In a distributed MongoDB setup, the data is partitioned and distributed across multiple servers,

allowing for horizontal scaling. Each server in the cluster, known as a replica set member,

holds a subset of the data. This distribution of data helps to distribute the workload and

allows for handling larger datasets and higher traffic volumes.

Sharding: MongoDB uses a technique called sharding to horizontally partition data across

multiple shards (servers). Each shard contains a subset of the data. This enables efficient

distribution and parallel processing of data across the cluster.

Replica Sets: Each shard typically consists of a replica set, which is a group of MongoDB

instances that replicate the data across multiple nodes. Replica sets provide redundancy

and fault tolerance, ensuring that if one replica set member fails, another can take its place.

Balancing and Routing: The MongoDB cluster has a built-in balancer that automatically migrates

data between shards to ensure an even distribution of data. The routing component directs client

requests to the appropriate shard based on the data being accessed.

Sharding is a key feature of MongoDB's architecture that enables horizontal scaling of data across multiple machines or nodes. By distributing data across multiple shards, MongoDB can handle larger data sets and higher write/read workloads. However, using sharding on a single machine or your local PC does not make it a distributed MongoDB database.

To have a distributed MongoDB database, you would typically deploy MongoDB across multiple machines or servers. Each machine would run as a separate MongoDB node, and the data would be distributed and replicated across these nodes for improved scalability, fault tolerance, and high availability. This setup allows you to handle large-scale deployments and provides the benefits of a distributed database system.

In summary, while sharding is an essential component of a distributed MongoDB database, simply using sharding on your local PC does not make it a distributed database. It would still be running on a single machine, albeit with data partitioned across multiple shards.

MongoDB Shell: MongoDB provides a command-line interface called the MongoDB Shell (formerly known as the MongoDB JavaScript Shell). It allows you to interact with a MongoDB server using JavaScript syntax. You can install the MongoDB Shell on your Linux server and use it to execute JavaScript scripts to perform various database operations.

In a distributed MongoDB deployment, there is no concept of a dedicated master server that sends requests to all other servers. MongoDB follows a distributed architecture called sharding, where data is divided across multiple servers called shards.

In a sharded cluster, the data is horizontally partitioned and distributed among different shard servers. Each shard server is responsible for storing a subset of the data. The shards work together to handle client requests and distribute the workload.

However, there is a special component in a sharded MongoDB deployment called a "config server" or "config replica set." The config server keeps track of the metadata related to the sharded cluster, such as the mapping between data and shards. It holds the cluster's configuration data, including which shard contains which data ranges.

When a client sends a request to a MongoDB cluster, it typically connects to a MongoDB router called **mongos**. The **mongos** acts as a query router, directing the requests to the appropriate shards based on the metadata stored in the config server. The **mongos** component is responsible for coordinating the distributed query execution, aggregating results from multiple shards if necessary.

So, in summary, while there is no dedicated master server that sends requests to all other servers in a distributed MongoDB deployment, the **mongos** router acts as the intermediary between the client and the shards, coordinating and forwarding the requests appropriately based on the cluster's configuration.

When you enable sharding on your local MongoDB database, you are implementing a distributed database architecture that allows you to scale your data horizontally across multiple machines or nodes. Sharding is primarily used to address the limitations of a single machine's storage capacity and processing power, enabling you to handle larger data sets and increased traffic.

Here's what happens when you enable sharding:

1. Sharding Key Selection: You need to select a sharding key, which determines how the data will be divided and distributed across the shards. The sharding key is chosen based on the access patterns of your data and the scalability requirements of your application.
2. Shard Cluster Setup: You set up multiple MongoDB instances, referred to as shards, that will store portions of your data. Each shard is essentially an independent MongoDB replica set or a standalone MongoDB instance.
3. Config Servers: MongoDB employs a set of config servers to store the metadata about the sharded cluster, including the mappings of data chunks to the corresponding shards. These config servers maintain a distributed configuration database.
4. Enable Sharding: You enable sharding on the database or databases that you want to shard. This step tells MongoDB to start dividing the data and distributing it across the shards.
5. Data Distribution: When data is inserted or updated in the sharded collection, MongoDB uses the sharding key to determine which shard(s) should store the data. The data is then distributed across the appropriate shards based on the defined sharding strategy.
6. Balancing: MongoDB automatically balances the data distribution across the shards to ensure an even distribution and optimal performance. If a shard becomes overloaded or if data distribution is uneven, MongoDB migrates chunks of data between the shards to achieve balance.
7. Query Routing: When you query the sharded collection, MongoDB's query router (mongos) determines which shards contain the relevant data based on the sharding key and routes the query to those shards. The query results are then aggregated and returned to the client.

By enabling sharding, you can achieve horizontal scalability, increased storage capacity, and improved performance for your MongoDB database, allowing it to handle larger workloads and grow as your data requirements expand.

When you enable sharding in MongoDB, you send your requests to a MongoDB query router, which is known as **mongos**. The **mongos** process acts as a middle-tier service between your application and the sharded cluster.

