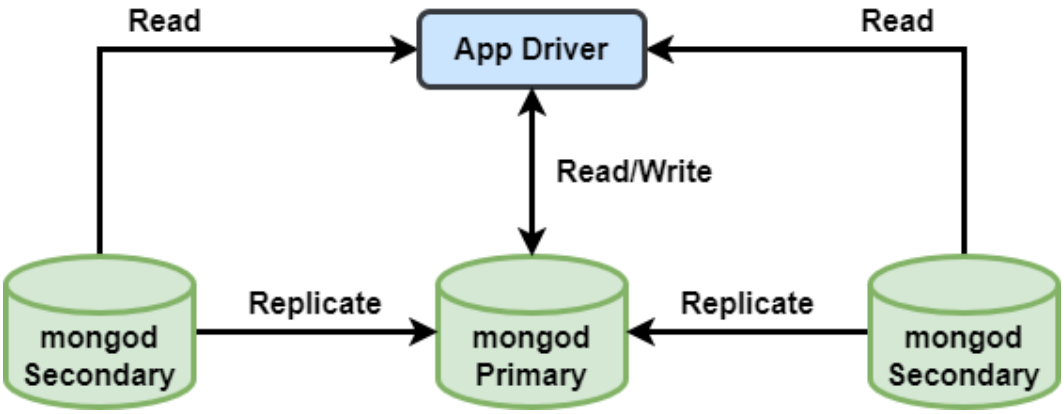
MongoDB

- **Overview**
  - Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database
  - MongoDB is a NoSQL database open source, document-oriented database.
  - Unlike RDBMS where data is stored in tables, MongoDB stores data in JSON-like documents with a dynamic schema.
  - Simple query language: rich document-based queries.
  - Scalability – The MongoDB environments are very scalable.

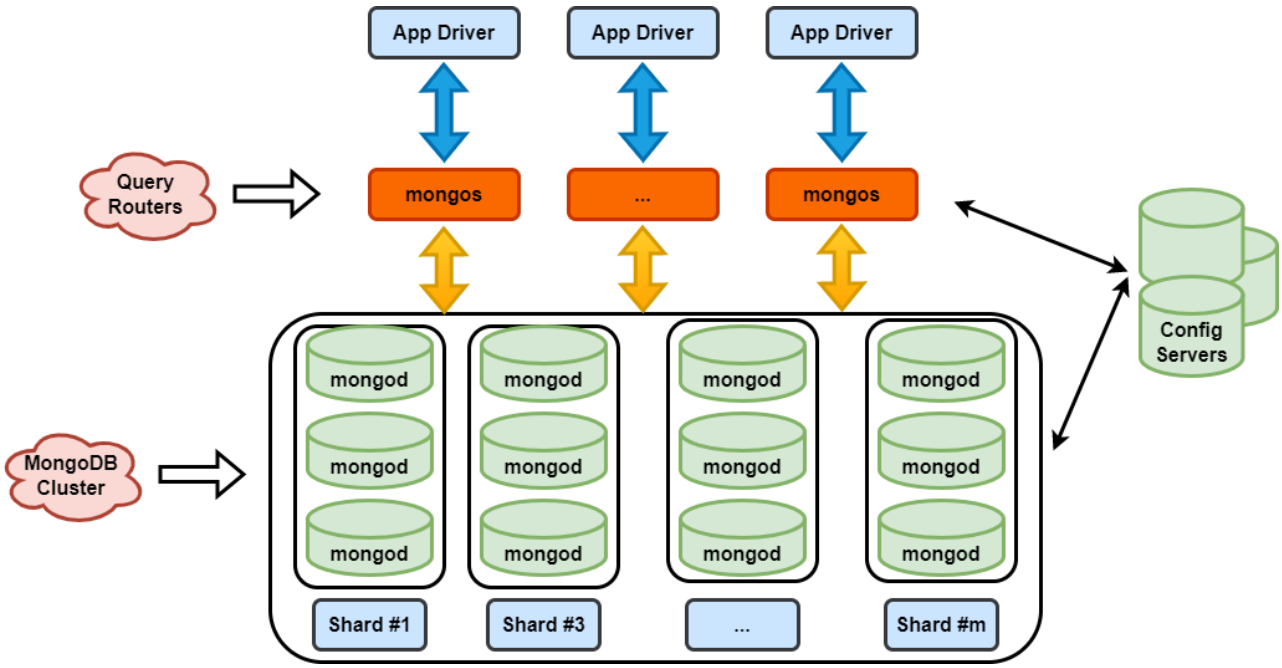
- **Features**
  - Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other
  - The document structure is more in line with how developers construct their 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.

