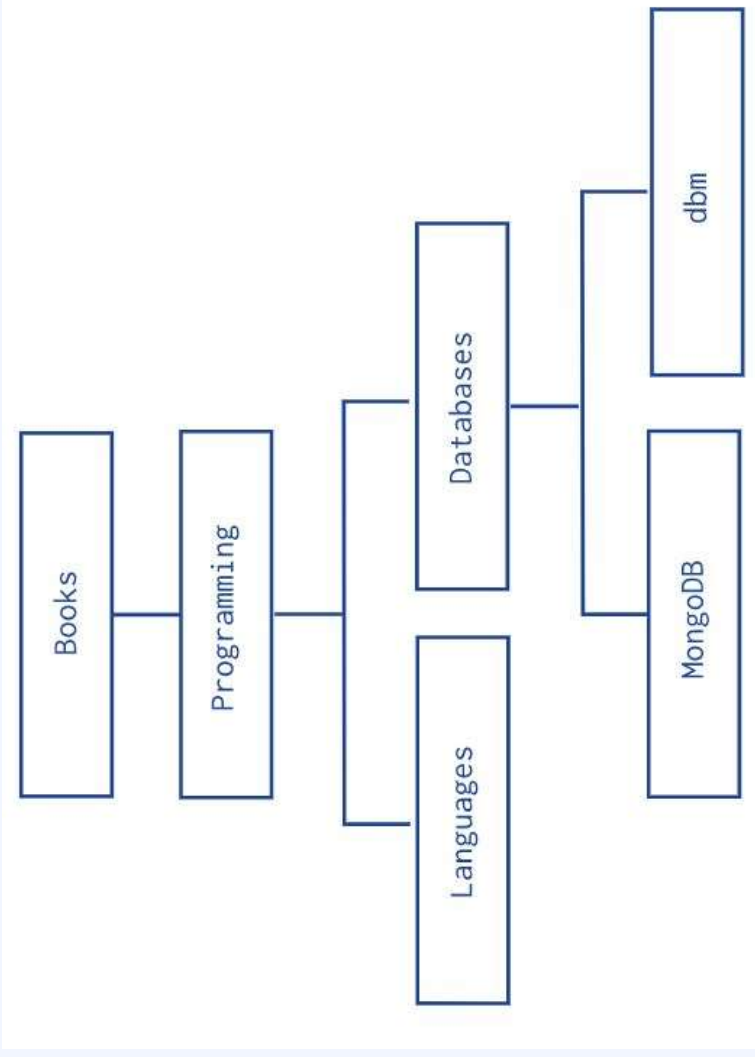
Key Components of MongoDB Architecture

- **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.
- **Replication** - MongoDB can provide high availability with replica sets
- **Indexing** - Indexes can be created to improve the performance of searches within MongoDB.
- **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.

MongoDB: CAP Approach

- Relative to the CAP theorem, MongoDB is a **CP** data store—it resolves network partitions by maintaining consistency, while compromising on availability.
- MongoDB is strongly consistent by default - if you do a write and then do a read, assuming the write was successful you will always be able to read the result of the write. This is because MongoDB is a single-master system and all reads go to the primary by default.
- When the primary node becomes unavailable, the secondary node with the most recent operation log will be elected as the new primary node

MongoDB: Hierarchical Objects



MongoDB: Hierarchical Objects

MongoDB allows various ways to use tree data structures to model large hierarchical or nested data relationships

- **Model Tree Structures with Parent References** - Presents a data model that organizes documents in a tree-like structure by storing references to "parent" nodes in "child" nodes.
- **Model Tree Structures with Child References** - Presents a data model that organizes documents in a tree-like structure by storing references to "child" nodes in "parent" nodes.
- **Model Tree Structures with an Array of Ancestors** - Presents a data model that organizes documents in a tree-like structure by storing references to "parent" nodes and an array that stores all ancestors.
- **Model Tree Structures with Materialized Paths** - Presents a data model that organizes documents in a tree-like structure by storing full relationship paths between documents.

