RemindMe App

Team#1 - Final Presentation

Agenda

1. Application Overview
2. Architecture
3. AP: Shared Sessions
4. AP: Logging
5. Experiments
6. Demo

Application Overview

Application Overview

Users can

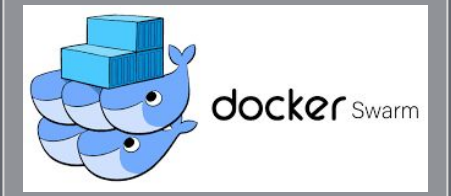
- Sign up / sign in
- Create, read, update, and delete their reminders with information such as
  - Title
  - Description
  - Public or Not
- Public reminders can be viewed publicly
- Only the creator of the reminder can edit or delete the record; other users can view the reminder if it is set to public.

Architecture

Software Stack

Server 1

Load Balancing

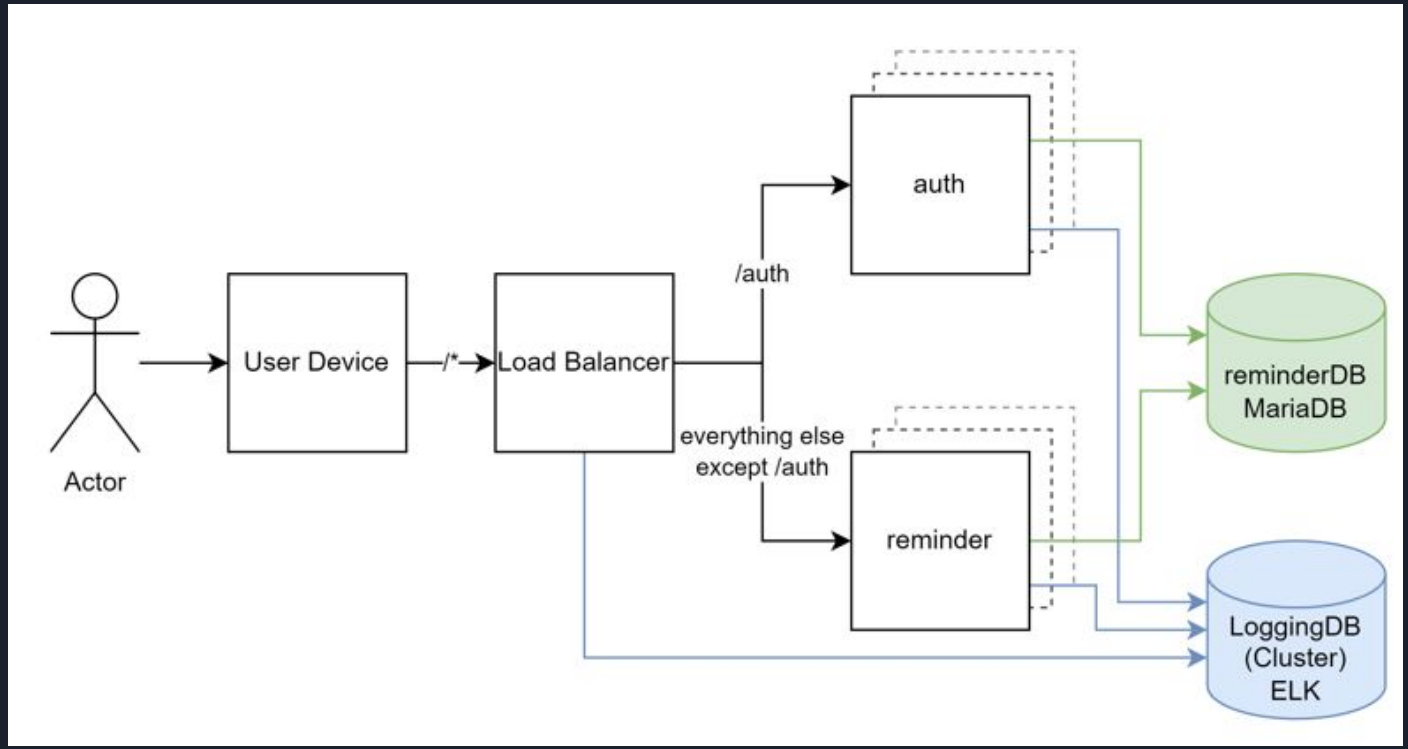


Server 2

backend

Database

Overview


Actor
User Device
/*
Load Balancer
/auth
auth
everything else
except /auth
reminder
reminderDB
MariaDB
LoggingDB
(Cluster)
ELK

Data Schema

User:

- **email**
- **username**
- **Password**

- id
- is_superuser
- first_name
- last_name
- ...

Reminder

- **id**
- **title**
- **content**
- **completed**
- **regisDate**
- **created_by**
- **public**

AP: Shared Sessions

AP: Shared Sessions

Problem: Many independent django instances must be aware of client's state (e.g. authentication state, user's application state (if applicable), …)

Solution: Client-based session with JWT

- Tokens are signed by authentication service upon successful authentication
- May contain data such as username and expiration date
- Data can't be changed by the client without breaking the signature

⇒ Each reminder service-instance can verify tokens **independently** and **without accessing** the database (unlimited scaling)

Typical Problems with JWT

Statelessness: Token can't be adapted to the application's state
⇒ Our application is designed to be stateless

How to invalidate tokens: Tokens are still usable even after logging out

- Remove token from client
  ⇒ doesn't help if someone stole the token earlier
- Keep a database of blacklisted tokens
  ⇒ defies the purpose of going client-based (but at least the database would be small)
- Keep expiration times short and get regular refresh tokens
  ⇒ doesn't solve the problem, only shortens the time in which outdated tokens are valid
  ⇒ additional load for authentication service

Why not database-side or server-side session?

