Technische Hochschule
Brandenburg
University of
Applied Sciences

Semesterprojekt

Teammitglieder

Jack Ainsworth	20217393	jack.ainsworth@th-brandenburg.de
Duc Thuan Le	20238765	led@th-brandenburg.de

Kursinformation

Studiengang	Master Informatik
Lehrveranstaltung	Cloud Computing: Entwicklung und Betrieb
Dozent	Prof. Dr. Thomas Preuss Dipl.-Inform. (FH) Lars Gentsch

Datum 19.01.2024

Inhaltsverzeichnis

1 Einleitung

Das Projekt stammt aus dem 4. Semester aus dem Modul Typescript von Martin Weißbach und wurde von Jack Ainsworth geschrieben. Das Projekt ist eine in React geschriebene 4 gewinnt Anwendung. In diesem neuen Projekt ist das Ziel, diese Anwendung in die AWS Cloud zu befördern, über einen Docker Webserver build und eine Socket Anwendung (nicht neu geschrieben). Aufgrund der ungünstigen Lage, dass die Webanwendung die Socket URL benötigt, muss die Infrastruktur in 2 Stages gebaut werden. Die erste Stufe baut das Netzwerk und die Load Balancer und die 2 Stage ECS Services. Dazwischen muss .env angepasst werden und das Projekt neu auf Git gepusht werden, so dass ein neues Docker Image über die Pipeline entsteht, welches die Socket Server URL beinhaltet. Man kann ein neues Spiel nur starten, wenn die Anwendung sich zum Socket-Server verbinden lässt.
Wahrscheinlich würde kein professionelles Setup das so machen, aber dieses Modul dient dazu, die Grundlagen des Cloud Computings zu lernen und was meines Erachtens auch dadurch passiert ist und ich eine viel bessere Übersicht erlangen konnte von der AWS und git Welt.

2 Zielsetzung

Das Ziel des Projekts:

- Mindesten 3-4 AWS-Dienste müssen verwendet werden
- Eigenes VPC mit public und private Subnet, IPv6-ready (optional)
- Installation mittels IaC/Terraform
- Verwendung eines Build Tools für das Projekt
- Implementierung von 2 Testphasen mit Beispiel Tests (mindesten einen je Testphase)
- CI/CD-Pipelines
- Statische Code-Analyse und Security Scanning (GitHub/SonarCloud)

Infrastruktur Starten und das Build in der Cloud laufen lassen

1. terraform.tfvars

```
aws_access_key      = "ASIAXAN…GCWZWWX"
aws_secret_key      = "Z5f7Q160q…lc6waySfG"
aws_token           = "FwoG…cS0FY="


socket_container_port       = 3001
connect4_container_port     = 80

socket_container_image      = "ducthuanle/cc_socket-server:latest"
connect4_container_image    = "ducthuanle/cc_react-app:latest"

deploy_ecs_services = false
```

1. aws credentials setzen
2. die image namen müssen für dieselbe repository sein wie in der docker compose außer "`docker.io/`"

2. terraform ausführen

1. root ordner console: terraform init
2. root ordner console: terraform plan
3. root ordner console: terraform apply

Das startet alles aws Dienste außer die ecs_service_socket und ecs_service module

3. .env updaten und zu github pushen

```
REACT_APP_ALB_HOSTNAME=socket-ecs-lb-721208492.us-east-1.elb.amazonaws.com
REACT_APP_SOCKET_PORT=3001
```

1. output variable für socket_hostname kopieren
2. in REACT_APP_ALB_HOSTNAME einfügen
3. commiten und pushen
4. warten bis die: "CD Docker Image + DockerHub push" pipeline durchgelaufen ist

Die .env setzt die Socket Url im Code und diese muss auch im Build widergespiegelt werden, sodass die React App mit dem Socket kommunizieren kann. Deswegen müssen wir warten, bis die images über die CD-Pipeline gepusht wurden, bevor die ECS Infrastruktur im nächsten Schritt erzeugt wird.

4. die service infrastruktur deployen

1. terraform.tfvars: deploy_ecs_services= true setzen
2. terraform plan & terraform apply
3.