- **MongoDB Architecture**
  - **Mongos Shard Deamons** : It is a daemon that route requests to MongoDB data servers
  - MongoDB Configuration servers: they store the sharding configuration
  - Three types of development environment:
    - Standalone: The application connects to a single server(Mongod).
    - Replica Set: The application connects to a cluster or servers one being the primary and the other secondary/arbiters.
    - Sharded replica set: The application connects to as set of replica sets called shards via request router (mongos).

- **Replica Set:**
  - Replication provides redundancy and increases data availability.
  - It provides a high fault tolerance.
  - Replication helps in avoiding performance bottlenecks and avoiding single point of failures
  - The smallest Replica Set comes with one primary and two secondary servers.
  - On failure of a primary server, a secondary server becomes a primary server.



- **Sharding:**
  - Sharding is a method for distributing data across multiple servers.
  - Sharding is required to big data.
  - Sharding is done using a shard key, record id.
  - Recharding method changes the shard key for a collection and changes the distribution of your data.
  - MongoDB sharded data using the following components:
    - **Shard**:
      - Each shard contains a subset of sharded data. Each shard is deployed a Replica Set.
    - **Mongos**:
      - It acts as a query router, providing an interface between client applications and the shared cluster.
    - **Config Servers**:
      - They store metadata and configuration settings for cluster. acts as query router, providing an interface between client applications and the shared cluster.



- **How does sharding work?**
  - There are two sharding strategies:
    - Algorithmic sharding or hashed sharding
    - Ranged/dynamic sharding
    - Entity-/relationship-based sharding
    - Geography-based sharding
  - **Ranged/dynamic sharding:**
    - It is a simple and an easy-to-understand strategy of horizontal partitioning,
    - It takes a field on the record as an input and, based on a predefined range, allocates that record to the appropriate shard.
    - It requires a lookup table or service available for all queries or writes.
    - For example, consider a set of data with IDs that range from 0-50. A simple lookup table might look like the following:

Range	Shard ID
0-199	A
200-399	B
400-599	C
600-799	D
800-999	E

    - The field on which the range is based is also known as the shard key.
    - A poor choice of shard key will lead to unbalanced shards, which leads to decreased performance.
    - An effective shard key will allow for queries to be targeted to a minimum number of shards.
      - Query record with IDs between 450 and 550 ➔ Only involves shards C and D.
    - Drawback: The lookup table may be a bottleneck.

- **Algorithmic/hashed sharding:**
  - It takes a record as an input and applies a hash function or algorithm to it which generates an output or hash value.
  - The has value is then used to allocate each record to the appropriate shard.
  - Hash function example: H(Record)=ID % Number of Shards
  - Hashing the inputs allows more even distribution across shards even when there is not a suitable shard key
  - No lookup table needs to be maintained.
  - Drawbacks:
    - A query output may reside on multiple shards.
    - Re-sharding can be expensive: it requires rebalancing all shards to move around records.
- **Entity-/relationship-based sharding:**
  - It keeps related data together on a single physical shard.
  - Example: Users and their transactions in shopping carts can be stored on the same shard.
- **Geography-based sharding:**
  - Geography-based sharding, or geosharding, keeps related data together on a single shard based on their geography.
  - It is similar to range sharding where the key contains geographic information; country filed.

- Advantages of sharding
  - Sharding allows you to horizontally scale your database to handle big data.
  - Increased read/write throughput
  - High availability:
  - A shard is a replica set.
  - In case a shard becomes unavailable, the database as a whole still remains partially functional.
- Disadvantages of sharding:
  - Query overhead:

        - Needs of separate machine or service to route a query to the appropriate shard ➔ additional latency.
    - Complexity of administration:
        - Maintain shards and replicated nodes.
    - Increased infrastructure costs:
        - Sharding requires additional machines and compute power over a single database server.

- **Data Modeling**
    - **Documents:**
        - A single entity of data
        - Embedded documents
        - BSON: field:value pairs
        - Maximum size of 16MB. MongoDB uses GridFS for storing files larger than 16 MB.
            - GridFS is a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB.
        - Depending on your business logic, you may design your documents using one of the following types of document:
            - Embedded documents
            - Reference or link documents
            - Embed and Reference

        - Embedded Documents:
            - Embedded documents are denormalized data models.
            - They are used to store relationships between data in the same main document.
            - Embedded documents should be used when a relationship exists between entities.
            - Advantage: You do not have to query multiple documents to get the desired results
            - Disadvantage: This can lead to redundancy

