

GitHub: Actions, Deploy

Die Dokumentation für das Modul 324



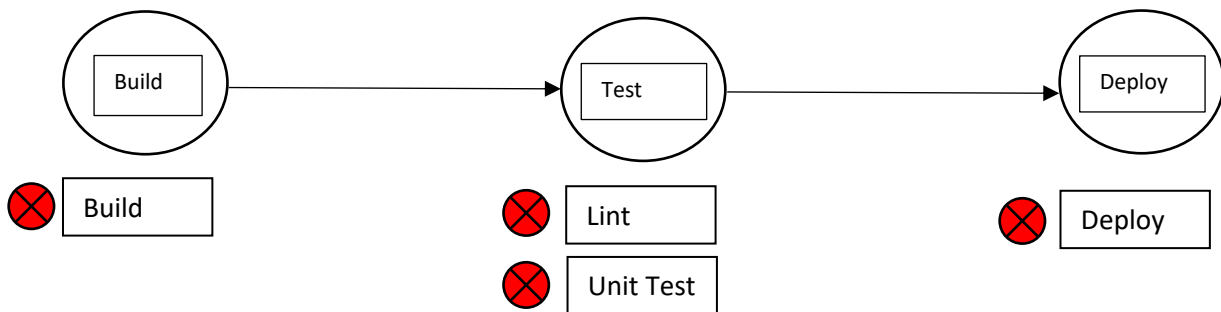
Gërvalla, Bytyqi, Selvaratnam & Ašanin

Vorbereitungen:

Einführung

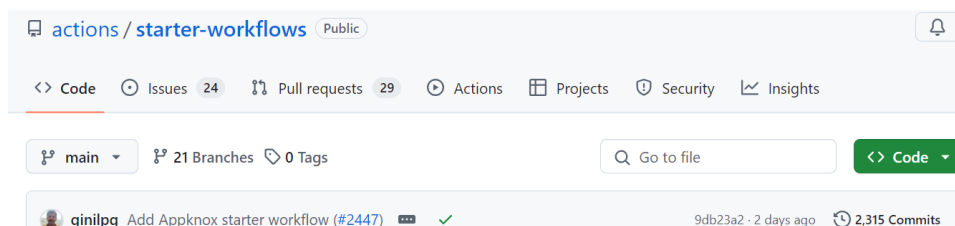
GitHub Actions ist ein Werkzeug von grossem Vorteil. Dank GitHub Actions, ist es den Entwicklern möglich, ihre Softwareentwicklungsprozesse zu automatisieren. Sie ist eine flexible Plattform die kontinuierliche Integration (CI) bis zu kontinuierlicher Bereitstellung (CD) bietet. Damit kann man Aufgaben in der Pipeline automatisieren wie das Testen, Bauen und Bereitstellen von Code.

Workflow Beispiel:



Vorgehen

1. (Bestehendes Projekt geklont und darauf navigiert.)

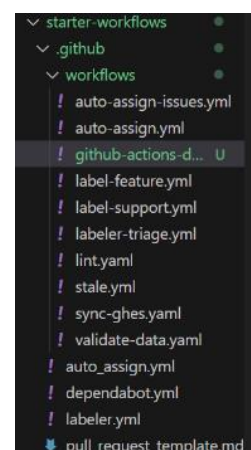


2. Ein File namens **github-actions-demo.yml** erstellt im Verzeichnis **.github/workflows**. Dieses YAML-File beinhaltet den Workflow unserer Pipeline.