Database-side session handling:

- Slower and poor scaling
- Doesn’t work if the database breaks (single point of failure)

Server-side session handling:

- Requests need to be forwarded to the same server instance or servers need to synchronize themselves
- Risk of data loss if an instance dies

AP: Logging

AP: Logging - Overview

Why not simply use MariaDB again to store all logs messages?

- Setup would be the easiest way
- Would probably work

⇒ Log data is inherently different from business data

- Only create and read, never update; delete based on data retention policy
- Can make use of bulk operations
- Must be easily searchable and displayable for developers

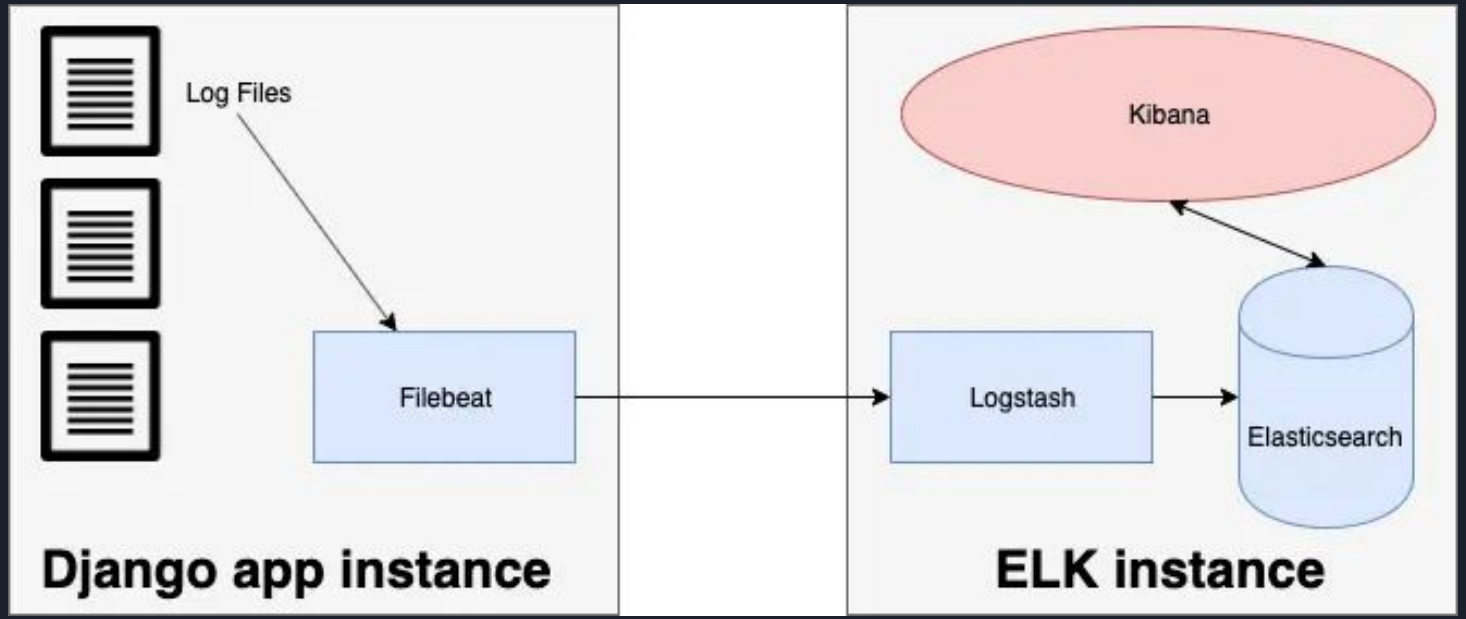
AP: Logging - Proposed Solution

Proposed solution: Filebeat → Logstash → Elasticsearch → Kibana

1. Log messages are not sent to the database one-by-one, but collected locally first
2. **Filebeat** fetches local log files and ships them to Logstash
3. **Logstash** processes log messages and sends them to **Elasticsearch**
4. **Elasticsearch** stores log data in indices
5. **Kibana** displays log data from Elasticsearch indices in a dashboard

Due to problems with the set-up, the solution is not fully implemented yet.

AP: Logging - Proposed Solution



AP: Logging - Pros and Cons

Pros

- Highly optimized and scalable for querying and indexing of log data
- Already includes the frontend dashboard
- Log files can be collected from any service and container

Cons

- ELK stack requires a lot effort for maintenance and setup
- High resource requirements

Experiments

Experiments

Short-term scope:

1. Load and performance testing with Apache JMeter
2. Resilience testing by turning of some of the docker nodes

Long-term scope:

3. More physical machines → turn some machines off randomly
4. Load testing of ELK stack

Demo

Thank you