Setting Up a MongoDB Replica Set with Docker Compose

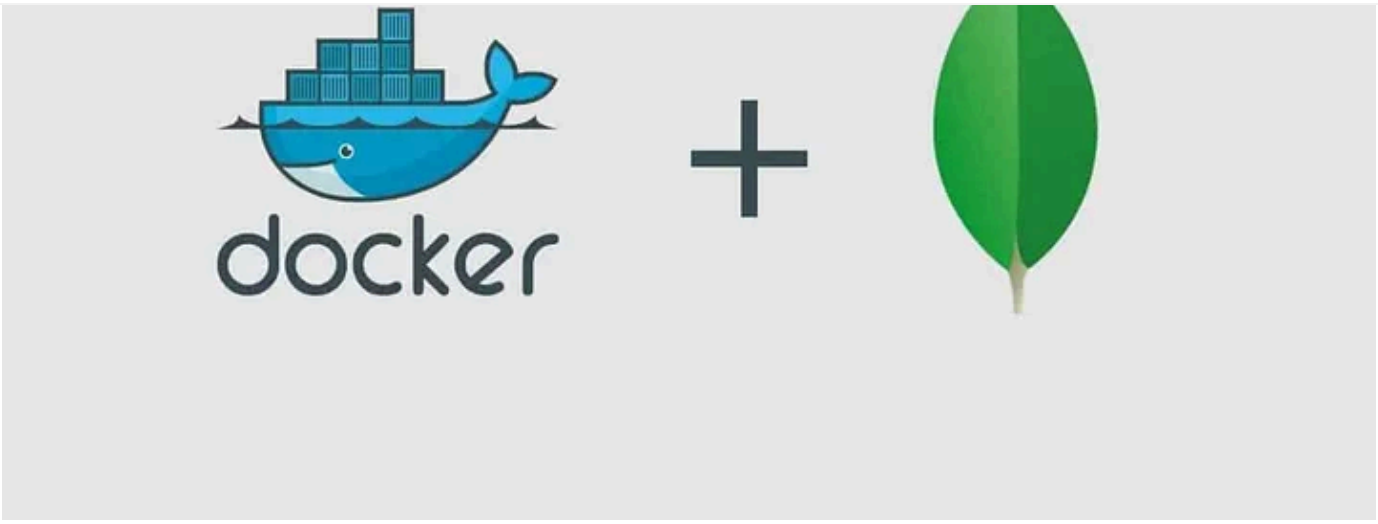
4 min read · May 28, 2024

Ravikushwaha Rk Follow

Listen Share

Open in app ↗ Sign up Sign in

Medium Search



Introduction

A MongoDB cluster is a configuration of multiple MongoDB instances designed to work together to store, manage, and process data efficiently. There are two main types of clusters in MongoDB: replica sets and sharded clusters.

Types of MongoDB Clusters

1. **Replica Sets:** A group of mongod instances that maintain the same data set, providing redundancy and high availability.

- **Components**
  - **Primary Node**: Handles all write operations and coordinates the replication process.
  - **Secondary Nodes**: Replicate the primary's data set and can serve read operations (depending on the read preference configuration).
  - **Arbiter**: Participates in elections to select a new primary but does not store data.
- **Importance:** Ensures data redundancy, automatic failover, and data consistency, which is critical for high availability and disaster recovery.

2. **Sharded Clusters:** Distributes data across multiple servers or clusters (shards) to handle large data sets and high throughput operations.

- **Components:**
  - **Shards**: Each shard holds a subset of the data, acting as a replica set to ensure high availability.
  - **Config Servers**: Store metadata and configuration settings for the cluster, managing data distribution and balancing.
  - **Query Routers (mongos)**: Direct client requests to the appropriate shard, providing a unified interface to the cluster.
- **Importance**: Facilitates horizontal scaling, allowing the database to handle growth in data volume and traffic by adding more shards, thus improving performance and capacity.

