

SwipeRight

DevOps

Luka Spaninks
Semester 6
RB03

Version 1

Preface

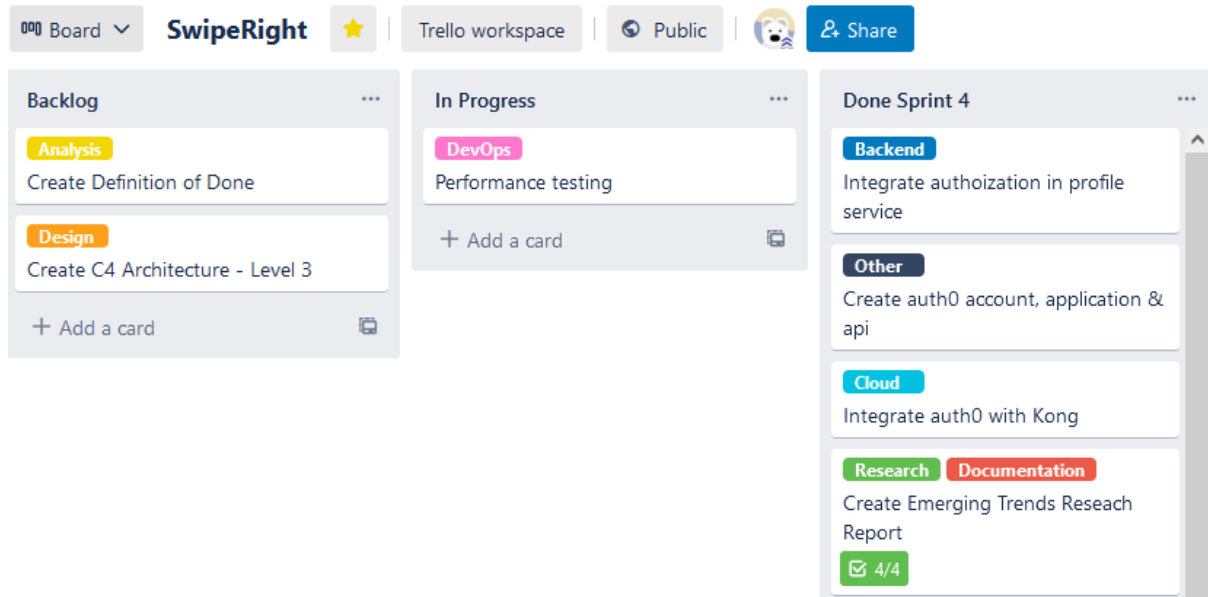
The complete DevOps phase of the project SwipeRight can be found in this document. This file will evolve over time and can always be expanded upon.

Table of contents

Methodology	3
Source Control.....	4
Automation	5
Building & Testing.....	5
Deployment.....	6
Monitoring.....	7
Security.....	8

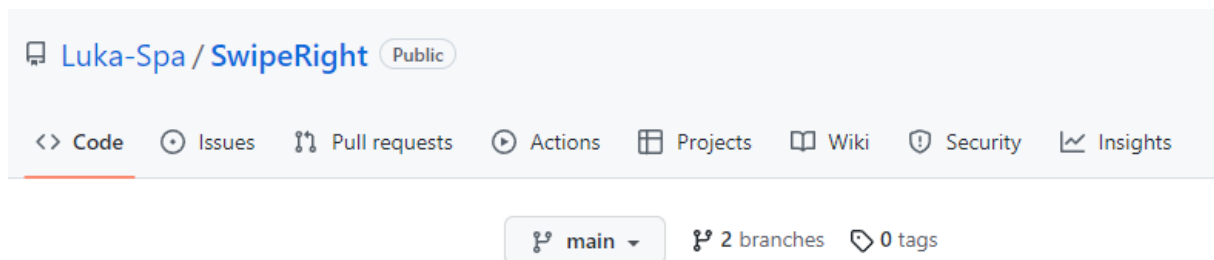
Methodology

The methodology I have chosen for this project is agile. Because I need to prove 8 learning outcomes the flexibility really helps. The project planning tool I have chosen is Trello, because I am already familiar with it.



Source Control

For source control I use the tool git in combination with a remote repository provided by GitHub. The repository is public and every item relevant to the project is stored there (monorepository). Because I am working alone on this project I decided to use git flow without the feature and hotfix branches for convenience.