Hier wird die ECS Infrastruktur gestartet mit dem neusten Docker build und der richtigen Socket url.

5. LB Hostname der React app aufrufen

1. alb_hostname = "ecs-lb-571829938.us-east-1.elb.amazonaws.com"
2. auf starten drücken und dann warten bis die zum socket connected ist (siehe developer console)
3. room joinen

Anweisungen, um zu spielen. Kein Raum joinen bevor zum socket connected wurde.

Implementierung

2.1 AWS-Dienste

in der documentation.md gelistet.

2.2 Build Tools

NPM (Node Package Manager)
ist das Standard-Paketverwaltungssystem für Node.js-Anwendungen. In Verbindung mit React wird npm häufig verwendet, um Pakete und Bibliotheken von Drittanbietern zu installieren, die in React-Anwendungen genutzt werden können. Das genaue Build-Tool, was verwendet wird, ist es-build.

- npm install

Dieser Befehl installiert alle in der package.json-Datei aufgeführten Abhängigkeiten.

- npm install <Paketname>

Installiert ein bestimmtes Paket und fügt es zu den Abhängigkeiten in der package.json-Datei hinzu.

- npm start

Startet die Entwicklungsserver

- npm test

Der Befehl führt alle im Projekt definierten Tests aus.

- npm run build

Erstellt eine optimierte Produktionsversion der React-Anwendung

Node
Node.js ist eine serverseitige JavaScript-Laufzeitumgebung, die auf der V8-JavaScript-Engine von Google basiert. Diese Umgebung ermöglicht es, JavaScript-Code außerhalb des Webbrowsers auszuführen, insbesondere auf Servern

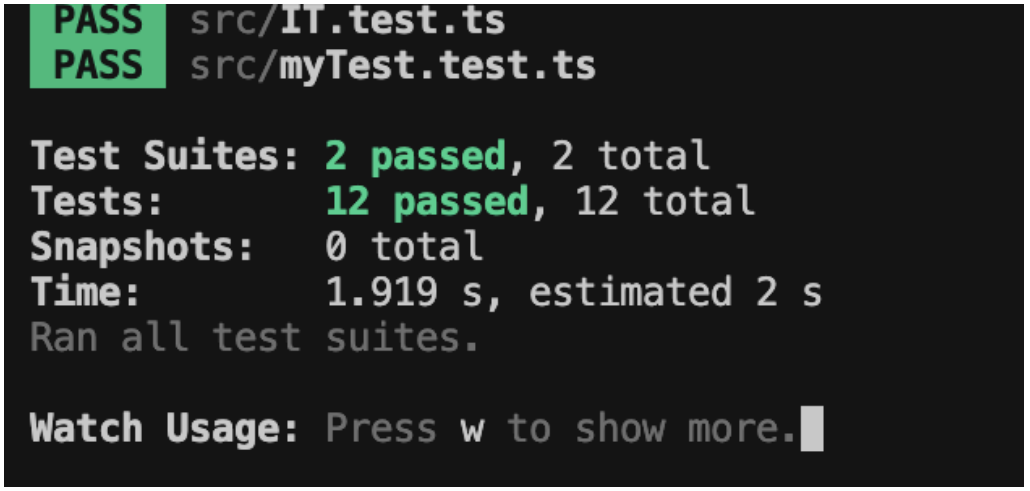
- node js-Datei

Um die Node.js-Datei auszuführen

2.3 Testphasen

Wir haben Unit Tests (src/myTest.test.ts) und Integration Tests (src/IT.test.tsx) implementiert.
Tests ausführen:

- npm test
- "a" eingeben
  Alle Tests ausführen



2.4 CI/CD-Pipelines

CI/CD (Continuous Integration und Continuous Delivery) ist ein CI/CD-Pipeline ist ein automatisierter Prozess, der Softwareentwicklung, Tests und Bereitstellung automatisiert.