Importance of MongoDB Clusters

- **Scalability**: Sharded clusters enable horizontal scaling, allowing databases to expand seamlessly as data grows, providing consistent performance and capacity.
- **High Availability**: Replica sets ensure that the database remains operational even in the event of hardware failures or maintenance, with automatic failover to secondary nodes.
- **Disaster Recovery**: Data redundancy across multiple nodes and locations protects against data loss and provides a robust disaster recovery solution.
- **Load Balancing**: Distributes read and write operations across multiple nodes, optimizing resource utilization and improving performance.
- **Geographic Distribution**: Data can be distributed across different geographic locations, enhancing data locality and reducing latency for global applications.

Prerequisites

Before we begin, ensure you have the following installed on your machine:

- Docker
- Docker Compose

Step-by-Step Guide

1. **Define the Docker Compose File**

We'll start by defining our Docker Compose file, which will specify the services, networks, and volumes required for our MongoDB replica set.

```
services:
  mongo-1:
    container_name: 'mongo-1-container'
    entrypoint: >
      /bin/bash -c '
      openssl rand -base64 756 > /data/keyfile.key &&
      chmod 400 /data/keyfile.key &&
      chown mongodb:mongodb /data/keyfile.key &&
      /usr/local/bin/docker-entrypoint.sh mongod --replSet rs0 --keyFile /data/keyfile.key --bind_ip_all'
    image: 'mongo:7.0.0'
    ports:
      - 127.0.10.1:27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=sa
      - MONGO_INITDB_ROOT_PASSWORD=Password123
      - MONGO_INITDB_DATABASE=myDatabase
    volumes:
        - 'mongo1data:/data/db'
        - 'mongo1config:/data/configdb'
        - 'sharedconfig:/data'
    healthcheck:
      test: mongosh
        -u $${MONGO_INITDB_ROOT_USERNAME}
        -p $${MONGO_INITDB_ROOT_PASSWORD}
        --eval "try { rs.status() } catch (err) { rs.initiate({_id:'rs0',members:[{_id:0,host:'mongo-1:27017',priority:1},{
      interval: 5s
      timeout: 30s
      start_period: 0s
      start_interval: 1s
      retries: 30
    networks:
      - "mynetwork"
```

```
    restart: unless-stopped

  mongo-2:
    container_name: "mongo-2-container"
    image: 'mongo:7.0.0'
    ports:
        - 127.0.10.2:27017:27017
    volumes:
        - 'mongo2data:/data/db'
        - 'mongo2config:/data/configdb'
        - 'sharedconfig:/data'
    command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27017", "--keyFile", "/data/keyfile.key"]
    networks:
      - "mynetwork"
    restart: unless-stopped

  mongo-3:
    container_name: 'mongo-3-container'
    image: 'mongo:7.0.0'
    ports:
       - 127.0.10.3:27017:27017
    volumes:
        - 'mongo3data:/data/db'
        - 'mongo3config:/data/configdb'
        - 'sharedconfig:/data'
    command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27017", "--keyFile", "/data/keyfile.key"]
    networks:
      - "mynetwork"
    restart: unless-stopped

volumes:
  mongo1data:
  mongo1config:
  mongo2data:
  mongo2config:
  mongo3data:
  mongo3config:
  sharedconfig:

networks:
  mynetwork:
    driver: bridge
```

2. Understanding the Configuration

- Version: We specify the Docker Compose file version to ensure compatibility (`version: '3.8'`).
- Services: We define three services (`mongo-1` , `mongo-2` , and `mongo-3`), each representing a MongoDB node.
- mongo-1: The primary node with an entrypoint script to generate a key file for authentication and initiate the replica set.
- mongo-2 and mongo-3: Secondary nodes configured to join the replica set.
- Volumes: We create named volumes for persistent data storage.
- Networks: We define a custom bridge network for our services to communicate.

3. Initializing the Replica Set

The healthcheck for `mongo-1` ensures the replica set is initiated. The `rs.initiate()` command configures the replica set with the three nodes. This healthcheck runs at intervals to ensure the replica set is correctly initialized.