- References:
  - The relationship between one data to other data is stored in the Reference documents.
  - The relationship between documents may be many to many or one to many.
  - We use an array of id's to reference the details of a given document.
  - Advantage: less or no redundancy data at all
  - Disadvantage: We need to query multiple collections to get the desired result.
- Document Relationships
  - A One-to-One relationship (1:1)
  - A One-to-Many relationship (1:N)
  - A Many-to-Many relationship (N:M)
  - Example:

Professors		
ProfID	Lname	Fname
P01	May	Kate
P02	John	Paul
P03	Omar	Anbar

Departments		
Dept	Name	Phone
D100	CS	111-111-1100
D101	Bio	111-111-1100
D102	ECE	111-111-1100

Students		
StudID	Lname	Fname
S100	Clark	Beth
S101	Smith	Zack
S102	Sue	Yin

Courses	
CourseID	Name
C001	Bio 101
C002	Computing
C003	Statistics
C004	Database

Chairs		
ChairID	Name	Dept
100	Mary	D100
101	Paul	D101

- **One-to-one Relationship:**
  - One chair can manage only one department

Link:

```
Chair ={
        "_id":100,
        "Name": "Mary".
        "Dept": "D100"
}
```

Embedded:

```
Chair ={
    "_id":100,
    "Name": "Mary".
    "Dept": {
            "Dept":"D100",
            "Name":"CS",
            "Phone":" 111-111-1100"
    }
}
```

- **One-to-many Relationships**
  - A professor can teach many courses:

Link:

```
Professor ={
    "_id":"P01",
    "Fname": "Mary".
    "Lname": "Kate",
    "Courses": [C002, C004]
}
```

Embedded:

```
Professor ={
    "_id":"P01",
    "Fname": "Mary".
    "Lname": "Kate",
    "Courses": {
```

```
            "CourseID":"C001",
            "Name":"Bio 101"
        }
        {
            "CourseID":"C004",
            "Name":"Database"
        }
    }
```

- **Many-to-many Relationships:**
  - Many students can take may courses

    Link:

```
students ={
    "_id":"S100",
    "Fname": "Beth".
    "Lname": "Clark",
    "Courses": [C002, C004]
}
Courses ={
    "_id":"C001",
    "Name": "Bio 101".
    "Students": [S100, S101]
}
```

- **Collections:**
  - A collection is a group of documents.
  - Similar to a table in a traditional database table.
  - MongoDB groups collections into databases
  - A single instance of MongoDB can host several databases
  - Each database groups together zero or more collections
  - A database has its own permissions, users and roles and each database is stored in separate files on disk

- **CRUD Operations**
  - They are the basic methods to manage data in a MongoDB database.[https://www.mongodbtutorial.org/]
  - CRUD stands for:
    - **Create** operation is used to insert new documents in the MongoDB database.
    - **Read** operation is used to query a document in the database.
    - **Update** operation is used to modify existing documents in the database.
    - **Delete** operation is used to remove documents in the database.
  - **Create:**
    - MongoDB provides two different create operations to insert documents into a collection:
      - db.collection.insertOne(): It inserts one document into the collection.
      - db.collection.insertMany(): It inserts multiple documents into the collection.
    - db.collection.insertOne(
      <document>,
      { writeConcern: <document>}
      ):
      - <document>: the document you want to insert.
      - writeConcern: is an optional argument that describes the level of acknowledgment requested from MongoDB
    - db.collection.insertMany(
      [document1, document2, ...],
      {
      writeConcern: <document>,
      ordered: <boolean>
      }):
      - <document1, document2, ..>: documents to be inserted
      - writeConcern: same as insertOne() operation

        - ordered: It is a boolean value that determines whether MongoDB should perform an ordered or unordered insert.

- **Read:**
    - MongoDB provides two different methos of reading documents from a collection:
        - db.collection.find():
            - It returns the documents that satisfy a specified condition and returns a cursor to the matching documents.
        - db.collection.findOne():
            - It returns all the documents from a collection.
    - db.collection.find():
        - db.collection.find(query, projection):
            - Query:
                - The search criteria
            - Projection:
                - Specifies the fields in the matching document that you want to return.
                - If you don't pass Projection, it will return all fields in the matching documents.
    - db.collection.findOne():
        - db.collection.findOne(query, projection):
            - same parameters as db.collection.find ()
            - If multiple documents satisfy the query, it returns the first document of the collection.
            - If no documents satisfy the search criteria, it returns null.
- **Update:**
    - MongoDB provides two different methos of updating documents in a collection:
        - updateOne() –update a single document that satisfies a condition.

        - updateMany() –update multiple documents that match a condition.
- **Update:**
    - MongoDB provides two deleting operations:
        - deleteOne() – delete a single document from a collection.
        - deleteMany() – delete all documents that match a condition from a collection.