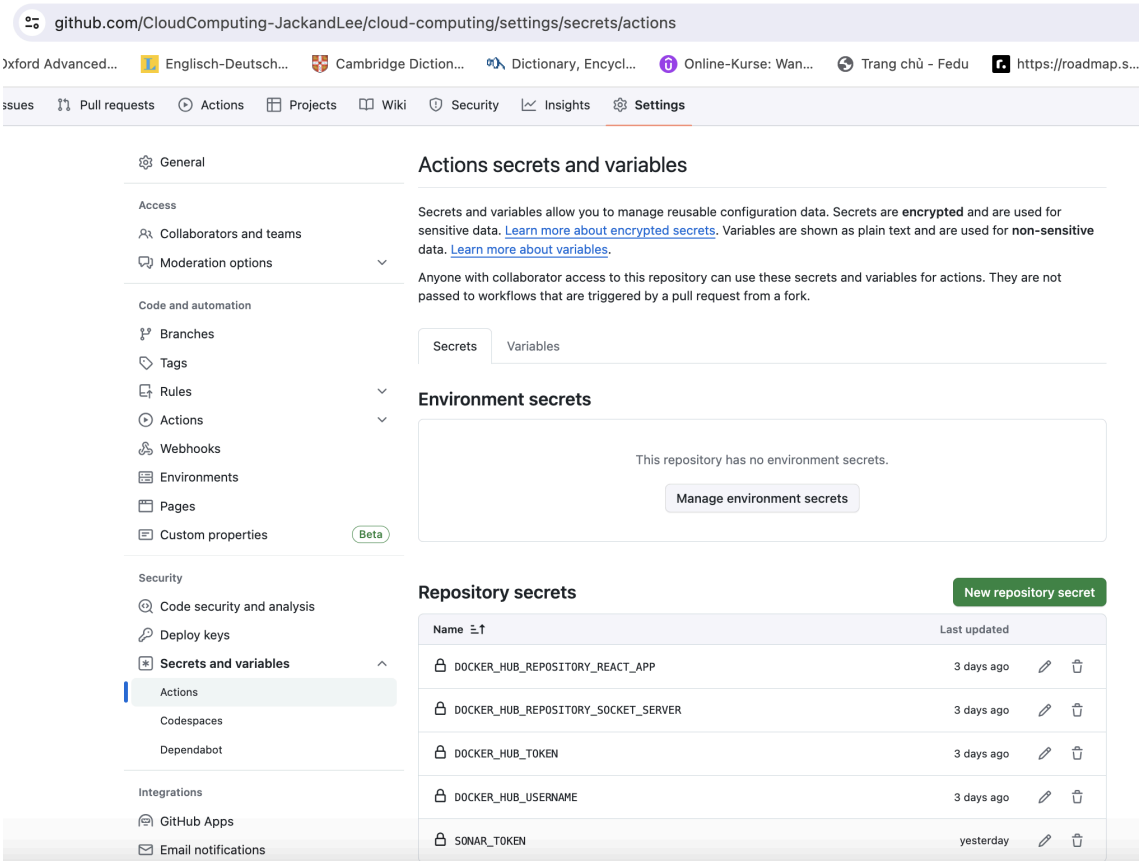
CI/CD-Pipeline mit GitHub Actions und Docker Hub einrichten

- Eine Workflow-Datei erstellen.