4. Running the Setup

To bring up the MongoDB replica set, navigate to the directory containing your `docker-compose.yml` file and run:

```
docker-compose up -d
```

This command will start the containers in detached mode.

5. Verifying the Replica Set

```
docker exec -it mongo-1-container mongosh --eval "rs.status()"
```

This command will display the status of the replica set, confirming that all nodes are functioning as expected.

6. Connect to Mongo Compass

To connect to the MongoDB nodes from outside the Docker environment, you need to map the Docker internal IP addresses to hostnames in your system's hosts file.

Add the following lines to map the IP addresses to hostnames:

```
127.0.10.1 mongo-1
127.0.10.2 mongo-2
127.0.10.3 mongo-3
```

On Windows, the hosts file is located at `C:\Windows\System32\drivers\etc\hosts`. Edit it with a text editor run as Administrator.

In MongoDB Compass, enter the following connection string in the connection dialog:

```
mongodb://sa:Password123@127.0.10.1:27017,127.0.10.2:27017,127.0.10.3:27017/myDatabase?authSource=admin
```

- sa: The username for authentication.
- Password123: The password for authentication.
- 127.0.10.1:27017,127.0.10.2:27017,127.0.10.3:27017: The IP addresses and ports of the MongoDB nodes.
- myDatabase: The initial database to connect to.
- authSource=admin: Specifies the database that holds the user's credentials.

Conclusion

Setting up a MongoDB replica set with Docker Compose is a straightforward process that ensures your database is resilient and highly available. By following this guide, you can create a robust MongoDB setup suitable for development and production environments. Happy coding!

Docker | Docker Compose | Mongodb | Mongo | Replication

Follow

Written by Ravikushwaha Rk

3 Followers · 2 Following

Responses (1)

Write a response

What are your thoughts?

Pedro Resende
Oct 23, 2024

Great but doesn't work...

When you try to check the status by running the command

docker exec -it mongo-1-container mongosh --eval "rs.status()"

it returns the error

MongoServerError: Command replSetGetStatus requires authentication

What's next:

Try... more

2 replies Reply

More from Ravikushwaha Rk

Ravikushwaha Rk

MongoDb Cluster Generic Repository Pattern For .Net Core

Introduction

Jun 14, 2024 2

Ravikushwaha Rk

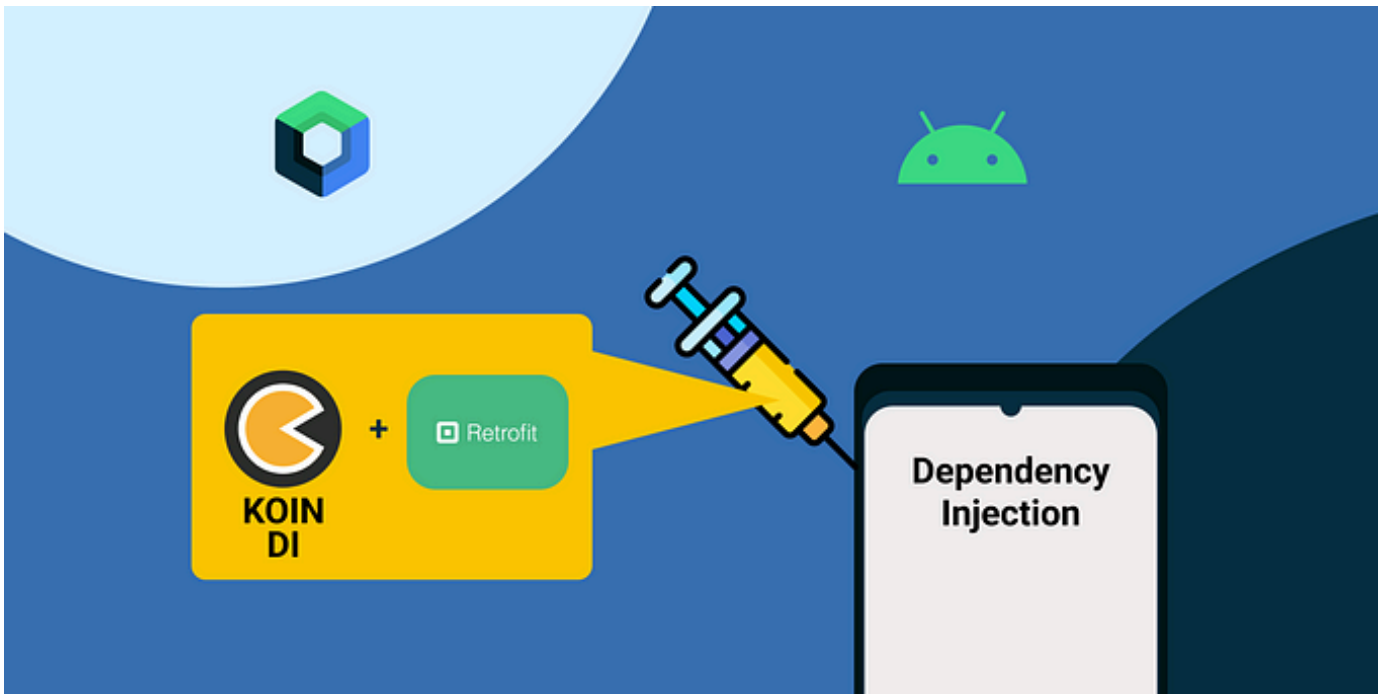
Microservices Communication Pattern: API Gateway with Ocelot

