Mongodb Scalability -   
MongoDB is highly scalable, meaning it can handle increasing amounts of data and user requests without a significant performance impact. This is achieved through two primary scaling strategies: vertical and horizontal scaling

Vertical Scaling:

* Vertical scaling involves increasing the resources of a single server, such as adding more RAM, CPU, or storage.

While this can increase capacity, it has limitations due to the physical constraints of a single machine and cloud-based providers' hardware ceilings.

Horizontal Scaling (Sharding):

* Horizontal scaling involves distributing data and processing across multiple servers.

MongoDB achieves horizontal scaling through its sharding feature, where data is partitioned and distributed across different servers or shards.

This allows MongoDB to handle massive datasets and high traffic loads without sacrificing performance.

Sharding requires careful planning and selection of a shard key that balances the load across different shards  
  
Resilience - "resilient" refers to the ability of a database system to withstand failures and continue operating with minimal data loss and disruption. It's a crucial aspect of building mission-critical applications, ensuring high availability and disaster recovery capabilities.

In essence, a resilient MongoDB deployment ensures that your data is always available, protected, and recoverable, even in the face of unexpected issues

Mission Critical - In MongoDB, "mission critical" refers to applications or systems that are essential for an organization's operations. If these systems fail, it can have a significant and negative impact on the business. MongoDB's features, like high availability and security, are designed to ensure the reliability and protection of data for such mission-critical workloads.

Sharding -

In MongoDB, sharding is a technique used to distribute data across multiple servers (shards) to improve scalability and performance, especially when dealing with large datasets or high user concurrency. It enables horizontal scaling by adding more servers to the cluster, effectively distributing the workload and avoiding a single point of failure.

Types of Sharding in MongoDB:

1. **1. Ranged Sharding:**

Divides data into ranges based on the shard key values.

Each shard stores a specific range of data, defined by the shard key.

Suitable for scenarios where data is naturally grouped by some attribute (e.g., user ID ranges).

1. **2. Hashed Sharding:**

Computes a hash of the shard key field's value.

Each shard stores a portion of the data based on the hash value.

Suitable for scenarios where data needs to be evenly distributed across shards and there's no natural grouping by an attribute

Authentication & Authorization:

In MongoDB, authentication verifies the identity of a client (e.g., a user, application) trying to connect to the database. Authorization, on the other hand, determines what actions that verified client is permitted to perform within the database. In essence, authentication is about "who" is trying to connect, while authorization is about "what" they can do

Authentication – who you are.  
Authorization – What role do you have.

Commands worked on   
  
use shopDB2025

db.createCollection("products")

db.createCollection("orders")

db.products.insertMany([

{ \_id: 1, name: "Laptop", price: 1000, category: "Electronics", stock: 50 },

{ \_id: 2, name: "Phone", price: 700, category: "Electronics", stock: 120 },

{ \_id: 3, name: "T-shirt", price: 20, category: "Clothing", stock: 200 }

])

db.orders.insertMany([

{ \_id: 101, product\_id: 1, quantity: 2, status: "delivered" },

{ \_id: 102, product\_id: 3, quantity: 5, status: "pending" },

{ \_id: 103, product\_id: 2, quantity: 1, status: "delivered" }

])

db.products.find()

db.products.updateOne({ \_id: 1 }, { $set: { price: 950 } })

db.products.deleteOne({ \_id: 3 })

//aggregate - use $

db.orders.aggregate([

{

$group: {

\_id: "$product\_id",

totalQuantity: { $sum: "$quantity" }

}

}

])

db.orders.aggregate([

{

$lookup: {

from: "products",

localField: "product\_id",

foreignField: "\_id",

as: "productDetails"

}

},

{ $unwind: "$productDetails" },

{

$project: {

\_id: 1,

product\_id: 1,

quantity: 1,

productName: "$productDetails.name",

status: 1

}

}

])

db.orders.aggregate([

{

$group: {

\_id: "$product\_id",

totalQuantity: { $sum: "$quantity" }

}

},

{

$merge: {

into: "orderSummary",

on: "\_id",

whenMatched: "replace",

whenNotMatched: "insert"

}

}

])

….. doc 4

use demoDB2025

db.users.insertMany([

{ name: "Alice", age: 25, city: "New York" },

{ name: "Bob", age: 32, city: "Chicago" },

{ name: "Carol", age: 29, city: "New York" },

{ name: "Dave", age: 40, city: "Chicago" },

{ name: "Eve", age: 35, city: "Boston" }

])

db.sales.insertMany([

{ product: "Laptop", price: 1000 },

{ product: "Phone", price: 700 }

])

db.users.countDocuments()

db.users.distinct("city")

db.users.aggregate([

{ $group: { \_id: "$city", averageAge: { $avg: "$age" } } }

])

db.sales.aggregate([

{ $project: { \_id: 0 } },

{ $merge: { into: "newCollection" } }

])

db.employee.findAndModify({

query: { name: "Ram" },

update: { $set: { department: "Development" } },

upsert: true

})

db.users.aggregate([

{ $match: { age: { $gt: 30 } } }

])

db.users.aggregate([

{ $sort: { age: 1 } }

])

db.users.find().sort({ age: 1 }).limit(2)

db.orders.insertMany([

{ \_id: 0, name: "Pepperoni", size: "small", price: 19, quantity: 10, date: ISODate("2021-03-13T08:14:30Z") },

{ \_id: 1, name: "Pepperoni", size: "medium", price: 20, quantity: 20, date: ISODate("2021-03-13T09:13:24Z") },

{ \_id: 2, name: "Pepperoni", size: "large", price: 21, quantity: 30, date: ISODate("2021-03-17T09:22:12Z") },

{ \_id: 3, name: "Cheese", size: "small", price: 12, quantity: 15, date: ISODate("2021-03-13T11:21:39.736Z") },

{ \_id: 4, name: "Cheese", size: "medium", price: 13, quantity: 50, date: ISODate("2022-01-12T21:23:13.331Z") },

{ \_id: 5, name: "Cheese", size: "large", price: 14, quantity: 10, date: ISODate("2022-01-12T05:08:13Z") },

{ \_id: 6, name: "Vegan", size: "small", price: 17, quantity: 10, date: ISODate("2021-01-13T05:08:13Z") },

{ \_id: 7, name: "Vegan", size: "medium", price: 18, quantity: 10, date: ISODate("2021-01-13T05:10:13Z") }

])  
  
db.orders.aggregate([

{ $match: { size: "medium" } },

{ $group: { \_id: "$name", totalQuantity: { $sum: "$quantity" } } }

])

db.orders.aggregate([

{

$match: {

date: { $gte: ISODate("2020-01-30"), $lt: ISODate("2022-01-30") }

}

},

{

$group: {

\_id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },

totalOrderValue: { $sum: { $multiply: ["$price", "$quantity"] } },

averageOrderQuantity: { $avg: "$quantity" }

}

},

{ $sort: { totalOrderValue: -1 } }

])