Eagles: MiniTwit

Trond Pingel Anchær Rossing trro@itu.dk, Roman Zvoda rozv@itu.dk, Rasmus Balder Nordbjærg rano@itu.dk, Daniel Spandet Grønbjerg dangr@itu.dk, Jan Lishak jlis@itu.dk

Course infomation:

Name: **DevOps, Software Evolution and Software Maintenance, MSc (Spring 2024)**
Code: KSDSESM1KU
Description: (https://learnit.itu.dk/local/coursebase/view.php?ciid=1391)[https://learnit.itu.dk/local/coursebase/view.php?ciid=1391]
Teaching staff: Helge Pfeiffer ropf@itu.dk, Mircea Lungu mlun@itu.dk

1.Introduction:

The project focuses on building and maintaining a mock version of Twitter called Minitwit by applying various DevOps techniques such as automatization, cloud deployment, scaling, maintainability, monitoring, testing and other.
Initial web-application whcih was in out of date with the latest technologies and was not following any standart pracctices was rewriten from the ground up and gradually equiped with automations and improvements so that it is able to process and high load of requests comming to the app.

Name	Description	Link
Golang	Statically typed, compiled high-level programming language	https://go.dev/
crypto	Package supplying different cryptography libraries for golang	https://golang.org/x/crypto
net/http	Package used to make HTTP requests	https://pkg.go.dev/net/http
Gorm	easy to use ORM library for Golang	https://gorm.io/gorm
Grafana	Open source analytics and monitoring solution used for database	https://grafana.com/
Mimir	long term storage for grafana data	https://grafana.com/oss/mimir/
Loki	Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus	https://grafana.com/oss/loki/
Prometheus	open-source software for monitoring webapps	https://github.com/prometheus/
xxhash	Golang implementation of the (fast) 64-bit xxHash algorithm	https://github.com/cespare/xxhash
Gorilla	Package that supplies different tools for developing web-applications in golang	https://github.com/gorilla
Gorilla/mux	Package for request routing	https://github.com/gorilla/mux
Docker	System for deployment, containerized applications and development	https://www.docker.com/

Name	Description	Link
Kubernetes	open source system for automating deployment, scaling, and management of containerized applications.	https://kubernetes.io/
Rancher	open-source multi-cluster orchestration platform	https://www.rancher.com/
Letsencrypt	free, automated, and open certificate authority used for SSL certificates.	https://letsencrypt.org/
zap	Package for fast logging in golang	https://pkg.go.dev/go.uber.org/zap@v1.27.0
SonarQube	open-source platform developed by SonarSource for continuous inspection of code quality	https://www.sonarsource.com/products/sonarqube/
Codeclimate	system that helps incorporating fully-configurable static analysis and test coverage data into a development workflow.	https://codeclimate.com/
pgx	PostgreSQL driver with toolkit for GO.	https://github.com/jackc/pgx/v5
pq	postgres for Go's database package	https://github.com/lib/pq
go-sqlite3	sqlite3 driver for Golang	https://github.com/mattn/go-sqlite3

State of the system

This section will break down the current state of the system looking through mltiple components and their current status. Such approach allows us to provide sufficient report and locate section of the project which require more work. Before, lets show some general data about the application to get the idea of the traffic. MiniTwit application has processed **14,5 million** request during its up-time with with something above 1 million of reported errors. This makes 6% error rate.

1. Code Quality Analysis

SonarQube and CodeClimate were used to determine our code quality. Based on the last provided analysis from SonarQube our code seems to be secure with no security concerns. Reliability part of the code is prove to have a stable code base where most of the issues are related to other datetime variable interpretation as SonarQube is advicing to use. Maintainability sections show the most issues with 87 recorded. Our code has a lot of error print statements which can be changed into constants. This would make the maintability part of the code much easier.

To sumarize our code base would appriciate some minor adjustments but none of these crate a potential harm to our code stablity and readability.

2. Dependency scan

Projects utilizes 100 dependencies based from the dependency report made by Snyk where there are 3 dependensies currently vulnerable towards SQl injection. GitHub dependency report shows only 63 dependencies reporting similar issue regarding SQL injection vulnerability in some of the dependencies. GitHubs

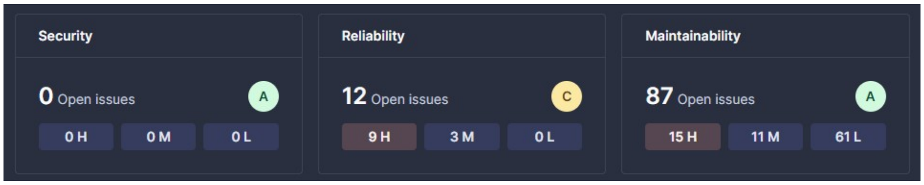


Figure 1: SonarQube general stats

denpenda bot created PR with needed update of the vulnerable dependencies which should solve the issue when merged into main.