Introduction

Jun 26, 2024 2

See all from Ravikushwaha Rk

Recommended from Medium

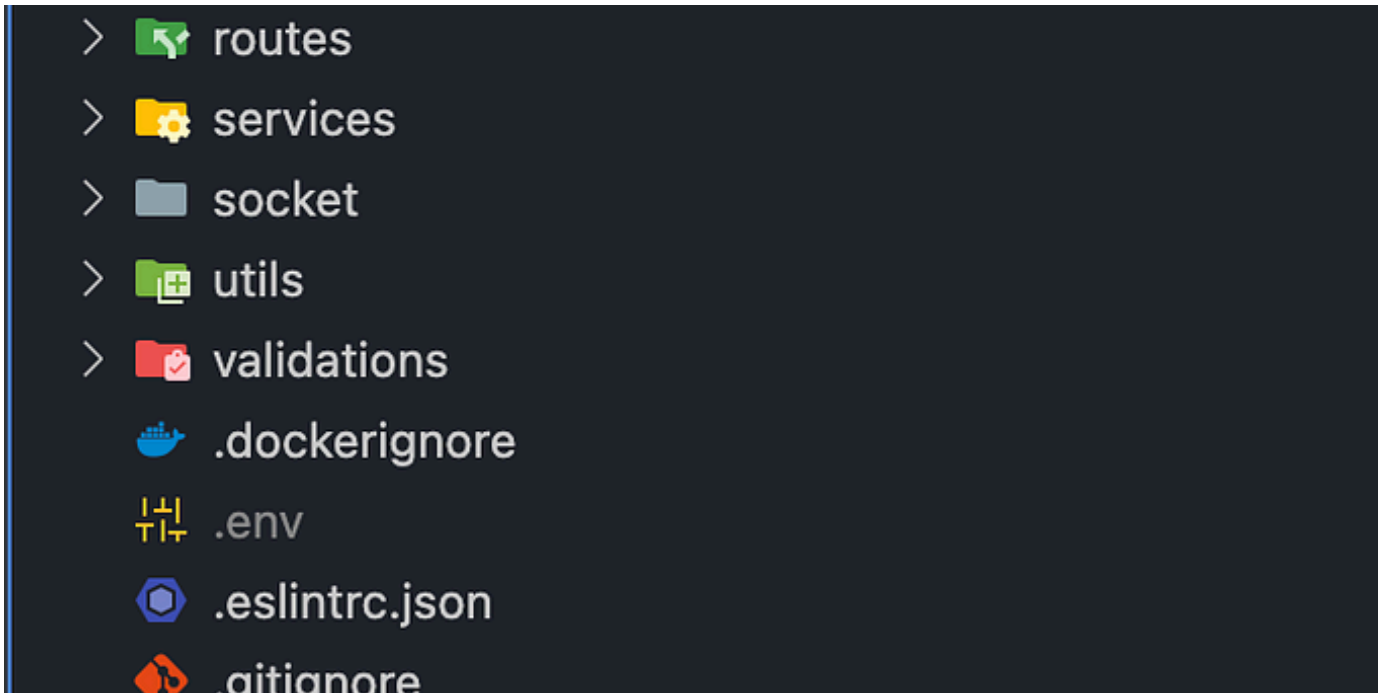


Modtion

Inject Retrofit with Koin Dependency Injection to Android Compose

Android Compose + Retrofit + Dependency Injection

Feb 26



Shan Abbas

The Ultimate Guide to Node.js Project Structure: Building Scalable Applications

