AT70.18 Software Architecture Design

PROJECT REPORT
(Team1 - Reminder App)

Submitted to:

Dr. Chaklam Silpasuwanchai

Submitted by:

Michael Mueller (st122797)
Sricharan Yedu Thirnathi (st122037)
Panuvit Chantara (st121410)
Sai Preetham Kamishetty (st122038)

https://github.com/M2701M/SAD-Team1-Submission

Date of Submission:

April 19th, 2022

School of Engineering and Technology

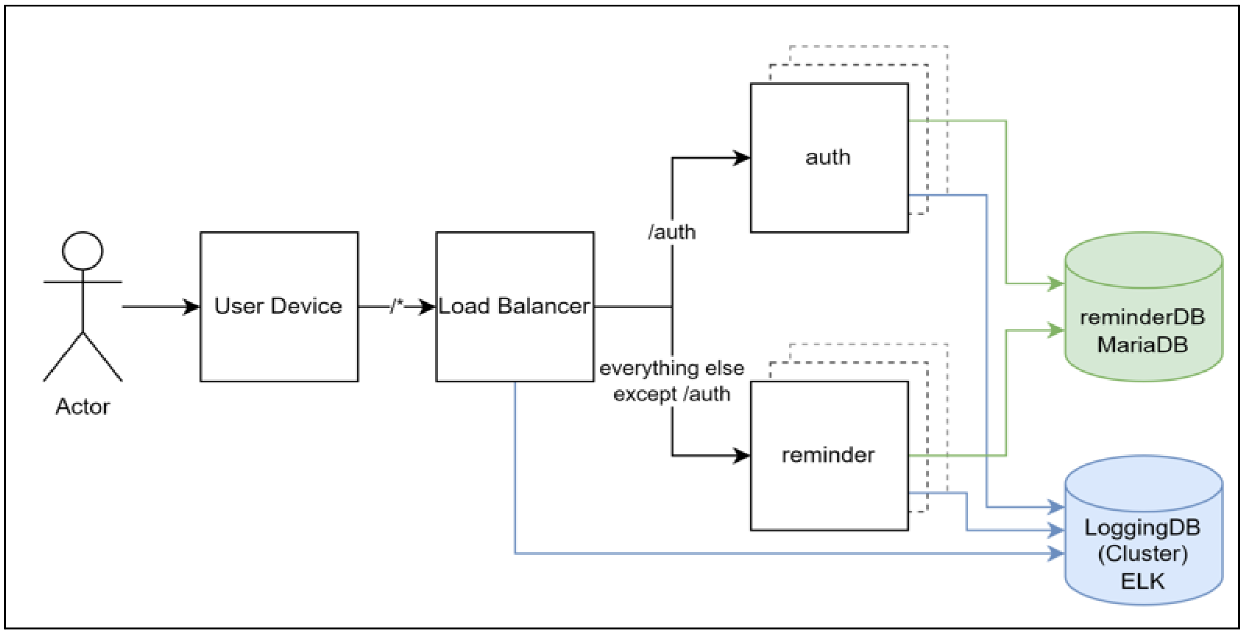
Asian Institute of Technology

Motivation and Scope

Time management plays a crucial role in our day to day lives. It is important that we set priorities on our day to day activities. To solve this problem, the best way to do this is to maintain a reminder list. Our Project “RemindMe” helps users in achieving this goal. It helps the user by allowing them to signup and login and create reminders for the tasks they want to perform in the near future. The user will be able to create, read, update and delete their reminders satisfying the CRUD terminology. They also checklist the reminders when they finish the task by changing the task to completed. Apart from this the user will also be able to share their reminders to other users by making them public.

The scope of this project is to build a microservice model architecture system for the RemindMe application. The application allows users to create, read, update and delete the reminders and also share the reminders they created to others by making them public. The system is developed using Django and Django REST framework, by incorporating MariaDb as database and DockerSwarm and Nginx for load balancing. The current system incorporates two servers, where docker manager node and nginx are present in the first server and django apps (including authentication_service and reminder_service) and the database are present on the second server.

Architectural Solution



Load balancing

Nginx is used as a load balancer and reverse proxy that redirects all requests from the users to either the authentication service or the reminder service. Docker Swarm is used to orchestrate the pods and add more resources to one of the services if necessary.

Pros:
- Reverse proxy can be used to access services running on different machines and ports in the same webpage
- More services can be added easily
- Docker containers allow for horizontal scaling

Cons:
- Containers don't run at bare metal speed. With only one physical server, it would make more sense to run all applications directly on the machine.
- The setup is more complex than having a monolithic application that handles everything
- Containers for databases are complicated because of consistency and replication
  ⇒ Need a separate solution for database clusters
  ⇒ More complexity yet again

⇒ The bigger the application, the more we benefit from nginx and docker swarm.

Backend

Django is used as a backend service to handle all requests, talk to the database, and return the response to the clients. Our backend is split into two applications: auth and reminders. Both applications can run separately in docker containers and be replicated for horizontal scaling.
Client-based session handling via **JWT** is used for authentication and authorization. Once a user logs in successfully, he is sent a token from the auth_service that contains the username and a signature. Given that the reminder_service has access to the same secret that is used to sign the token, the reminder_service can easily verify the token without the need of making any request to the database. Our application is designed to be able to handle all CRUD requests of the reminders with only the information inside the token.

Pros:
- Auth_service and reminder_service can be scaled independently
- JWT does not require a database for the tokens and thus scales very well and is not relying on a working database
- Django integrates well with frontend frameworks and can be extended easily

Cons:
- Separation of auth_service and reminder_service necessitates reverse proxy
- JWT is hard to invalidate upon user logout and may require a blacklist database

Logging

We propose to use a software stack consisting of Filebeat, Elasticsearch, Logstash, and Kibana for all logging purposes. Instead of sending every single log message one-by-one to a database, logs can be collected in log files locally first, then shipped via Filebeat and Logstash to Elasticsearch and displayed and queried in Kibana.

Pros:
- Highly optimized bulk operations possible
- Logs can be collected from container and any source

Cons:
- Complex set up
- High resource requirements

Database

MariaDb as a database is being used in our system architecture. Both authentication and reminder services are connected to a single database.

Pros
- Better performance, security, and backwards compatibility than mysql
- Can be clusterized and horizontally scaled
- Better integration with django than e.g. MongoDB

Cons
- SQL data schemas are less flexible than NoSQL

Experiments

In the short run, we plan to run the following experiments to assess performance and resilience of the server:

1. Load and performance testing with Apache JMeter
2. Resilience testing by turning off some docker containers randomly

In the long run, we plan to perform additional tests as follows:

1. Load and performance tests of the elasticsearch
2. With more physical machines in our cluster, turn off some machines at random to assess the resilience.