MSc_group_c

ITU MiniTwit Report Skeleton

May 2024

1 Introduction

2 System perspective

3 Process Perspective

3.1 CI/CD (GitHub Actions)

GitHub Actions was chosen based on its simplicity, familiarity, and pricing (free for public repositories). A motivating factor, was the suite of services supported natively in GitHub. Therefore:

- GitHub *Secrets & Variables* was used to store environment variables, and deployment keys..
- GitHub *Tags*, *Releases*, and *Artifact Storage* were utilized, in order to create a clean version history of our application.
- GitHub integrations, such as *Dependabot*, *SonarCube*, and *Webhooks*.

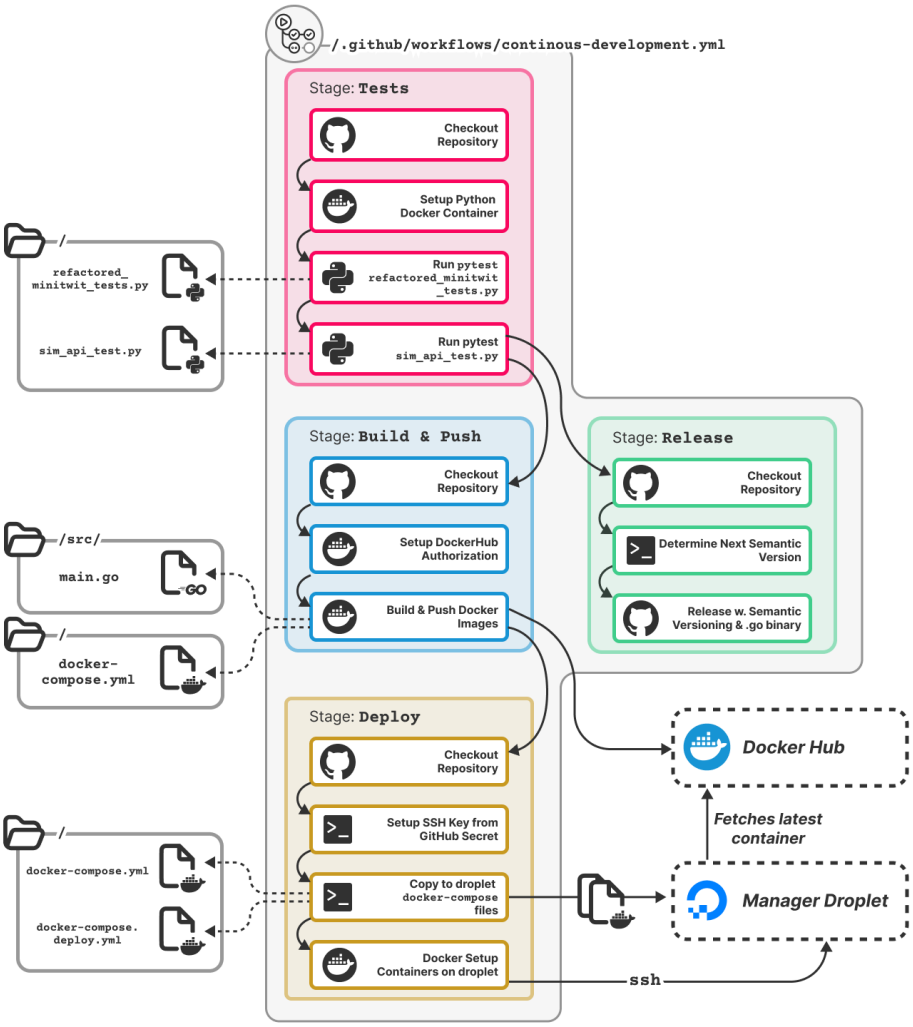
Write about: * *Linters* * *Python testing*

A total of **7** workflows are established, and can be found under `/.github/workflows/`. These are:

File	Purpose	Runs when
`continous-development.yml`	Primary CI/CD flow for continous integration & delivery. Consists of steps `Tests`, `Build & Push`, `Release`, and `Deploy`	Any changes to `main`
`codeql.yml`	Analyzes GoLang source code using CodeQL Analysis tool	Any push to `main`. Any pull-request to `main`. Once a week in cron-job.