Designs of MongoDB

MongoDB provides two types of data models: — [Embedded data model](#) and [Normalized data model](#).

Embedded Data Model

In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model.

```
{
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details:{
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
  },
  Contact: {
    e-mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
  },
  Address: {
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana"
  }
}
```

Designs of MongoDB

Normalized Data Model

In this model, you can refer the sub documents in the original document, using references. For example, you can re-write the above document in the normalized model as

```
Employee:
{
    _id: <ObjectId101>,
    Emp_ID: "10025AE336"
}
Personal_detail:
{
    _id: <ObjectId102>,
    empDocID: " ObjectId101",
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
}
```

Designs of MongoDB

```
Contact:
{
  _id: <ObjectId103>,
  empDocID: " ObjectId101",
  e-mail: "radhika_sharma.123@gmail.com",
  phone: "9848022338"
}
Address:
{
  _id: <ObjectId104>,
  empDocID: " ObjectId101",
  city: "Hyderabad",
  Area: "Madapur",
  State: "Telangana"
}
```

Designs of MongoDB

```
Contact:
{
  _id: <ObjectId103>,
  empDocID: " ObjectId101",
  e-mail: "radhika_sharma.123@gmail.com",
  phone: "9848022338"
}
Address:
{
  _id: <ObjectId104>,
  empDocID: " ObjectId101",
  city: "Hyderabad",
  Area: "Madapur",
  State: "Telangana"
}
```

JSON & BSON Format

JavaScript Object Notation (JSON) is a standard file format that uses human type readable text to transmit data with attribute-value pairs and array data types.

BSON (Binary JSON), on the other hand, is a computer interchange format that is mainly used for data storage and as a network transfer format in the MongoDB database. BSON is a binary version of JSON which adds two things to the mix, lengths and types. MongoDB stores data in BSON format

JSON & BSON Format

JSON	BSON
Standard file format	Binary file format
Comparatively less fast	Faster
Consumes less space	More space is consumed
Transmission of data	Storage of data
No encoding is done	Faster encoding and decoding
JSON format needs not be parsed as they are in human readable format already.	BSON needs to be parsed as they are easy for machines

Schema Flexibility

- Firstly, JSON documents are polymorphic – fields can vary from document to document within a single collection
- Secondly, there is no need to declare the structure of documents to the database – documents are self-describing.
- Thirdly, if a new field needs to be added to a document, it can be created without affecting all other documents in the collection, without updating a central system catalog and without taking the database offline.

Features of MongoDB

- 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.
- 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.
- The data model available within MongoDB allows you to represent hierarchical relationships to store arrays, and other more complex structures more easily.

