SwipeRight

DevOps

Luka Spaninks
Semester 6
RB03

Version 1

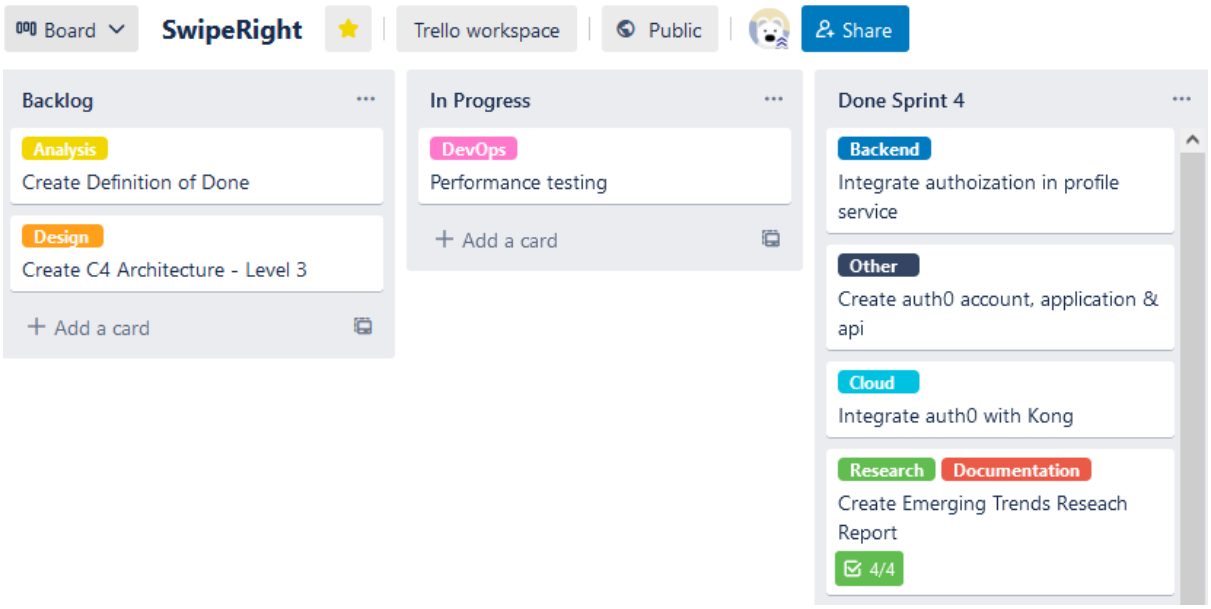
Preface

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

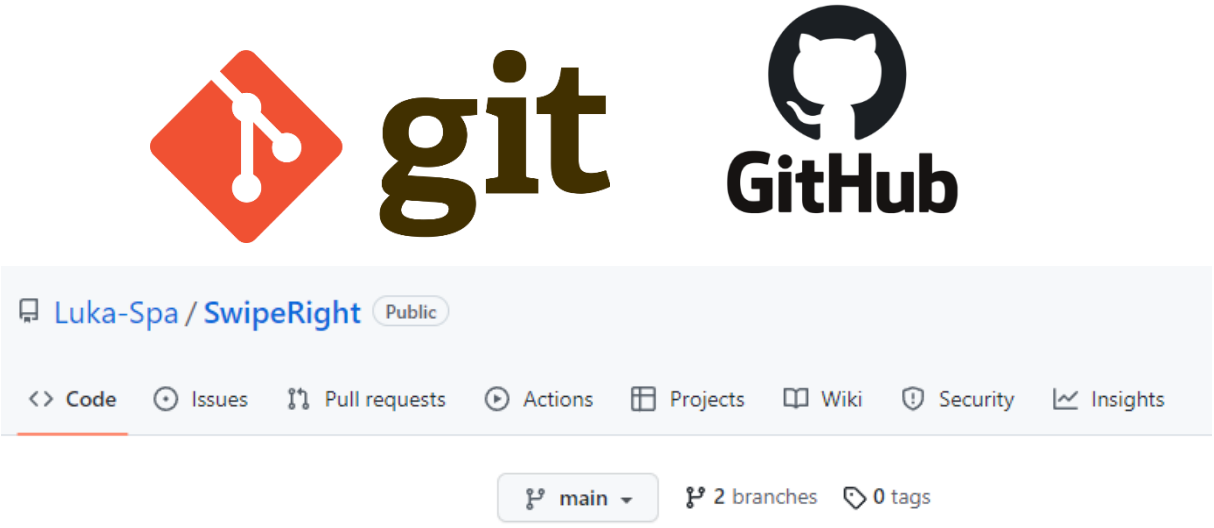
Methodology

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.