3. Security Analysis

Static code analysis already showed us code quality when it comes to security point of view. In this field our projects shows good score. Application does not expose any vulnerable secrets which can be used to access any parts of the system. Our vulnerable information such as login details, SSH keys and other information are store either in GitHub secrets or we have an .enf file which each of us have on their local machine. Sharing such sensitive information between developer is done via USB drive or sharing them through BitWarden. Moreoever, when potential problem occurs GitHubs Advanced Security bot will create an alert and block open PR.

4. Test coverage

Application has a different sets of test - end-to-end, simulator tests, API tests as well as linters. Even with these test in place we do not have a 100% code coverage and some errors may slip through. Before every merge into main we did manual tests as well to catch bugs or other errors by hand. This method is not suitable for a long run. In the future projects would require some time into making more tets cases as well as different focus test sets.

5. Metrics, Logs and Dashboards

Application has a monitoring running on Grafana utilizing Loki and Mimir as the data sources. Monitoring an application can be a huge project itself when done properly and in detail. Currently our application monitors the basic data which we were using to estimate the application performance. We can divide the data into 2 sections. First one is focus on more technical parameters which help the developers to asses the current errors or any other potential problems. Main monitored factors: failed requests, request duration, database read/writes. Second section are data related to business which can be easily understood by non-tech person. This includes number of users registered, amout of requests, number of messages and overall appliation status.

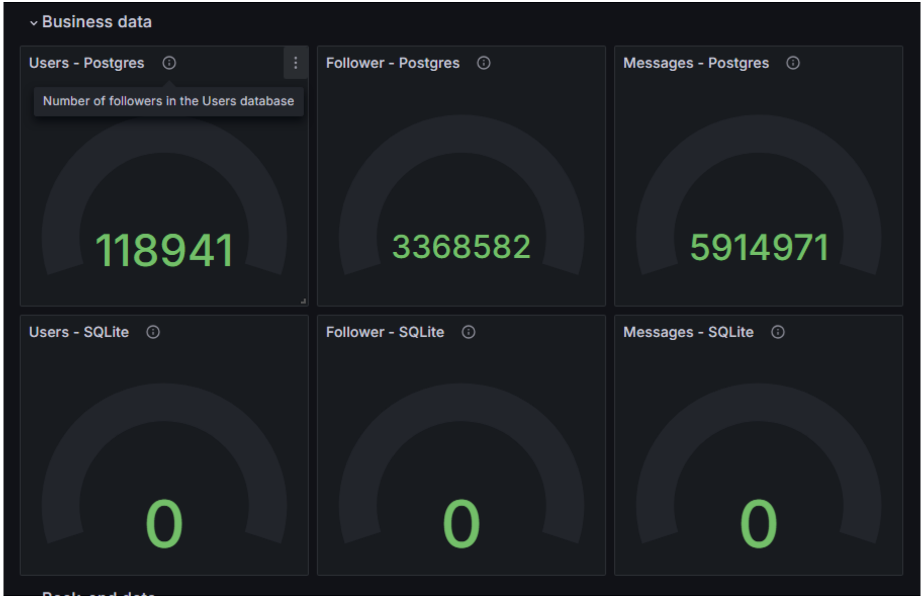
Application and all of its functionalities work as expected also under higher load of upcomming requests to the server. Applciatons uptime was satisfactory except one major outage happenign during *25/03 01:30* till *28/03 20:30* caused by a code error and was not spotted for few days. In real life scenario such outage would be unacceptable and would trigger alerts and other security tools to inform the developers about downtime.

Logging

How do you monitor your systems and what precicely do you monitor?

For monitoring we use Prometheus with Grafana. We do so by incrementing gauges or vectors whenever an event has succesfully occured. In the system we monitor a multitude of things, for business data we log: - We monitor the amount of users getting created. - The amount of new followers on the platform - Amount of new messages posted - The total amount of reads and writes made to the database between releases. Besides incrementing counters we also monitor back-end data: - The amount of failed database read-writes - Whether there is a connection to the database - Succesful HTTP requests

Monitoring these gives us an insight to the extend of traffic passing through our API. For ease of access to the monitored data and for visualization, the group uses Grafanas dashboards, see //



⌄ Business data

Users - Postgres ⓘ

Number of followers in the Users database

118941

Follower - Postgres ⓘ

3368582

Messages - Postgres ⓘ

5914971

Users - SQLite ⓘ

0

Follower - SQLite ⓘ

0

Messages - SQLite ⓘ

0

What do you log in your systems and how do you aggregate logs?

We log every error that happens during any database request. These are written through *zap*. The setup is such that the individual error logs are collected by logtail. Logtail then sends it to a Loki database that handles aggregation of the logs. The logs are visible through the Grafana Dashboard *Error Logs*.

It is important to note that we, due to time constraints, did not migrate our logs when moving to Kubernetes. The old logs are still hosted on a droplet

Lessons learned

Biggest Issues

Reflection

Conlusion

Appendix