A well-organized folder structure is crucial for maintaining a scalable and maintainable Node.js application. This guide presents a...

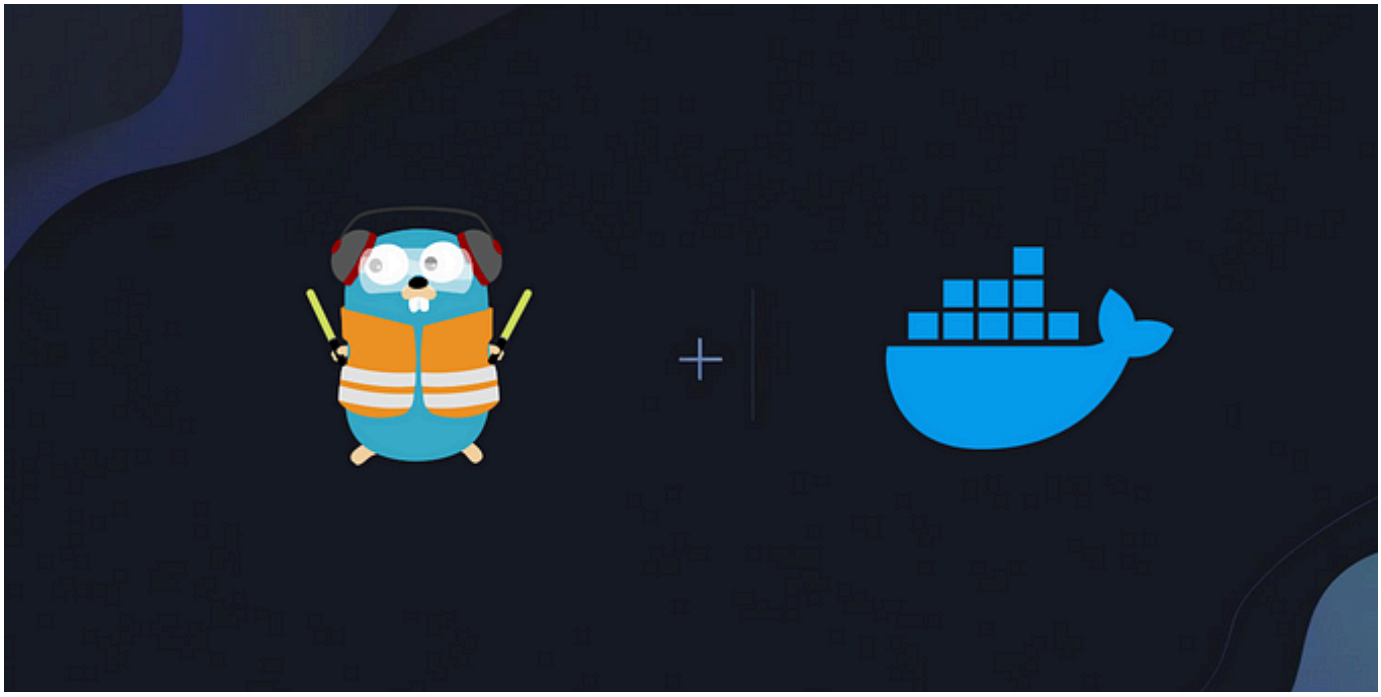
Jan 9 5

Kaloyan Manev

How to Run RabbitMQ in Docker Compose

Discover how to run RabbitMQ locally with its management console enabled—save time with a quick Docker Compose fix!