When you configure sharding, you set up one or more **mongos** instances, typically on separate machines or VMs, which serve as the entry point for your application to interact with the sharded database.

Your application communicates with the **mongos** process as if it were a standalone MongoDB instance. The **mongos** router is responsible for routing the requests to the appropriate shards based on the sharding key. It maintains the knowledge of the sharded cluster's metadata and knows which chunks of data are stored on which shards.

When you send a read or write request to the **mongos** router, it determines which shards need to be involved in processing the request, forwards the request to those shards, and aggregates the results (in the case of read operations) or coordinates the write operation across shards.

In summary, you send your requests to the **mongos** process, and it takes care of routing the requests to the appropriate shards in the sharded cluster, allowing you to interact with the distributed database seamlessly.

Yes, the **mongos** query router in MongoDB has its own port. By default, **mongos** listens on port 27017 for client connections. This port is used by applications and clients to connect to the **mongos** process and send queries. The **mongos** instance acts as a proxy between the client applications and the underlying MongoDB sharded cluster, routing queries to the appropriate shards.

When you install MongoDB locally, the MongoDB query engine is part of the MongoDB server itself. The query engine is responsible for processing queries and executing them against the data stored in MongoDB

each replica set becomes sharded

increased load ---> scaling

each shard has its own memory, replica set, CPU

shard key is important.

\*\*query writer\*\*

-range sharding

-hash sharding

-zone sharding (teritorija)

Kako query zna gdje se nalazi info? U kom shardu?

Configuration server (ima routing tabelu)

Sta ako query ne sadrzi sharding key? Pogadjamo svaki shard (broadcast) i oni odgovore da li imaju te podatke ili ne..

Svaki put kada insertujemo podatke update nad routing tabelom.

\*\* Id kao sharding, i to hash (ne zelimo da se stalno upisuje u novi shard, sto bi bio slucaj kod range shardinga)

replica set - grooup of servers (nodes) that are completely the same

\*\* you dont actually shard db, you shard collection

When you shard a collection, MongoDB uses a process called chunking to split the data into smaller chunks. Each chunk represents a range of values based on the sharding key. These chunks are then distributed across the available shards.

So, while each shard contains a portion of the data from a single collection, it does not necessarily mean that each shard contains data for only one specific collection. In fact, a shard can contain data from multiple sharded collections.

To simulate a distributed MongoDB environment on your PC using Python, you can utilize the pymongo library, which is the official Python driver for MongoDB. Follow these steps to create a configuration server:

Install pymongo by running the following command:

pip install pymongo

Import the necessary modules in your Python script:

python

Copy code

**from pymongo import MongoClient**

**from pymongo.errors import ConnectionFailure**

*Create a function to set up the configuration server:*

**def create\_config\_server():**

*# Specify the connection details*

**host = 'localhost'**

**port = 27019**

*# Create a MongoClient instance*

**client = MongoClient(host, port)**

**try:**

*# Access the admin database*

**admin\_db = client.admin**

*# Create a dictionary for the configuration server options*

**config\_options = {**

**'\_id': 'config\_server',**

**'configsvr': True,**

**'members': [**

**{'\_id': 0, 'host': f'{host}:{port}'}**

**]**

**}**

*# Create the configuration server*

**admin\_db.command('replSetInitiate', config\_options)**

**print('Configuration server created successfully.')**

**except ConnectionFailure:**

**print('Failed to connect to MongoDB.')**

*Call the create\_config\_server() function to create the configuration server:*

**create\_config\_server()**

When you run the Python script, it will create a configuration server on the specified host and port. Ensure that the host and port are available and not being used by any other processes.

Note: Simulating a distributed MongoDB environment on a single machine may have limitations and may not fully reflect the behavior of a true distributed setup. It's recommended to use a proper distributed system for accurate testing and evaluation.

Sta sam ustvari uradila ovim? Ono sto nisam zeljela uopste..

1. Replica Set: A replica set in MongoDB is a group of database servers that maintain the same data set, providing redundancy and high availability. It consists of multiple nodes, where one node acts as the primary and the others serve as secondary nodes.
2. Configuration Server: The configuration server is a special type of MongoDB node responsible for storing and managing the metadata and configuration information of the replica set. It keeps track of the replica set's topology, membership, and various settings. The configuration server acts as a central repository for storing this information.

In summary, configuring a replica set with a configuration server in MongoDB provides a centralized and fault-tolerant mechanism for managing the metadata and configuration information of the replica set. It simplifies administration, ensures high availability, and supports scalability and sharding features.

In a MongoDB replica set, you can perform write operations on any node, regardless of whether it is a primary or a secondary node. However, for certain administrative operations, including creating a new collection, it is recommended to connect to the primary node.

The primary node is responsible for accepting write operations and coordinating data replication to the secondary nodes. It maintains the authoritative copy of the data set and metadata. By directing administrative operations to the primary node, you ensure that the operation is performed on a single node and replicated to the other nodes for consistency.