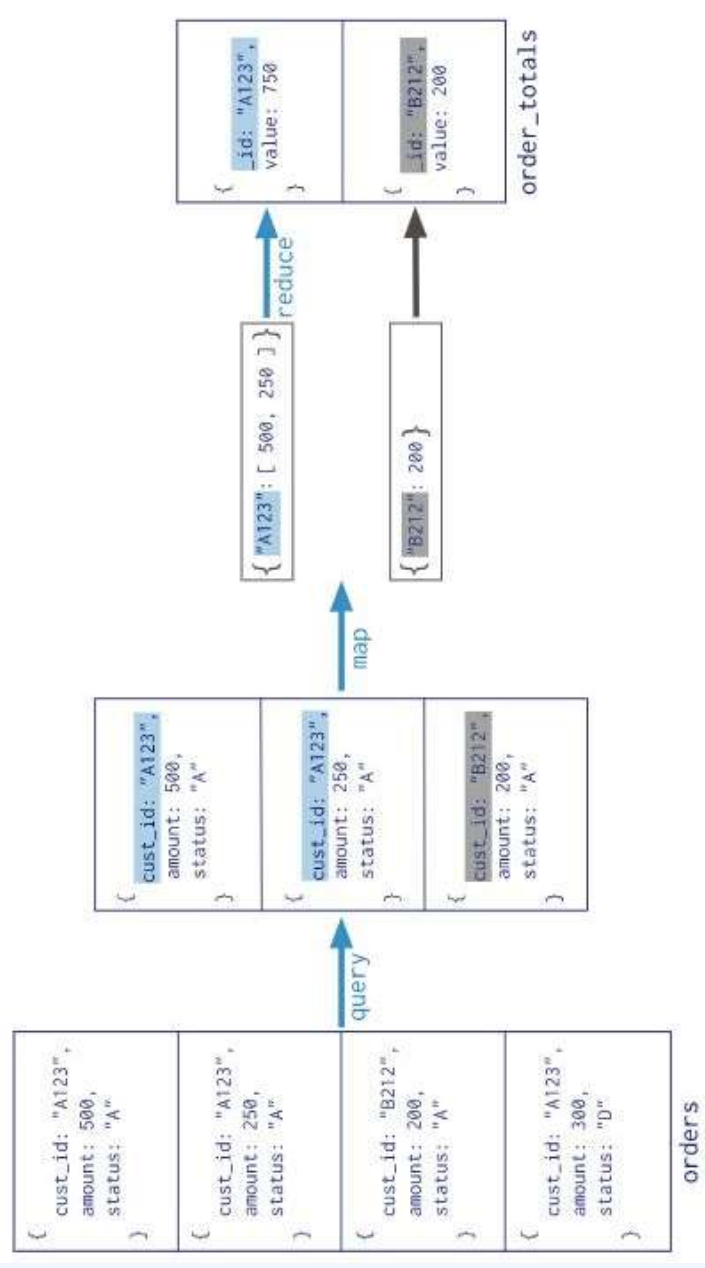
Features of MongoDB

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

Syntax:

```
db.collection.mapReduce (
  function() {emit(key,value);}, //map function
  function(key,value) {return reduceFunction}, { //reduce function
    out: collection,
    query: document,
    sort: document,
    limit: number
  }
)
```

Features of MongoDB



Datatypes in MongoDB

MongoDB supports many datatypes

String

Integer

Boolean

Double- This Type is used to store floating point values

Min/Max keys

Arrays

Timestamp

Object

Null

Symbol - This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

ObjectId

Create Documents

InsertOne

Syntax:

```
db.inventory.insertOne(  
  { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }  
)
```

InsertMany

Syntax:

```
db.inventory.insertMany([  
  { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },  
  { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },  
  { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }  
)
```

Find Documents

Find All Documents in a Collection

```
db.collection.find()
```

Query for Equality

```
db.bios.find( { _id: 5 } )
```

The following operation returns documents in the collection where the field last in the name embedded document equals "Hopper":

```
db.bios.find( { "name.last": "Hopper" } )
```

Note: To access fields in an embedded document, use dot notation ("**<embedded document>.<field>**").

Find Documents

Query Using Operators

The following operation uses the **\$in** operator to return documents in the bios collection where _id equals either 5 or ObjectId("507c35dd8fada716c89d0013"):

```
db.bios.find(  
  { _id: { $in: [ 5, ObjectId("507c35dd8fada716c89d0013") ] } }  
)
```

The following operation uses the **\$gt** operator returns all the documents from the bios collection where birth is greater than new Date('1950-01-01'):

```
db.bios.find( { birth: { $gt: new Date('1950-01-01') } } )
```


Find Documents

The following operation uses the **\$regex** operator to return documents in the bios collection where name.last field starts with the letter N (or is "LIKE N%")

```
db.collection.find( { "name.last": { $regex: /^N/ } } )
```

Query for Ranges

```
db.collection.find( { birth: { $gt: new Date('1940-01-01'), $lt: new Date('1960-01-01') } } )
```

CRUD-Update

- **Update one collection**

```
db.books.update(  
  { _id: 1 },  
  {  
    $set: {  
      item: "ABC123",  
    }  
  }  
)
```

CRUD-Update

- **Update multiple collection**

```
db.books.update(  
  { _id: 1 },  
  {  
    $set: {  
      item: "ABC123",  
    }  
  },  
  { multi: true }  
)
```