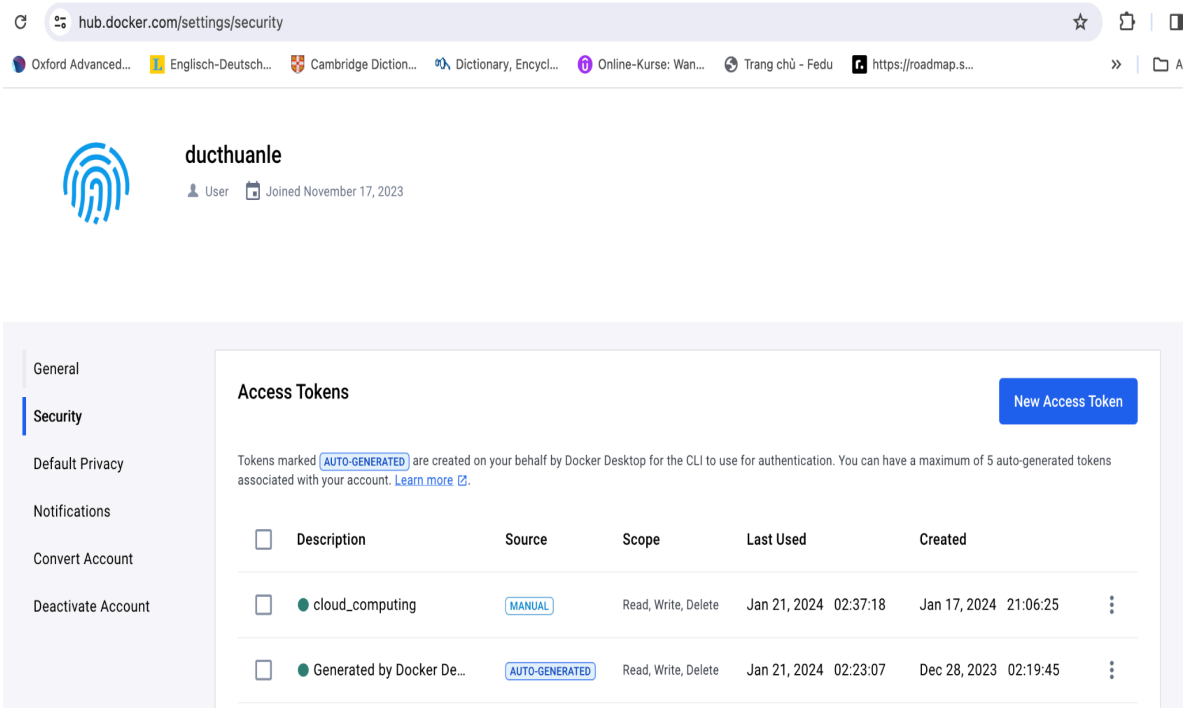
Eine neue Datei .github/workflows/docker.yml erstellen. Diese Datei definiert die GitHub Actions Workflow.

- Docker Hub-Zugangsdaten konfigurieren.

  GitHub Secrets anlegen



Access Token anlegen



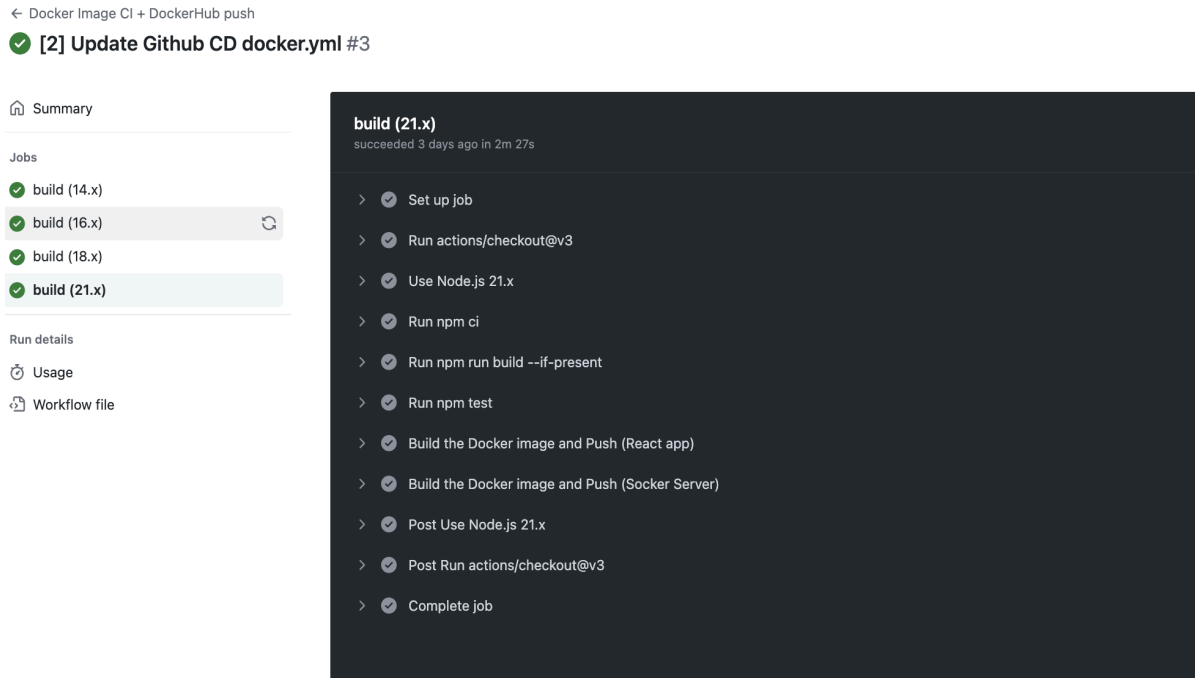
DOCKER_HUB_USERNAME: Docker Hub Benutzername