3. Folgenden Code ins YAML-File schreiben:

```

1 name: GitHub Actions Demo
2 run-name: ${{ github.actor }} is testing out GitHub Actions
3 on: [push]
4 jobs:
5   Explore-GitHub-Actions:
6     runs-on: ubuntu-latest
7     steps:
8       - run: echo "🎉 The job was automatically triggered by a ${{ github.event_name }} event."
9       - run: echo "🔔 This job is now running on a ${{ runner.os }} server hosted by GitHub!"
10      - run: echo "📁 The name of your branch is ${{ github.ref }} and your repository is ${{ github.repository }}"
11      - name: Check out repository code
12        uses: actions/checkout@v4
13      - run: echo "📄 The ${{ github.repository }} repository has been cloned to the runner."
14      - run: echo "🚀 The workflow is now ready to test your code on the runner."
15      - name: List files in the repository
16        run: |
17          ls ${{ github.workspace }}
18      - run: echo "🍏 This job's status is ${{ job.status }}."
19
  
```



4. Die Änderungen commiten nach der Anleitung online.

5. **"Änderungen vorschlagen"** anklicken und dann Option wählen, dass einen neuen Branch für diesen Commit erstellt und einen Pull Request erstellt. Danach auf **"Propose Changes"** drücken.

Propose changes

Commit message

Create github-actions-demo.yml

Extended description

Add an optional extended description..

☐ Commit directly to the main branch

☒ Create a new branch for this commit and start a pull request

[Learn more about pull requests](#)

Neues Wissen

Im Rahmen dieser Aufgabe haben wir mehr über die Funktionsweise und den Nutzen von CI/CD mit GitHub Actions, React und Docker Hub gelernt.

- **GitHub Actions:** GitHub Actions ist ein Tool, mit dem man automatisierte Abläufe erstellen kann, die auf Ereignisse wie Code-Änderungen reagieren. Es hilft dabei, den Prozess der kontinuierlichen Integration und Bereitstellung zu automatisieren.
- **Continuous Integration (CI):** CI sorgt dafür, dass jeder Code, der in das Projekt eing检echt wird, automatisch getestet wird, um sicherzustellen, dass keine Fehler in den bestehenden Code eingefügt werden.
- **Continuous Deployment (CD):** CD geht einen Schritt weiter und stellt den Code nach erfolgreichen Tests automatisch in der Produktionsumgebung bereit.
- **React:** Durch die Integration von React in den CI/CD-Prozess wird jede Änderung an der Benutzeroberfläche automatisch gebaut und getestet, bevor sie veröffentlicht wird.
- **Docker Hub:** Docker Hub ist eine Plattform, auf der Container gespeichert und verwaltet werden. Mit Docker Hub und GitHub Actions können Container automatisch erstellt und bereitgestellt werden, was den Bereitstellungsprozess einfacher und schneller macht.

Dieses Wissen hilft uns, Entwicklungsprozesse zu automatisieren und effizienter zu gestalten.

Aufgabe 1: CI-CD, GitHub, React, Docker Hub

Abhängigkeiten

Um diesen Vorgang Schritt für Schritt auszuführen können braucht es folgende Abhängigkeiten, die man im Vorhinaus besitzen muss:

- Docker installiert
- Node.js installiert

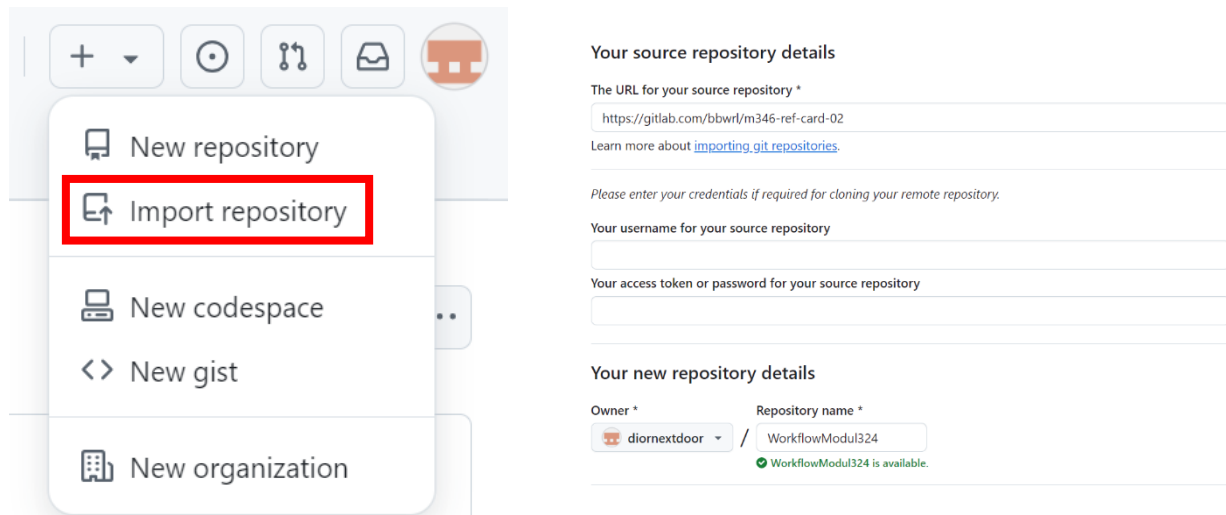
Mit folgenden Commands kann man testen, ob man die beiden Abhängigkeiten korrekt installiert, hat:

- docker --version
- node --version

Umsetzung

1. Wechsel GitLab auf GitHub

Wir haben das Repository [Rinaldo / m346-ref-card-02 · GitLab](#) auf GitLab bekommen. Der erste Schritt ist es, dieses Projekt auf GitHub zu importieren, damit wir dann basierend auf das unseren Workflow bauen können und mit Hilfe von GitHub Actions ausführen können. Für das gehen wir auf GitHub und dann auf das "Plus Icon" und wählen "Import repository".



Your source repository details

The URL for your source repository *

[Learn more about importing git repositories.](#)

Please enter your credentials if required for cloning your remote repository.

Your username for your source repository

Your access token or password for your source repository

Your new repository details

Owner *

Repository name *

WorkflowModul324 is available.

2. Projekt klonen und Struktur

Nach einiger Zeit sollte das Projekt fertig geklont sein und man kann es dann in einem Ordner klonen und auf das beliebige Betriebssystem öffnen (In diesem Fall VS Code). Das Projekt ist ein sehr simples React Projekt

```
PS C:\Users\TAAGED13> cd C:\Users\TAAGED13\CavuotiPipelineModul
PS C:\Users\TAAGED13\CavuotiPipelineModul> git clone https://github.com/diornextdoor/Modul324.git
Cloning into 'Modul324'...
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (22/22), done.
Receiving objects: 81% (18/22)
Receiving objects: 100% (22/22), 393.39 KiB | 18.73 MiB/s, done.
```

3. Erstellung des Dockerfile

Nun können wir im **src** Folder vom Projekt unser **Dockerfile** erstellen. Ein **Dockerfile** ist eine Textdatei die sogenannten Anweisungen beinhaltet, um ein Docker-Image zu erstellen. Darin wird zum Beispiel das Basisimage, Kopieren von Dateien oder Installation von Softwarepackages festgelegt. Unser Dockerfile haben wir wie folgt implementiert:

```
dockerfile X JS index.js ! c
dockerfile > ...
1 FROM node:16-alpine
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm install
5 RUN npm update
6 COPY . .
7 EXPOSE 3000
8 CMD ["npm", "start"]
```

FROM node:16-alpine: Hier wird das sogenannte Basisimage festgelegt, dass von dem aus dem neuen Image erstellt wird.

Das sind Anweisungen, welche die Befehle nach RUN ausführen wie **npm install** und **npm update**.

Hier wird Docker informiert, dass der Container etwas auf den Port 3000 bereitstellen muss.