CRUD-Update

- **Find one and Update collection**

```
db.books.findOneAndUpdate(  
  { _id: 1 },  
  {  
    $set: {  
      item: "ABC123",  
    }  
  },  
  { upsert: true, returnNewDocument: true }  
)
```

CRUD- Delete

- DeleteOne

```
db.orders.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } );
```

- DeleteMany

```
db.orders.deleteMany( { "client" : "Crude Traders Inc." } );
```

Indexing Functionality

- Indexes provide users with an efficient way of querying data. When querying data without indexes, the query will have to search for all the records within a database to find data that match the query.
- In MongoDB, querying without indexes is called a collection scan. A collection scan will:
 - Result in various performance bottlenecks
 - Significantly slow down your application
- **INDEX** - Indexes are special data structures that store a small part of the Collection's data in a way that can be queried easily.
- In MongoDB, indexes are defined in the collection level and indexes on any field or subfield of the documents in a collection are supported.

Indexing Functionality

When creating documents in a collection, MongoDB creates a unique index using the `_id` field.

MongoDB refers to this as the **Default `_id` Index**.

SYNTAX : **`db.<collection>.createIndex(<Key and Index Type>, <Options>)`**

When creating an index, you need to define the field to be indexed and the direction of the key (1 or

-1) to indicate ascending or descending order.

GET all indexes in collection

SYNTAX : **`db.<collection>.getIndexes()`**

DROP an index

SYNTAX: **`db.<collection>.dropIndex(<Index Name / Field Name>)`**

MongoDB index types

Single Field Index

These user-defined indexes use a single field in a document to create an index in an ascending or descending sort order (1 or -1).

Compound Index

You can use multiple fields in a MongoDB document to create a compound index. This type of index will use the first field for the initial sort and then sort by the preceding fields.

Multikey Index

MongoDB supports indexing array fields. When you create an index for a field containing an array, MongoDB will create separate index entries for every element in the array

Aggregation Functionality

\$Group - With the \$group() stage, we can perform all the aggregation or summary queries that we need, such as finding counts, totals, averages or maximums.

```
db.universities.aggregate([  
  { $group : { _id : '$name', totaldocs : { $sum : 1 } } }  
]).pretty()
```

\$out - it allows you to carry the results of your aggregation over into a new collection.

```
db.universities.aggregate([  
  { $group : { _id : '$name', totaldocs : { $sum : 1 } } },  
  { $out : 'aggResults' }  
])
```

Aggregation Functionality

\$sort - Need \$sort() stage to sort results by the value of a specific field.

```
db.universities.aggregate([
  { $match : { name : 'USAL' } },
  { $project : { _id : 0, 'students.year' : 1, 'students.number' : 1 } },
  { $sort : { 'students.number' : -1 } }
```

Aggregation Functionality

```
$lookup
{
  $lookup:
  {
    from: <collection to join>,
    localField: <field from the input documents>,
    foreignField: <field from the documents of the "from" collection>,
    as: <output array field>
  }
}
```

MapReduce Functionality

- Map-reduce is a data processing programming model that helps to perform operations on large data sets and produce aggregated results.
- This function has two main functions, i.e., map function and reduce function.
- The map function is used to group all the data based on the key-value and the reduce function is used to perform operations on the mapped data.
- The data is independently mapped and reduced in different spaces and then combined together in the function and the result will save to the specified new collection.

MapReduce Functionality

```
db.collection.mapReduce(  
  function() {emit(key,value);}, // map function  
  function(key,values) {return reduceFunction}, // reduce function  
  { out: collection }  
)
```

What is Atlas?

The core of MongoDB Cloud is MongoDB Atlas, a fully managed cloud database for modern applications.