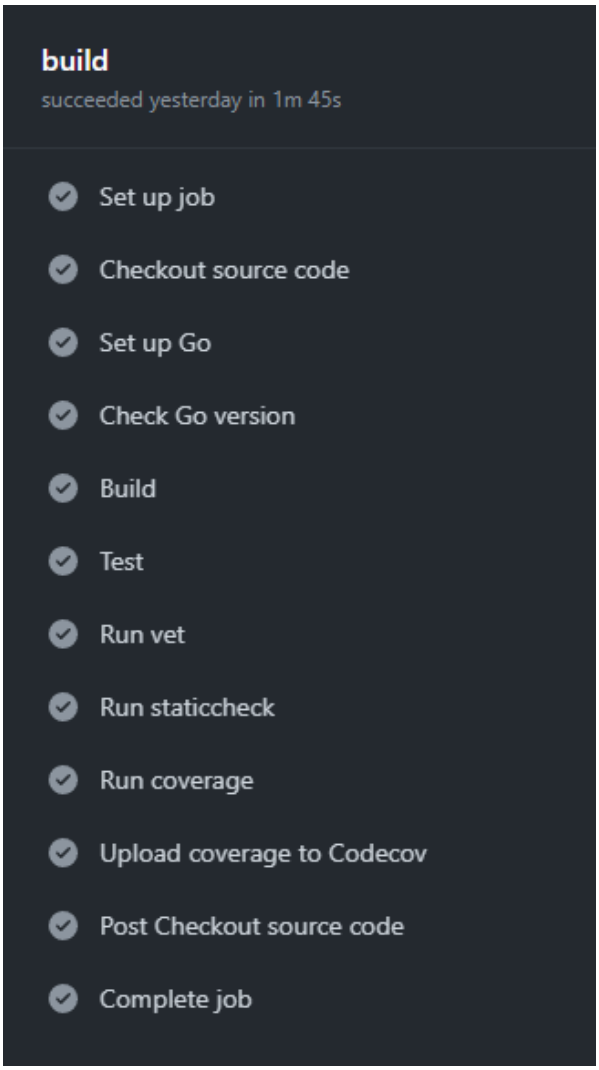


Automation

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

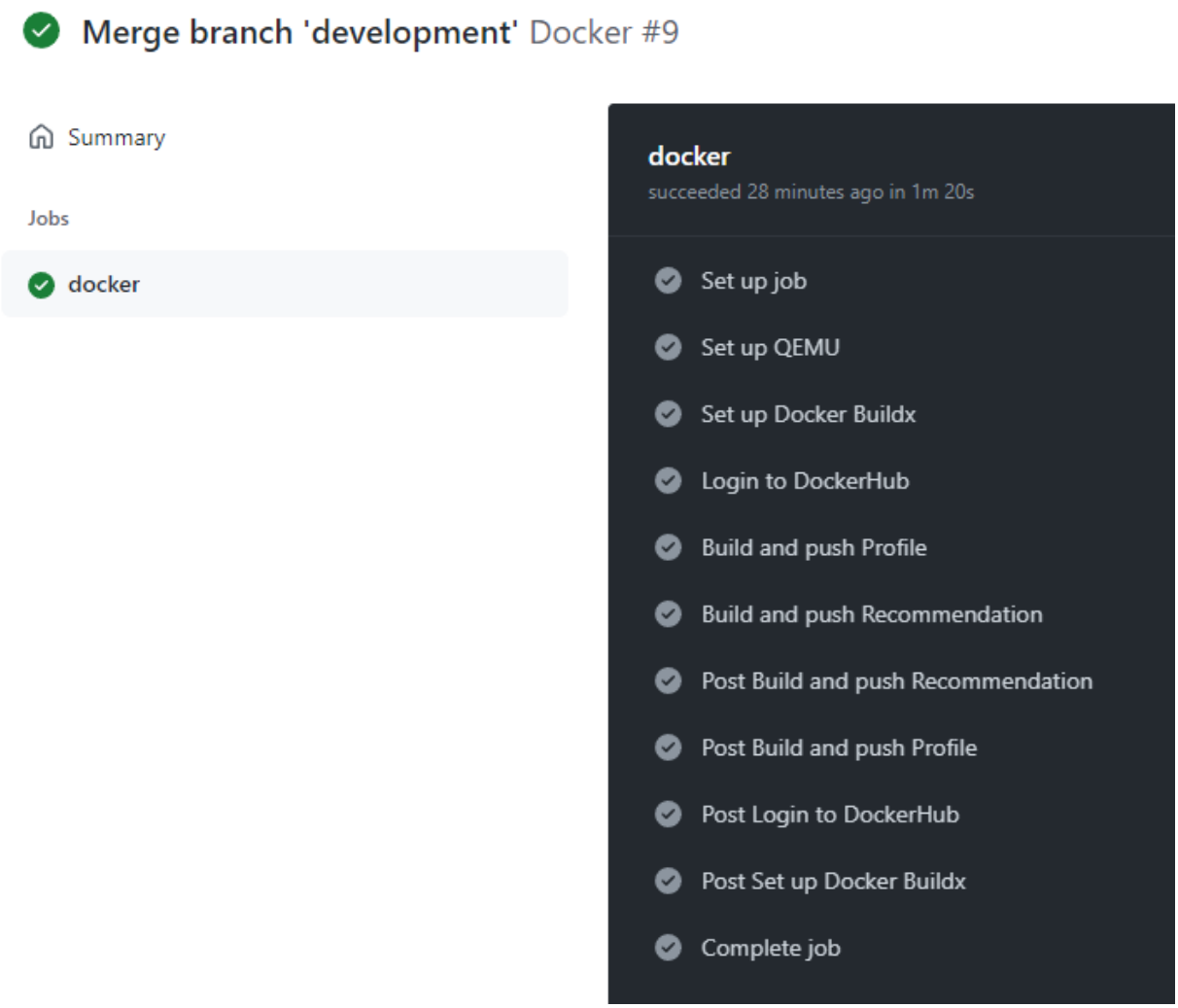
Building & Testing

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

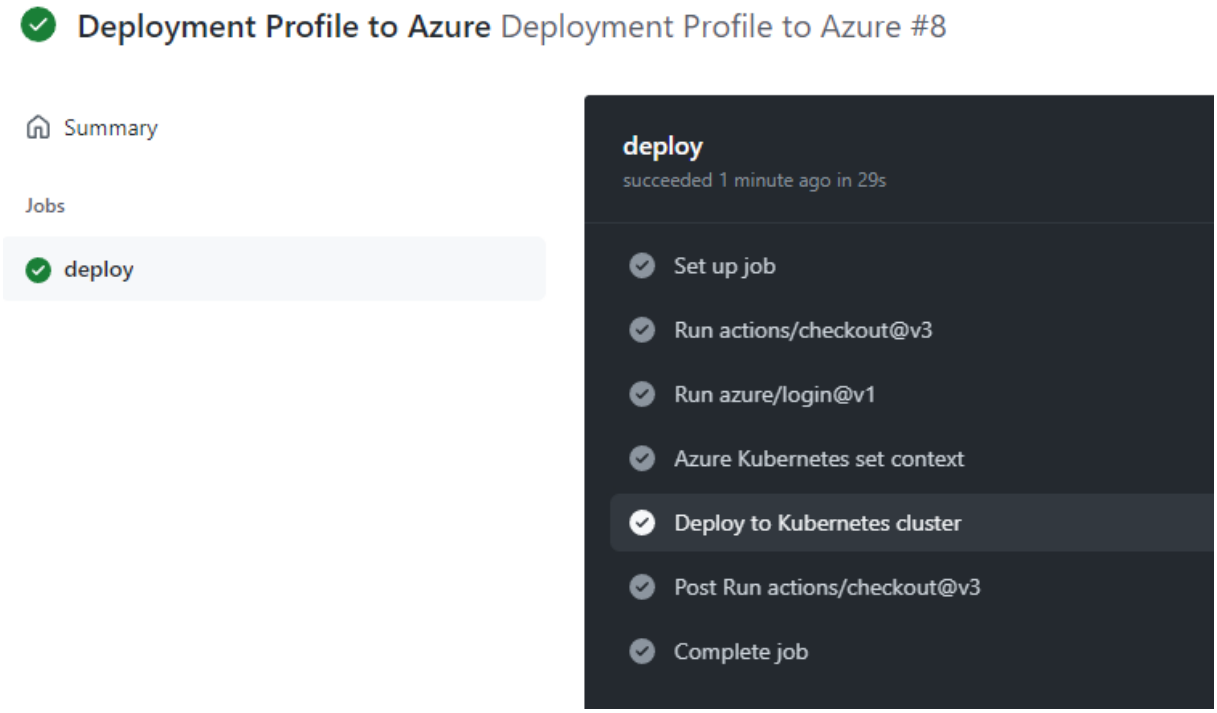


Deployment