4. Erstellung des ci.yml File

Als nächstes erstellen wir das ci.yml File. Dieses **YAML-File** ist eine Art Konfigurationsdatei für GitHub Actions der unseren CI-Workflow definiert. Das heisst, es wird definiert, was für Ereignisse im Repository ausgeführt werden. Unser YAML-File haben wir wie folgt implementiert:

```
dockerfile JS index.js ! ci.yml X
.github > workflows > ! ci.yml
1 name: CI
2
3 on:
4   push:
5     branches: [main]
6
7 jobs:
8   build:
9     runs-on: ubuntu-latest
10
11     steps:
12       - uses: actions/checkout@v3
13
14       - name: Build the Docker image
15         run: docker build -t diornjeri/modul324 .
16
17       - name: Log in to Docker Hub
18         run: docker login -u diornjeri -p ${ secrets.DOCKER_PASSWORD }}
19
20       - name: Push the Docker image
21         run: docker push diornjeri/modul324
22
```

Name: CI: Hier wird dem Workflow einen Namen "CI" gegeben.

On: Hier wird festgelegt, dass der Workflow bei jedem Push auf den Main Branch ausgeführt wird.

Jobs: Hier werden verschiedene Jobs / Aufgaben definiert, die den Workflow ausführen sollte. Hier kann man, wie wir es gemacht haben, einen Build-Job implementieren und weitere wie zum Beispiel einen Test-job.

Steps: Hier werden einzelne Schritte im Job ausgeführt. Wie man herauslesen kann, wird zum Beispiel das Docker Image gebuildet mit dem Tag diornjeri/modul324. Ebenfalls wird auf Docker Hub ein Login geführt, um dort dann Images hochzuladen. Für das haben wir einen GitHub Secret definiert namens DOCKER_PASSWORD (Zu dem mehr im nächsten Schritt). Auch das Pushen des davor gebauten Docker Images wird ausgeführt.

5. Erstellung des GitHub Secret Token

Wie im vorherigen Schritt erwähnt, braucht es einen definierten GitHub Secret Token. Zuerst müssen wir aber einen Token generieren auf Docker Hub. Dafür navigieren wir bei Docker Hub zu den Settings und dann zu Personal Access Tokens. Auf dieser neuen Seite klicken wir auf "**Generate new token**". Wichtig ist, dass wir den **password prompt** kopieren, weil wir den gleich benötigen. Danach kann man auf "**Back to access tokens**" gehen.

Create access token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access token description
pipeline token for module 324

Access permissions
Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

[Cancel](#) [Generate](#)

Access permissions
Read, Write, Delete

To use the access token from your Docker CLI client:

1. Run
`$ docker login -u diornjeri` [Copy](#)
2. At the password prompt, enter the personal access token.
`dckr_pat_scVpDtK0HnbcMAqXw6w-6LDkIFo` [Copy](#)

[Back to access tokens](#)

Description	Scope	Status	Source ⓘ	Created	Last Used
pipeline token for module 324	Read, Write, Delete	Active	Manual	Sep 25, 2024 at 20:32:21	Sep 25, 2024 at 20:34:33

Nun müssen wir zurück auf unser GitHub Repository navigieren und dann links auf der Nav Bar unter Secrets and variables und dann Actions gehen. Dort erstellen wir bei "**New repository secret**" ein neues Secret mit dem Namen DOCKER_PASSWORD

⌵

Secrets and variables

⌵

Actions

Codespaces

Dependabot

⌵

Environment secrets

⌵

This environment has no secrets.