Atlas handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice

Benefits of Atlas

One of the most convenient parts about MongoDB Atlas is that it is available on-demand through a pay-as-you-go model and billed on an hourly basis

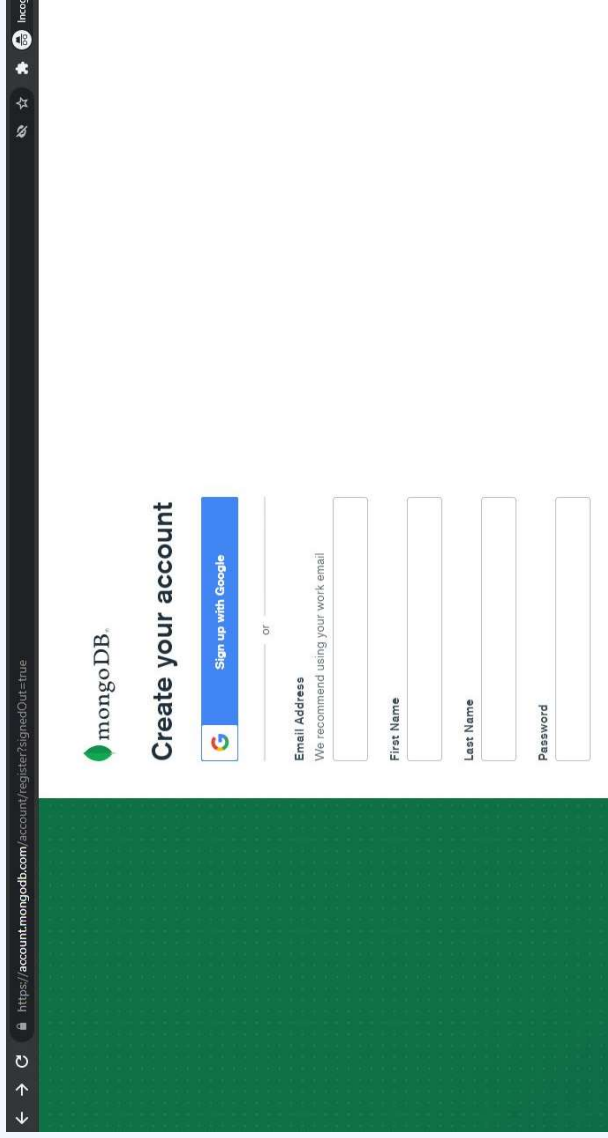
Automated Security Features — MongoDB Atlas makes sure you know who's got eyes on your data and helps you keep all others out.

Built-In Replication — Their platform provides you with multiple servers for always-on availability, to make sure you're up, even when your primary master is down.

Monitoring & Alerts — You get instant visibility into the database and hardware metrics that matter, so you stay ahead of any issues that could impact performance and user experience.

Scalability - MongoDB Atlas grows with you, all with the click of a button. You can scale up across a range of instance sizes, and scale-out with MongoDB Atlas' "automatic sharding"

Create Account on Atlas



The screenshot shows the MongoDB Atlas registration page. The browser's address bar displays the URL: `https://account.mongodb.com/account/register?signedOut=true`. The page features a green header bar on the left. The main content area has a white background with the MongoDB logo and the heading "Create your account". Below the heading, there is a "Sign up with Google" button. A horizontal line with the word "or" in the center separates this from the email registration section. The email section is titled "Email Address" and includes the text "We recommend using your work email". It contains four input fields: "Email Address", "First Name", "Last Name", and "Password".

Create Account on Atlas



Create Account on Atlas

← → 🔒 <https://cloud.mongodb.com/v2/60e81976cc7bd8652986d5ab#setup/onboarding>

Let's get your account set up

Name your organization and project

Organization
Your organization can be a business, team, or an individual

My Organization

Project Name
Use projects to isolate different environments (development/testing/production)

My Project Name

What is your preferred language?

We'll use this to customize code samples and content we share with you. You can always change this later.

<input checked="" type="radio"/> JavaScript	<input type="radio"/> C++	<input type="radio"/> C# / .NET	<input type="radio"/> Go
<input type="radio"/> Java	<input type="radio"/> C	<input type="radio"/> Perl	<input type="radio"/> PHP
<input type="radio"/> Python	<input type="radio"/> Ruby	<input type="radio"/> Scala	<input type="radio"/> Other

[Skip](#) [Continue](#)

Create Account on Atlas

MONGODB ATLAS

Choose a path. Adjust anytime.

Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud

Dedicated Multi-Cloud & Multi-Region Clusters

For teams developing world-class applications that require multi-region resiliency or ultra-low latency.

- ✓ Includes all features from Shared and Dedicated Clusters
- ✓ Replicate data across clouds and regions
- ✓ Globally distributed read and write operations
- ✓ Control data residency at the document level

Create a cluster

Starting at
\$0.13/hr*
*estimated cost \$95.86/month

Dedicated Clusters

For teams building applications that need advanced development and production-ready environments.

- ✓ Includes all features from Shared Clusters
- ✓ Auto-scaling
- ✓ Network isolation
- ✓ Realtime performance metrics

Create a cluster

Starting at
\$0.08/hr*
*estimated cost \$68.64/month

Shared Clusters

For teams learning MongoDB or developing small applications.

- ✓ Highly available auto-healing cluster
- ✓ End-to-end encryption
- ✓ Role-based access control

Create a cluster

Starting at
FREE

Create Account on Atlas

Cloud Provider & Region

aws

Google Cloud

Azure

AWS, Mumbai (ap-south-1)

★ Recommended region

NORTH AMERICA

★ N. Virginia (us-east-1)

★ Oregon (us-west-2)

EUROPE

★ Frankfurt (eu-central-1)

★ Ireland (eu-west-1)

ASIA

★ Singapore (ap-southeast-1)

Mumbai (ap-south-1)

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with 3 data bearing servers.

Shared Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	Free forever

M0 clusters are best for getting started, and are not suitable for production environments.

500 max connections | Low network performance | 100 max databases | 500 max collections

M2

Shared

2 GB

Shared

\$8 / MONTH

M5

Shared

5 GB

Shared

\$25 / MONTH

EdYoda - Developer Programs

<https://www.edyoda.com>

Create Account on Atlas

Cluster Name

One time only: once your cluster is created, you won't be able to change its name.

MongoCluster ▼

Cluster names can only contain ASCII letters, numbers, and hyphens.

Create Account on Atlas



Create Account on Atlas



Create Account on Atlas

Connect to MongoDB

Setup connection security

Choose a connection method

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

You can't connect yet. Set up your firewall access and user security permission below.

1 Add a connection IP address

Add Your Current IP Address

Add a Different IP Address

Allow Access from Anywhere

2 Create a Database User

This first user will have `atlasAdmin` permissions for this project.
Keep your credentials handy, you'll need them for the next step.

Username

etx_dbUser

Password

Autogenerate Secure Password

etx_dbUserPassword

SHOW

Create Database User

Close

Choose a connection method

EdYoda - Developer Programs

<https://www.edyoda.com>

Create Account on Atlas

Connect to MongoDB

Setup connection security

Choose a connection method

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

You can't connect yet. Set up your firewall access and user security permission below.

1

Create a connection IP address

IP Address

40.36.190.88

Description (Optional)

An optional comment describing this entry

Cancel

Add IP Address

2

Create a Database User

This first user will have atlasAdmin permissions for this project.
Keep your credentials handy, you'll need them for the next step.

Username

ex. dbUser

Password

ex. dbUserPassword

Generate Secure Password

SHOW

Create Database User

Close

Choose a connection method

2

Create a Database User

This first user will have atlasAdmin permissions for this project.
Keep your credentials handy, you'll need them for the next step.

Username

tushar

Password

Generate Secure Password

SHOW

Create Database User

EdYoda - Developer Programs

<https://www.edyoda.com>

Create Account on Atlas

Connect to MongoCluster


✓ Setup connection security

Choose a connection method


Connect

Choose a connection method [View documentation](#)


Get your pre-formatted connection string by selecting your tool below.



Connect with the **mongo shell**
Interact with your cluster using MongoDB's interactive Javascript interface



Connect your application
Connect your application to your cluster using MongoDB's native drivers



Connect using **MongoDB Compass**
Explore, modify, and visualize your data with MongoDB's GUI

Go Back

Close