DOCKER_HUB_TOKEN: Dies ist ein Zugriffstoken, der für die Authentifizierung bei Docker Hub verwendet wird.

DOCKER_HUB_REPOSITORY_REACT_APP: Name des Docker Hub-Repositorys sein, das für die React Anwendung

DOCKER_HUB_REPOSITORY_SOCKET_SERVER: Name des Docker Hub-Repositorys für den Socket-Server

Ausführung der GitHub-Action



Docker Image nutzen

docker pull <docker.com-Zugang>/<Ihr DockerHub-Repository-Namen>:latest

Docker-Container lokal starten

docker run --rm -i -t -d -p 80:80 <Name des Docker–Image React App>

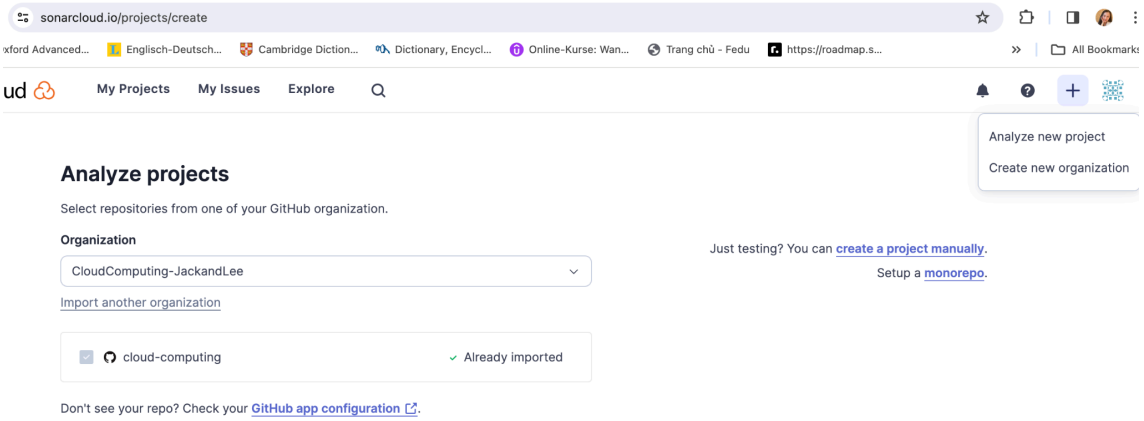
docker run --rm -i -t -d -p 3001:3001 <Name des Docker–Image Socket Server>