File	Purpose	Runs when
`generate-report.yml`	Generates `report.pdf` from markdown files & images in `/report/`	Any changes to `/report/*` recursively.
`linter-workflow.yml`	Runs golangci-lint linter on GoLang source code. Configured by `/golangci.yml`.	Push to `main` or any action to pull-requests.
`pull-request-tests.yml`	Runs python tests.	Any actions to pull-requests
`test-deployment.yml`	Secondary CI/CD flow for continous integration & delivery against TEST-environment. Consists of `Tests`, `Build & Push`, `Deploy`	On push with tag `test-env*`.
`sonarcube_analysis.yml`	Analyses GoLang source code using SonarCloud.	On pull-requests to `main`.

Table: GitHub Action workflows employed.



> **Figure**: Visualization of `continous-development.yml`

3.1.0.1 Choice of CI/CD Since GitHub was chosen as the git repository management site, options such as GitLab CI/CD and BitBucket Pipelines were discarded as candidates, as they are specific to alternative git repository management sites. As such, the choice was between GitHub's native GitHub Actions or CI/CD systems agnostic to repository management sites.

Furthermore, commercial automation tools such as Azure DevOps and TeamCity were discarded due to the pricing and limitations of their free plans. An overview of the comparison performed can be seen in the table below.

CI/CD Tool / Platform	GitHub Actions	Jenkins	Azure DevOps	TeamCity (JetBrains)
Ease-of-use	Simple [1]	Medium [1]	*Undetermined*	*Undetermined*
Version Control	Native GitHub Integration [2]	Agnostic [2]	Agnostic [2]	Agnostic [2]
Hosting	Primarily cloud-based [2]	Self-hosted [2]	Cloud-based [2]	Cloud-based or self-hosted [2]
Pricing Model	Free for public repositories, tiered for private [2]	Open-source (MIT License), only cost is for hosting [2]	Commercial with a limited free tier [2]	Commercial [2]

Table: Comparison between CI/CD systems.

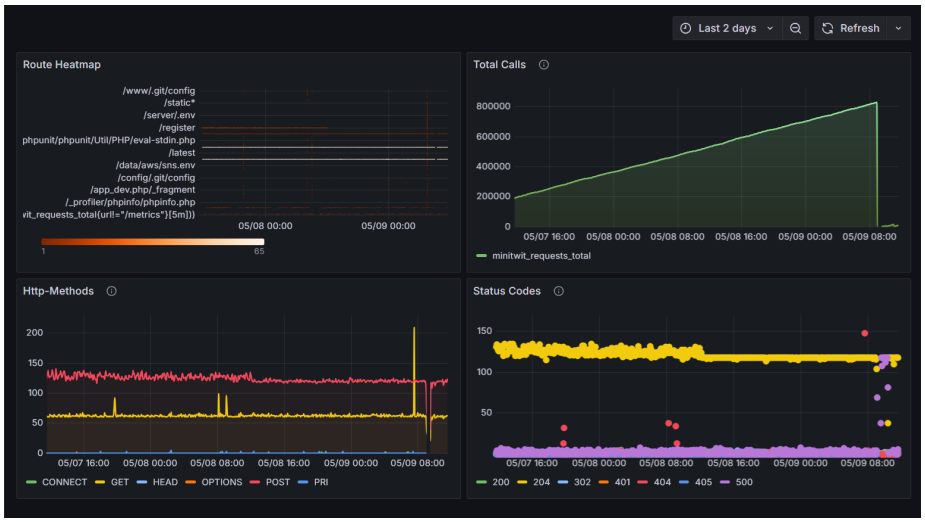
It was decided that time-to-production, in the case of establishing working CI/CD pipelines, was the biggest priority. As an alternative, the self-hosted automation system Jenkins was considered, but the perceived learning curve along with the self-hosted infrastructure setup [1] dissuaded it as the choice of CI/CD system.