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



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.



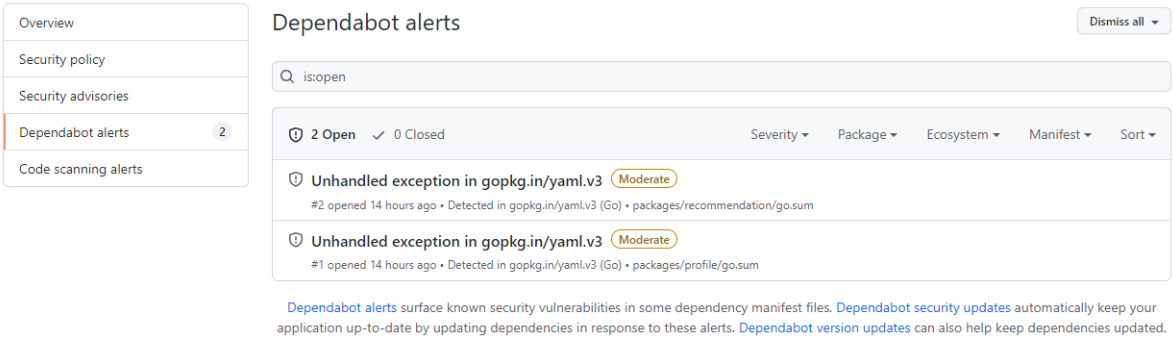
Monitoring

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



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.