Manage environment secrets

Repository secrets

New repository secret

Name ⌵⬆

Last updated

🔒 DOCKER_PASSWORD

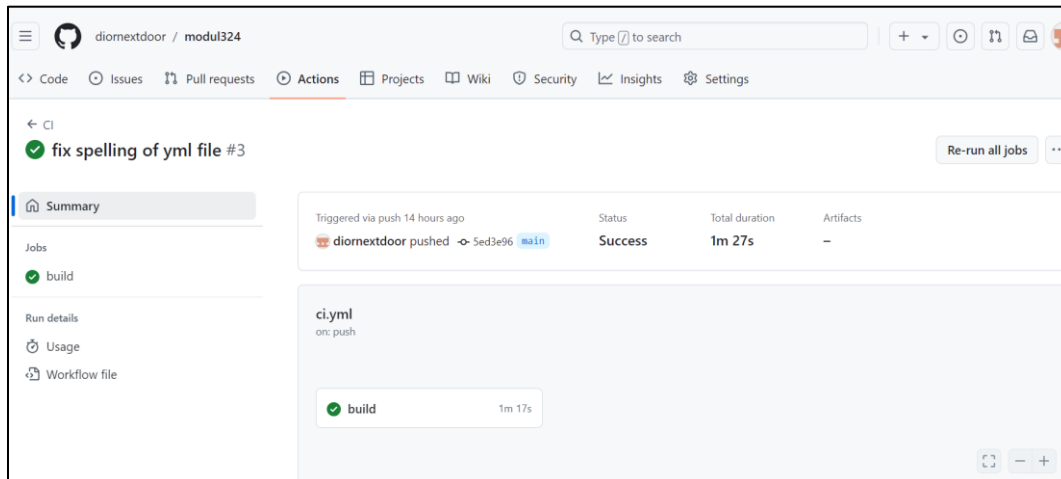
14 hours ago

✎

🗑

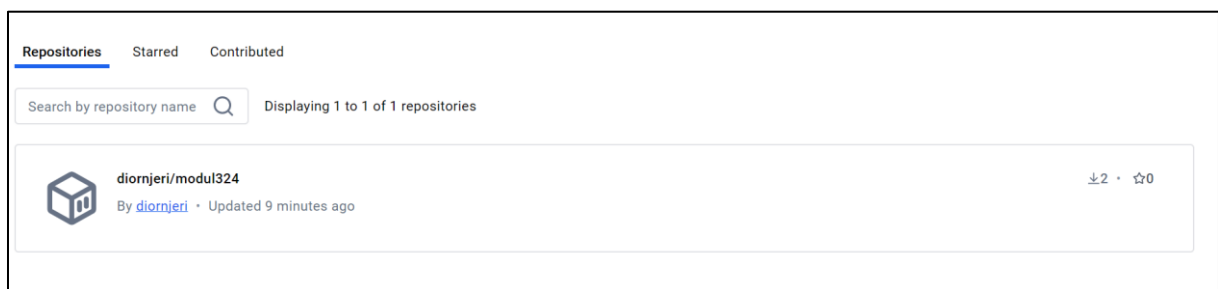
6. Pushen und Auslösen des Workflows

Wenn wir nun all diese Schritte gemacht haben, können wir zurück auf das Projekt auf VS-Code navigieren und unsere Changes (Erstellung Dockerfile und YAML-File) **adden**, **commiten** und **pushen**. Der Push löst damit unseren Workflow automatisch aus und man kann auf GitHub unter unserem Repository und im Abschnitt Actions den ganzen Workflow Prozess beobachten.



7. Docker Image auf Docker Hub

Da wir in unserem Workflow auch den Push des Images auf Docker Hub implementiert haben, sollte es bei einem erfolgreichen Bau der Pipeline dort auffindbar sein.



Zusatzaufgabe: Best practice:

Häufige, kleine Commits: Kleine, regelmässige Änderungen erleichtern die Fehlersuche und reduzieren das Risiko, bestehende Funktionen zu beeinträchtigen.

Einfache, modulare Workflows: Halte Workflows einfach und unterteile sie in klar definierte Jobs wie Bauen, Testen und Bereitstellen.

Matrix-Builds nutzen: Teste den Code in verschiedenen Umgebungen (z.B. Betriebssysteme und Software-Versionen) für umfassendere Abdeckung und Stabilität.

Sensible Daten schützen: Verwende GitHub Secrets für vertrauliche Informationen wie API-Schlüssel, um unbefugten Zugriff zu verhindern.

Monitoring und Logs: Nutze die detaillierten Protokolle von GitHub Actions zur Fehlerverfolgung und richte Benachrichtigungen für wichtige Ereignisse ein.

Versionierung der Workflows: Halte Workflow-Dateien unter Versionskontrolle, um Änderungen nachverfolgen und bei Bedarf frühere Versionen wiederherstellen zu können.