It's important to note that once the collection is created on the primary node, the data will be automatically replicated to the secondary nodes in the replica set, ensuring data redundancy and high availability.

Remember to always consider the MongoDB replication and sharding considerations to design your application and distribute the workload effectively.

Yes, replicas of MongoDB are typically deployed on different hosts. MongoDB's replication feature allows you to create a replica set, which consists of multiple instances of MongoDB running on separate machines. Each member of the replica set serves a specific role: one primary node for read-write operations and one or more secondary nodes for replication and read scaling.

The primary node is responsible for processing write operations and propagating the changes to the secondary nodes. The secondary nodes replicate the data from the primary node and can serve read operations. By distributing the workload across multiple hosts, replica sets enhance the availability, reliability, and scalability of your MongoDB deployment.

Nije bitno koji server iz replica set-a gadjam, automatski se odrzava da bude isti sadrzaj u svim.

REPLICATION AND SHARDING

Replication and sharding are two key features of MongoDB that address different aspects of data management and scalability:

1. Replication: Replication in MongoDB ensures high availability, fault tolerance, and data redundancy. It involves maintaining multiple copies of data across a set of database servers called a replica set. The primary purposes of replication are:

a. High Availability: By having multiple copies of data, if the primary node fails, one of the secondary nodes can automatically be elected as the new primary, ensuring that the database remains accessible.

b. Fault Tolerance: Replication provides resilience against server failures. If one or more nodes go offline, the replica set can continue to function with the remaining nodes, preserving data availability.

c. Data Redundancy: Data is replicated across multiple nodes, reducing the risk of data loss. If a node fails, the data can still be accessed from the other nodes.

Replication also supports read scaling by allowing clients to read from secondary nodes, offloading some of the read traffic from the primary node.

1. Sharding: Sharding is a technique used for horizontal scaling in MongoDB. It involves partitioning data across multiple servers called shards based on a shard key. The primary purposes of sharding are:

a. Scalability: Sharding allows you to distribute data and operations across multiple servers, enabling horizontal scaling as your data and workload grow. Each shard can store a subset of the data, reducing the amount of data stored on each individual server.

b. Performance: By distributing data and operations, sharding can improve query performance by allowing parallel execution of queries across multiple shards. This enables the database to handle a higher volume of read and write operations.

c. Storage Capacity: Sharding helps you overcome storage limitations by allowing you to store large amounts of data across multiple servers.

Sharding involves a sharded cluster architecture, which includes three main components: the query router (mongos), config servers, and shard servers. The query router routes client requests to the appropriate shard based on the shard key defined for the data.

By combining replication and sharding, you can achieve both high availability and scalability. Replica sets provide redundancy and fault tolerance for each shard, while sharding allows you to distribute data and workload across multiple shards for improved performance and increased storage capacity.

Knjiga koju sam nasla:

Range sharding stvara casacade efekar kod prebacivanja podataka izmedju jednog sharda do drugog. Dodavanje novog sharda pomaze, ali malo.

Range of data is called a chunk. When we split chunks range into two ranges, it becomes two chunks.

MIGRACIJA OGRANICENJA:

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017')

db = client['your\_database']

# Define the validator

validator = {

'$jsonSchema': {

'bsonType': 'object',

'required': ['name', 'age'],

'properties': {

'name': {'bsonType': 'string'},

'age': {'bsonType': 'int', 'minimum': 0}

}

}

}

# Add the validator to an existing collection

collection\_name = 'your\_collection'

collection = db.create\_collection(collection\_name, validator=validator)

# Test validation by inserting a document

document = {'name': 'John', 'age': 30}

collection.insert\_one(document) # This will succeed

document = {'name': 'Jane', 'age': -5}

collection.insert\_one(document) # This will fail due to the age being negative

Clanak je odlican, kljucni dijelovi:

The MongoDB cluster architecture mainly includes three parts: the shard server, the router server,

and the config server. The actual data are stored in the shard server, which can be a replica set or a

server. The router server is the entry that clients request the database and is used to address and locate

the requests from these clients. All these requests need to be handled by mongos and forwarded to the

corresponding shard server. The config server is used to store the configuration information of the

router and the shard, which is set up at first and does not need significant space and resources.

Typically, a MongoDB sharded cluster consists of three config servers for redundancy and fault tolerance. These config servers can be deployed as a replica set for high availability. Each config server replica set member runs on a separate server to ensure fault isolation and reliability.

Yes, a replica set is a set of servers that work together to provide high availability and data redundancy in a distributed database system. In the context of MongoDB, a popular NoSQL database, a replica set consists of multiple MongoDB instances (servers) that store the same data set. One of the instances acts as the primary node, which receives all write operations and propagates changes to the other nodes in the replica set, known as secondary nodes. The secondary nodes replicate the data from the primary node, allowing them to take over as the primary in case of a failure or during maintenance operations. Replica sets ensure data durability and availability by providing automatic failover and data replication across multiple servers.