2.5 Statische Code-Analyse und Security Scanning

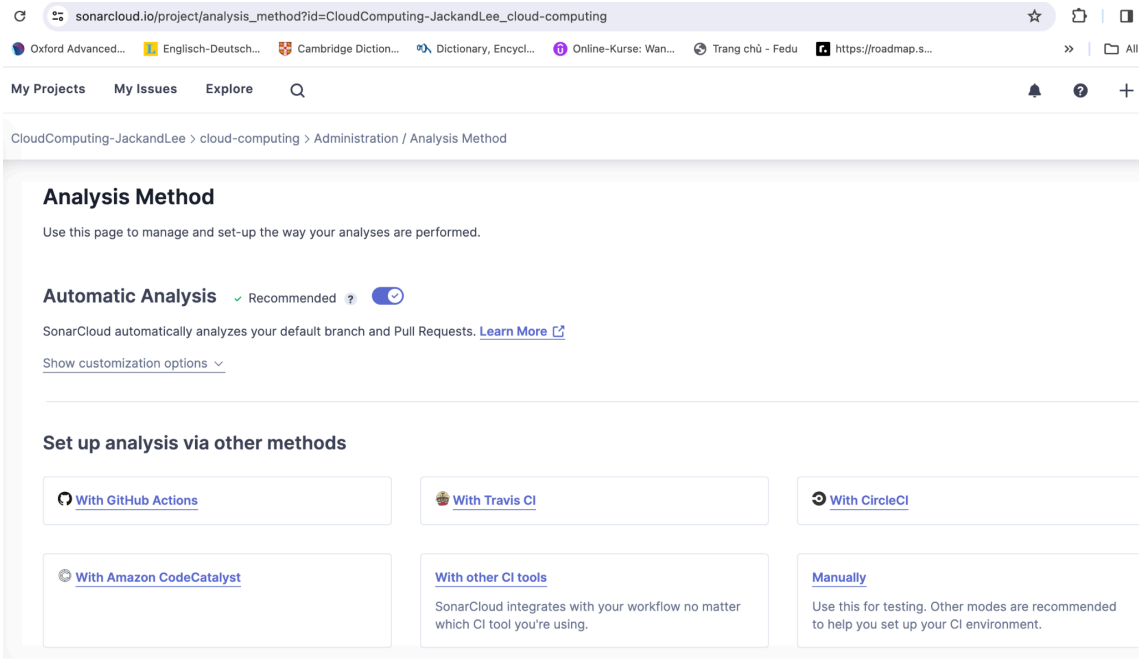
- GitHub mit SonarCloud verbinden

- Ein neues Projekt auf SonarCloud erstellen

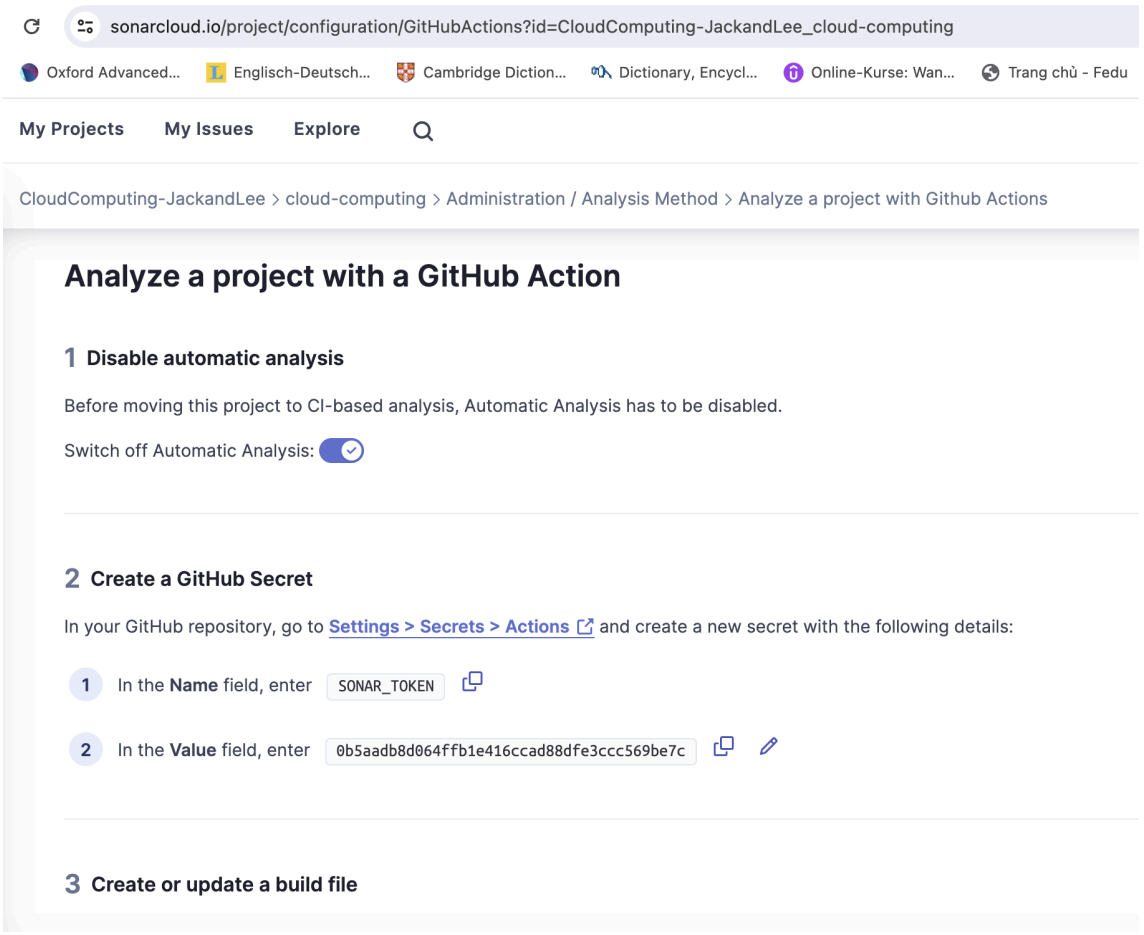
  sich bei SonarCloud anmelden.
  Auf "Create new project" klicken.
  Projektrepository (GitHub) Wählen und folgen den Anweisungen, um Projekt zu verknüpfen.