3.2 Monitoring

- Prometheus as an Echo middleware, with additional custom made metrics.
  - TODO: make list of custom metrics.
  - Was chosen on the background of:
    - Demonstrated in Class
    - Easy integration with golang/echo via. middleware.
- Grafana
  - As of writing this the dashboards does not work due to swarm scaling. All pictures are from the day of the simulator stopping.
  - Users:
    - Admin user with password shared with the group.
    - Helge and Mircea specific login as described on Teams.
  - Was chosen on the background of:
    - Demonstrated in Class
    - Rich Visualization

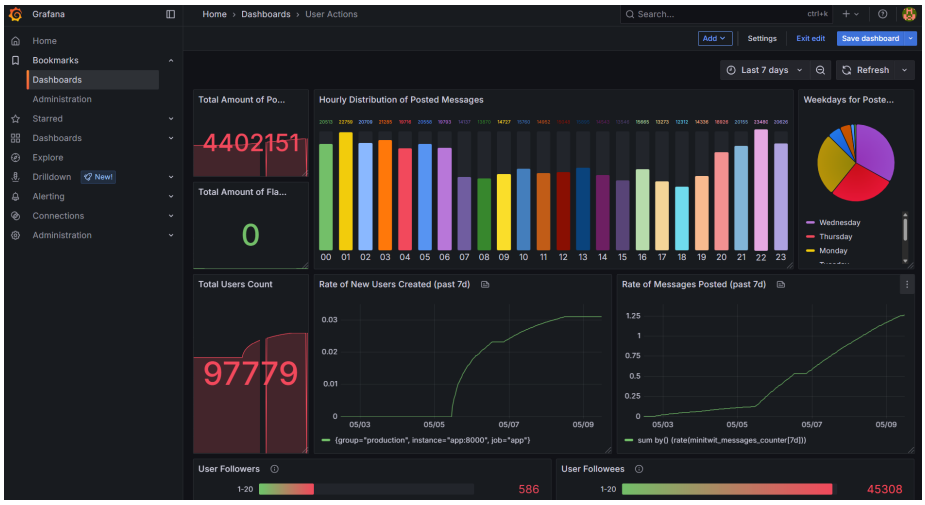
Whitebox Request and response monitoring dashboard:

Timeframe: last 30 minutes:

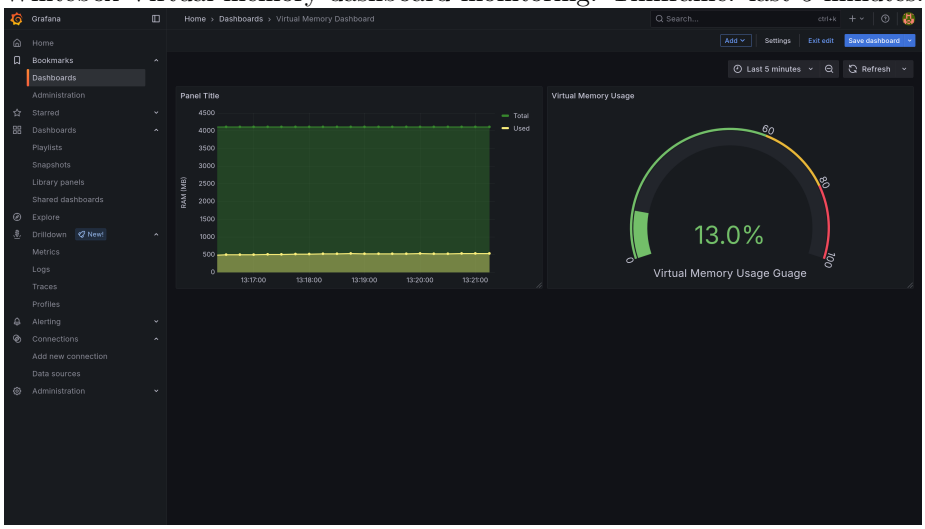


Timeframe: Last 2 days:

Whitebox User action dashboards monitoring: Timeframe: Last 7 days:



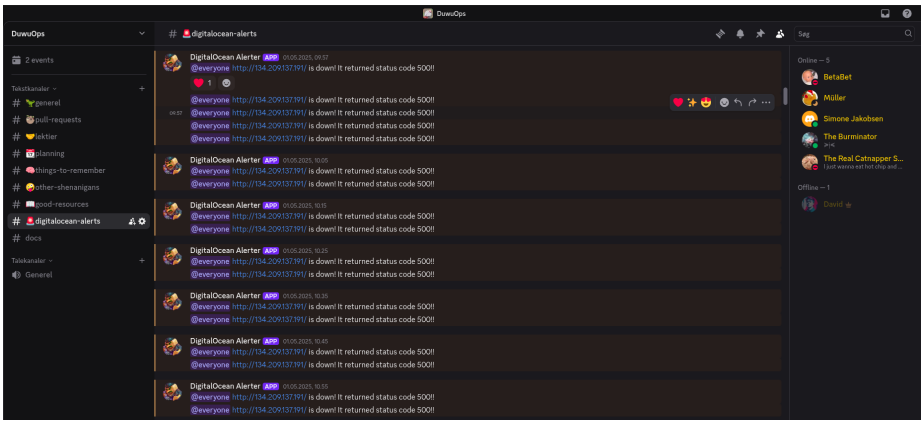
Whitebox Virtual memory dashboard monitoring: Timframe: last 5 minutes:



Black box user side error monitoring was given by the Helge and Mircea in form of the Status and Simulator API errors graf. We were encouraged to just use this as our client side error monitoring.

DigitalOcean provides some monitoring capabilities (Bandwidth, CPU usage, and Disk I/O). This did help to identify an attack. More on that [Insert refrence here]

An alert system was set up via a Discord bot that on the server via a cronjob that checks every 5 miniutes. If the application is not up it sends a Discord message and tags everyone on our group server.



3.3 Logging

- The ELK method was implemented but ultimatly scraped in favor of using loki/alloy that intergrate with Grafana which gather our logging and monitoring the same place.
- Practical Principles:
  - TODO: A process should not worry about storage
  - TODO: A process should log only what is necessary
  - TODO: Logging should be done at the proper level: Mention emoji use
  - Logs should be centralised: All logs can be found via Grafana->Drilldown->Logs

3.4 Strategy for scaling and upgrades

- Used docker swarm using docker stack so that we can leverage the docker compose setup that was already made. Some changes were made to accomidate the swarm set up. These are a network overlay, setting how many replicas per service, where nesecary setting where the service should be placed, and update configs.
- The update config for the minitwit application is set so that it updates one at a time. This is set as we only have two instances of minitwit and if an update fails we don't want more than one instance to be down. On failure we do a rollback.

3.5 AI use

Throughout the development process, all team members leveraged artificial intelligence tools to varying degrees and for diverse applications. The primary AI systems employed included ChatGPT, Claude, DeepSeek, and GitHub Copilot. Team members provided contextual information regarding code issues or implementation challenges, utilizing AI-generated responses as foundational guidance

for problem-solving methodologies rather than direct solution implementation. This methodology facilitated the identification of potential problem domains and remediation strategies while preserving critical assessment of AI-derived recommendations. In accordance with transparency requirements, AI tools have been formally acknowledged as co-authors in relevant version control commits where their contributions influenced the development process. (This paragraf was written using AI lol)

3.5.1 References

[1]: githubactions_vs_jenkins
[2]: 20_cicd_comparison