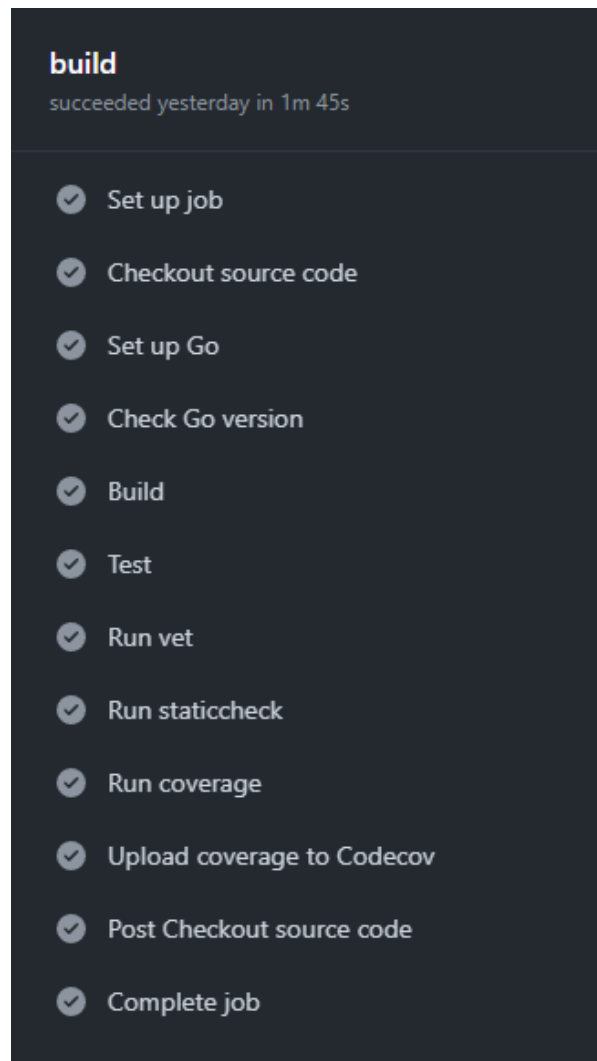


Automation

In order to automate several components of DevOps I am using GitHub actions.

Building & Testing

Whenever new changes are pushed to one of the branches a test and build job starts. The following is an example of what happens when the profile service is updated. The code gets tested, analyzed and the coverage gets uploaded to Codecov. Development can only be merged into main when it passes the building, tests and static code analysis



Deployment

Whenever changes are merged into the master or staging branch, all services are automatically pushed to Docker hub. The docker images are publicly available.

✓ Merge branch 'development' Docker #9

Summary

Jobs

✓ docker

docker

succeeded 28 minutes ago in 1m 20s

- ✓ Set up job
- ✓ Set up QEMU
- ✓ Set up Docker Buildx
- ✓ Login to DockerHub
- ✓ Build and push Profile
- ✓ Build and push Recommendation
- ✓ Post Build and push Recommendation
- ✓ Post Build and push Profile
- ✓ Post Login to DockerHub
- ✓ Post Set up Docker Buildx
- ✓ Complete job

The complete stack is currently deployed on a Kubernetes cluster on Azure. When the docker images are done being pushed to Docker hub, the services are updated in the cluster by the deployment workflow.

✓ Deployment Profile to Azure Deployment Profile to Azure #8

🏠 Summary

Jobs

✓ deploy

deploy

succeeded 1 minute ago in 29s

- ✓ Set up job
- ✓ Run actions/checkout@v3
- ✓ Run azure/login@v1
- ✓ Azure Kubernetes set context
- ✓ **Deploy to Kubernetes cluster**
- ✓ Post Run actions/checkout@v3
- ✓ Complete job

Monitoring

For monitoring I use Grafana in combination with Prometheus. Prometheus fetches metrics from my services which are then used by Grafana to visualize in dashboards.





Security

I took some actions to keep the development environment as secure as possible. For example whenever new code is uploaded a workflow starts it scans for the following:

- Outdated or insecure dependencies
- Insecure/vulnerable code
- Insecure log functions

The screenshot shows the Dependabot alerts interface. On the left, a sidebar contains navigation links: Overview, Security policy, Security advisories, Dependabot alerts (selected), and Code scanning alerts. The main content area is titled "Dependabot alerts" and includes a search bar with the text "is:open". Below the search bar, it shows "2 Open" and "0 Closed" alerts. The alerts are listed in a table with columns for Severity, Package, Ecosystem, Manifest, and Sort. The first two alerts are "Unhandled exception in gopkg.in/yaml.v3" with a "Moderate" severity. A footer note states: "Dependabot alerts surface known security vulnerabilities in some dependency manifest files. Dependabot security updates automatically keep your application up-to-date by updating dependencies in response to these alerts. Dependabot version updates can also help keep dependencies updated."

As already mentioned builds and tests will also be ran automatically.

I also use environment variables in both GitHub and my local environment to ensure private credentials won't be found by anyone malicious.

Furthermore, the gateway secures all microservices by checking for a valid access token (authentication) and every individual microservice is responsible for authorization.

The goal is to eventually use TLS in between any communication, but due to time constraints I have not started with this.