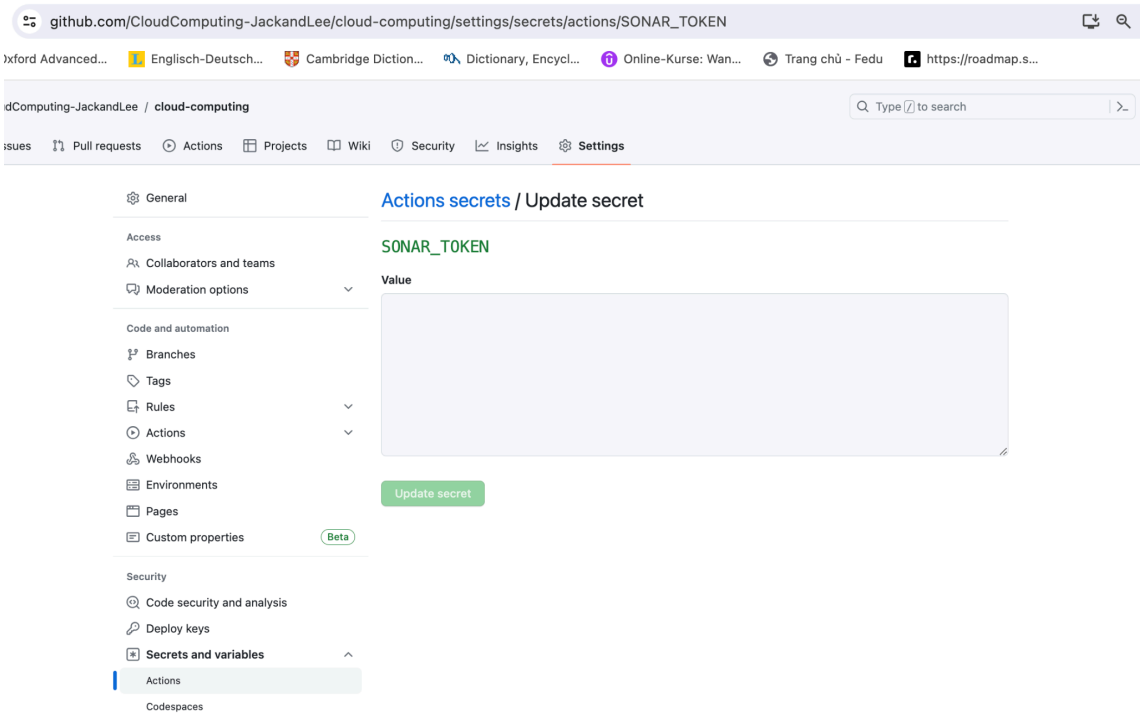
- GitHub-Actions-Integration auswählen



- Einen Token generieren



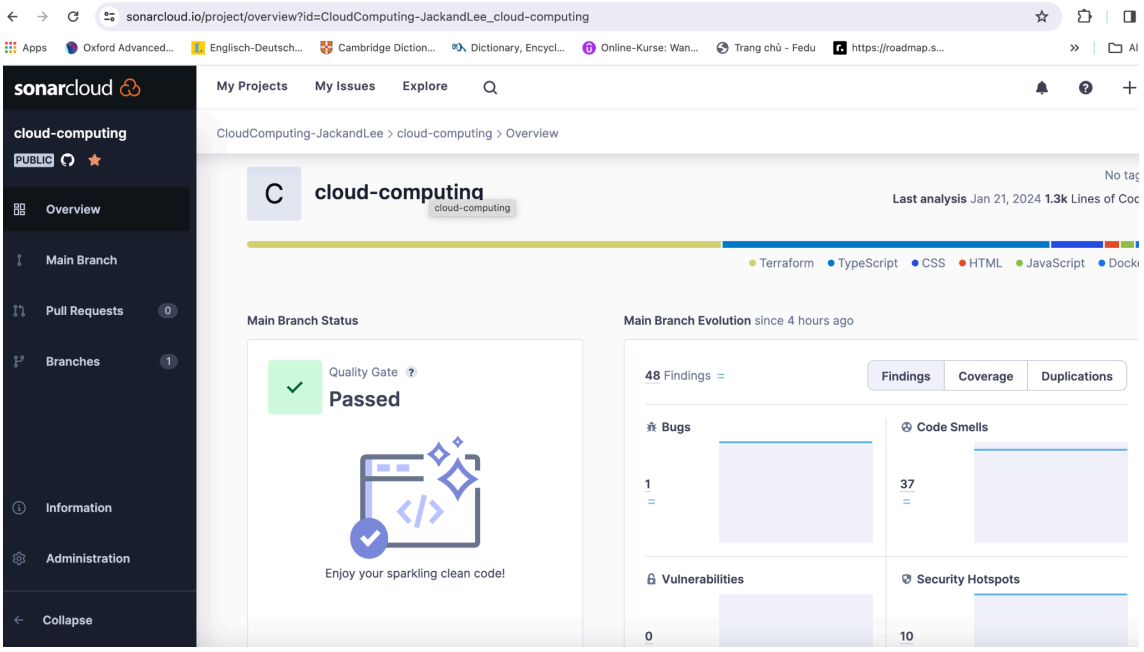
- GitHub Secrets anlegen



- Eine Workflow-Datei erstellen

Eine neue Datei .github/workflows/sonarcloud.yml erstellen. Diese Datei definiert die GitHub Actions Workflow.

- Ergebnis auf die Seite von Sonar Cloud



3 Fazit

- Testphasen
    - Unit Tests erfolgreich erstellen und ausführen
    - Integration Tests erfolgreich erstellen und ausführen
- Github Action CI/CD
    - Docker Image von React App und Socket Server erfolgreich erstellen und auf Docker Hub erfolgreich pushen.
    - Docker Image von React App erfolgreich pullen und laufen
    - Docker Image von Socket Server erfolgreich pullen und erfolgreich laufen
- Statische Code-Analyse und Security Scanning
    - Daten wird auf Sonar Cloud angezeigt
    - Sonar Cloud mit Github Action hat schon funktioniert

Um das Projekt zu erweitern, könnte man ECR einbinden und über Pipelines und Lambda Triggers automatisch über github action push die ECS tasks erneuern. Eine weitere mögliche Erweiterung wäre es, die Terraform Cloud zu verwenden und über github actions die Infrastruktur zu erstellen und zu erneuern, für mich persönlich wäre das genau ein neuer Account zu viel und bestimmt würde das auch keinen Spaß machen zu testen, weil man dafür noch github Secrets erstellen müsste den unsere aws Accounts laufen ja session basiert.
Die Terraform Struktur ist eine layer zu groß geworden aufgrund der Idee .env Variablen automatisch zu setzen aber das lässt sich nicht besonders schön integrieren und würde das Projekt noch sensibler machen, vielleicht werde ich die extra schicht noch entfernen (aber dafür hätte ich nur noch 1h).
Bei den Load Balancer sowie Service Modulen hätte ich nur ein Modul erstellen müssen und alles mit Variablen versehen, somit hätte ich die Module in main.tf mit anderen werten versetzen können und somit ein dutzend dateien sparen können, außerdem würde es dann nicht ganz so amateurhaft aussehen.