GitHub Actions Konfiguration (cicd.yaml)

Basics

- Clone folgendes Repo https://github.com/0xfabio/ts-wiederholung
- Navigiere auf folgenden Branch: start-here
- Lösche den .git Ordner (erstelle ggf. ein neues git repo git init)

Schritt 1: Erstellen einer GitHub Actions Pipeline

- Handlung: Füge eine neue GitHub Actions Workflow-Datei hinzu.
- Erklärung: Diese Datei definiert die Schritte, die bei jedem push-Event in deinem GitHub-Repository ausgeführt werden sollen. Es automatisiert den Prozess der Codeprüfung, des Baus, der Tests und der Bereitstellung.
- Code: ```yaml name: Wiederholung Techstarter on: [push] jobs: Deploy-App: runs-on: ubuntu-latest steps: ...

```
### Schritt 2: Festlegen der Ausführungsumgebung
- **Handlung:** Spezifiziere, auf welcher Umgebung die Pipeline laufen soll.
- **Erklärung:** `runs-on: ubuntu-latest` weist GitHub Actions an, die neueste verfü
- **Code:**
```yaml
runs-on: ubuntu-latest
```

Schritt 3: Abhängigkeiten einrichten

- Handlung: Installiere AWS CLI und Terraform in der Pipeline.
- Erklärung: Die AWS CLI wird verwendet, um mit AWS-Diensten zu interagieren, während Terraform für die Infrastrukturautomatisierung benötigt wird. Diese Tools sind essenziell, um die Infrastruktur und die Anwendung zu verwalten.
- Code: ```yaml
- uses: actions/checkout@v4
- uses: unfor19/install-aws-cli-action@v1
- uses: hashicorp/setup-terraform@v2

```
### Schritt 4: AWS Credentials konfigurieren
- **Handlung:** Konfiguriere die AWS-Anmeldeinformationen für die Verwendung in der
- **Erklärung:** AWS-Anmeldeinformationen sind notwendig, um auf AWS-Dienste zuzugre
- **Code:**
```yaml
- run: mkdir -p ~/.aws/
- run: echo "$super_secret" > ~/.aws/credentials
env:
super_secret: ${{ secrets.AWS_CONFIG }}
```

Schritt 5: Terraform Befehle ausführen

- Handlung: Führe Terraform-Befehle in der Pipeline aus.
- Erklärung: Terraform wird verwendet, um die Infrastruktur auf AWS zu erstellen und zu verwalten. init, plan, und apply sind grundlegende Befehle in Terraform für die Initialisierung, Planung und Anwendung der Infrastrukturänderungen.

- Code: ```yaml
- run: terraform -chdir=terraform/ init -backend-config="bucket=${STATE_BUCKET}" env: STATE_BUCKET: ${{ secrets.TF_BUCKET }}
- run: terraform -chdir=terraform/ plan
- run: terraform -chdir=terraform/ apply -auto-approve

```
## Terraform Änderungen

### Schritt 6: Einrichten des Terraform Backends
- **Handlung:** Konfiguriere das Terraform Backend.
- **Erklärung:** Das Backend speichert den Zustand der Terraform-Infrastruktur. Hier
- **Code:**
```hcl
terraform {
    backend "s3" {
        key    = "terraform.tfstate"
            region = "eu-central-1"
            profile = "techstarter"
    }
}
```

Schritt 7: Änderungen am EC2-Modul

- Handlung: Füge Ressourcen für SSH-Key-Management in Terraform hinzu.
- Erklärung: Dies ermöglicht Terraform, ein SSH-Schlüsselpaar zu erstellen, das für den sicheren Zugriff auf EC2-Instanzen verwendet wird. Der öffentliche Schlüssel wird in AWS gespeichert, während der private Schlüssel

sicher für den SSH-Zugriff genutzt wird.

- Code: ```hcl resource "tls_private_key" "key" { algorithm = "RSA" }

resource "aws_key_pair" "aws_key" { key_name = "ansible-ssh-key" public_key = tls_private_key.key.public_key_openssh }

resource "aws_instance" "main" { ... key_name = aws_key_pair.aws_key.key_name ... }

```
Verstanden, ich werde jetzt die Schritte für die Einrichtung von SSH und die Ausführung de

### Ansible Konfiguration

### Schritt 8: SSH Key Setup für Ansible
- **Handlung:** Konfiguriere SSH-Keys für die Verwendung mit Ansible.
- **Erklärung:** SSH-Keys sind erforderlich, um eine sichere Verbindung zu den EC2-Instanz
- **Code:**
```yaml
- run: |
terraform -chdir=terraform/ output -raw private_key > ssh
tail -n +2 ssh > ssh_key
shell: bash
- name: Setup SSH
shell: bash
run: |
mkdir -p /home/runner/.ssh/
cp ssh_key /home/runner/.ssh/id_rsa
chmod 700 /home/runner/.ssh/id_rsa
```

Schritt 9: Ausführen des Ansible Playbooks

- Handlung: Führe das Ansible Playbook aus, um die Konfiguration auf den EC2-Instanzen anzuwenden.
- Erklärung: Ansible nutzt den konfigurierten SSH-Key, um sich bei den EC2-Instanzen anzumelden und die nötigen Einstellungen (wie die Einrichtung der Flask-App) durchzuführen. Die Ausführung des Playbooks wird von der GitHub Actions Pipeline gesteuert.
- Code: ```yaml
- name: Run ansible run: | service ssh status export ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook -vvv --private-key /home/runner/.ssh/id_rsa -u ubuntu -i ansible/inventory.ini ansible/playbooks/playbook.yml --extra-vars "ansible_ssh_private_key_file=/home/runner/.ssh/id_rsa"