Feb 25 11

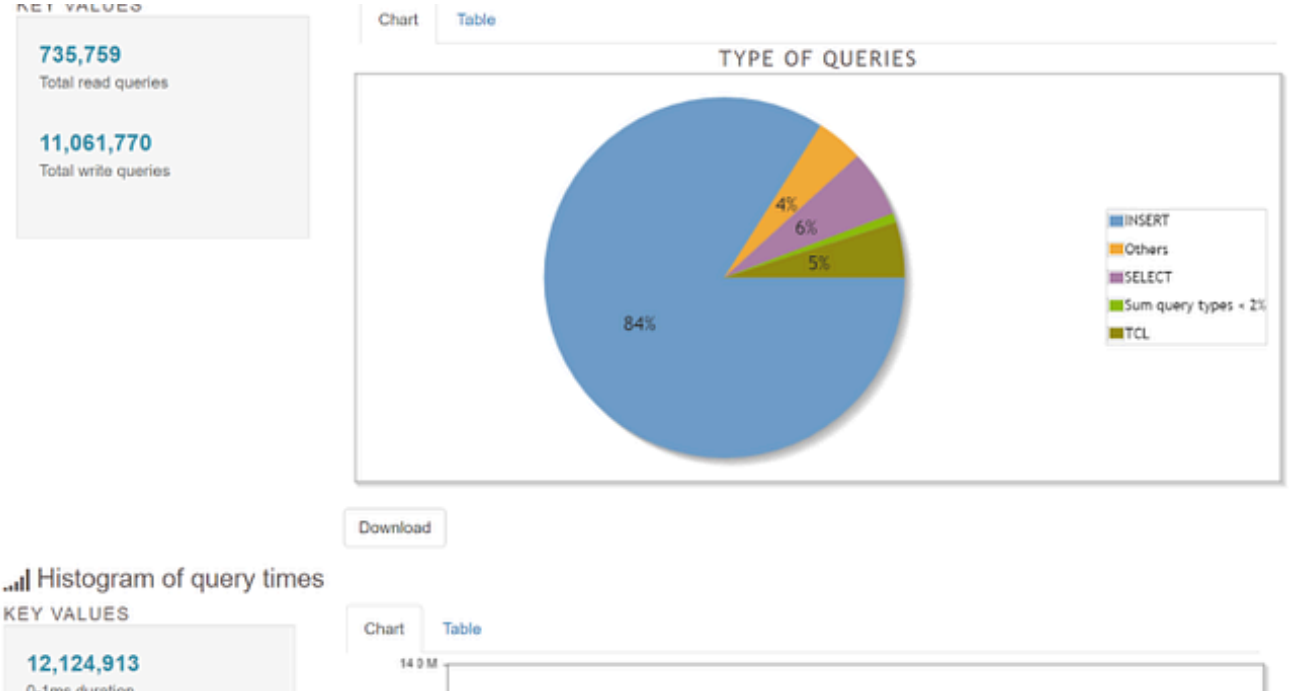


Prateek Jain

Why I Replaced NGINX with Traefik in My Docker Compose Setup

For a long time, NGINX was my go-to solution for setting up reverse proxies. Whether it was a Node.js app, a Django backend, or even a...

6d ago 565 10



Mohsin Ali

Oracle AWR in PostgreSQL? Here's What I Use to Troubleshoot Like a Pro

As an Oracle DBA stepping into the world of PostgreSQL or MySQL, one of the first questions that naturally pops up is: "What's the...

Apr 12 15

Tihomir Manushev

Cron Jobs with PostgreSQL and the pg_cron extension

plus Docker and Ubuntu

Mar 5 11 1

See more recommendations