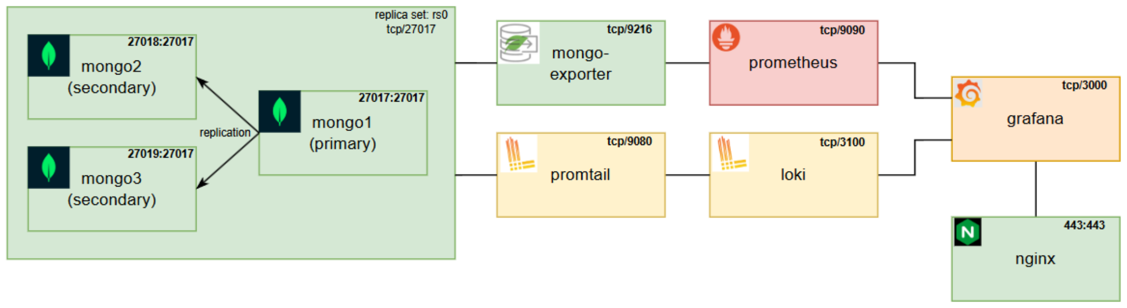
MongoDB Cluster, Metrics, Logs and Monitoring Setup Documentation

Warsaw 2025

Table of Contents

1. Overview

This document details the setup of a secure, high-availability MongoDB cluster with logging, custom metrics, and a complete monitoring system using Docker Compose, Prometheus, Loki, and Grafana. The project ensures robust data management, real-time monitoring, and secure communications across services. The architecture of the whole system is shown below:



Picture 1. Architecture of system

2. MongoDB Cluster Setup

Replica Set for High Availability

The MongoDB is deployed as a replica set to ensure high availability. Three MongoDB instances are created (one primary and two secondary) and the replica set is initiated for fault tolerance.

Authentication

The cluster is secured using authentication. Initial admin account and application-specific users are created using environment variables (e.g., MONGO_INITDB_ROOT_USERNAME and MONGO_INITDB_ROOT_PASSWORD) during initialization.

TLS for Secure Communication

TLS protocol is enabled to encrypt communications between MongoDB instances and clients. TLS certificates are generated with *openssl* utility and stored in PEM format. MongoDB is configured to use these certificates by mounting certificate directory as a volume.

Data Persistence

MongoDB is ensuring data persistence by using Docker volumes for data storage.

3. Database and Collections

For testing purposes three databases and several collections are created. Sample documents are inserted into each of the collection. Everything is set using *mongo-init.js* init script. In order to create additional objects it is possible to do it manually by accessing MongoDB:

1. Connect to MongoDB using a client (e.g., mongo shell or MongoDB Compass).
2. Create databases using the 'use' command and create collections with 'db.createCollection()'.
3. Insert sample documents using 'insertOne()' or 'insertMany()' methods.

4. MongoDB Logs

Configuring MongoDB Logging in JSON Format

The MongoDB cluster is configured to output logs in JSON format. Cluster configuration is adjusted to enable JSON log output with the appropriate settings (such verbosity levels).

Forwarding Logs with Promtail

Promtail is set up to tail the MongoDB cluster logs, parse the JSON log entries, and forward them to Loki for centralized log management.

5. Monitoring System

Prometheus Setup and MongoDB Exporter

Prometheus is deployed to collect metrics from MongoDB using a MongoDB Exporter. Prometheus is configured to scrape metrics from the exporter endpoint, capturing cluster metrics such as uptime, rates of inserts, updates, deletes, database operations, active connections and memory consumption.

Grafana Dashboards

Grafana dashboards are created to visualize metrics and logs. The system has configured two dashboards:

- Dashboard 1: MongoDB Metrics
- Dashboard 2: MongoDB Logs

6. Docker Compose & Security

Persistent Volumes

Docker volumes are used to ensure persistent storage for MongoDB, Loki, and Grafana data, thereby preventing data loss.

Environment Variables and Secrets for Sensitive Data

Sensitive configuration data (e.g., usernames, passwords, certificate paths) is passed via environment variables and secrets in the Docker Compose files to avoid hard-coding credentials.

Minimal Port Exposure and TLS Configurations

Only the necessary ports are exposed to the host network. TLS is enabled where required for secure communications (for example, using an NGINX reverse proxy for Grafana with HTTPS, or MongoDB cluster management).

7. Installation and Setup Instructions

1. Prerequisites:

- Install Docker and Docker Compose on your host system.
- Generate or obtain TLS certificates for MongoDB and reverse proxy (NOTE: for ease of testing, predefined certificates were added to the repository; for production environment it is advised to generate new certificates).
- Generate admin password in the secret/env files (NOTE: for ease of testing, predefined passwords were added to the repository; for production environment it is advised to generate new passwords).

2. Clone the Repository:

git clone https://github.com/Internowany/mongodb-project.git

3. Review and Update Configurations (optional):

- Update secrets, env files and mongo-keyfile, replacing placeholders with your environment-specific values (passwords, certificates).
- Adjust other configuration files as needed.

4. Start the Services:

Execute starting script in the terminal (the script also adjusts files ownership and permissions (NOTE: mongo-keyfile **MUST** have set *chown 400* (only read by a user) and *chmod 999:999* (the owner of the file must be user docker)))

./start.sh

5. Verification:

You can use below command to verify the connectivity to the MongoDB cluster:

mongosh --host 127.0.0.1:27017 -u admin -p password --tls --tlsCAFile ./tls/ca.crt --tlsCertificateKeyFile ./tls/mongo.pem

8. Accessing the Dashboards

Grafana can be accessed via NGINX reverse proxy with HTTPS at https://localhost. Login using the default or configured credentials and import the provided dashboards:

- Dashboard 1: MongoDB Metrics
- Dashboard 2: MongoDB Logs

9. Summary

This setup provides a robust, secure, and highly available MongoDB cluster with integrated logging and monitoring solutions. By leveraging Docker Compose, Prometheus, Loki, and Grafana, the system offers persistent storage, custom metrics, and centralized log management, making it well-suited for production workloads. Follow the provided instructions to deploy, secure, and maintain the environment effectively.