Oppgave

Implementer kodekvalitetstester lokalt og sikkerhet i CI/CD

I denne øvingen skal du jobbe med å implementere automatisk validering av Terraform-kode ved hjelp av TFLint og Checkov. Du vil starte med å teste verktøyene lokalt, deretter automatisere prosessen med scripts, og til slutt integrere valideringen i en GitHub Actions workflow.

Start med å opprette to konfigurasjonsfiler i rot-mappen av ditt Terraform-prosjekt. Den første filen skal hete `.tflint.hcl` og inneholde Azure plugin (versjon 0.27.0 eller nyere), samt regler for navnekonvensjoner (snake_case), dokumentasjon av variabler og typesikkerhet. Utforsk gjerne flere regler som kan være av interesse. Den andre filen skal hete `.checkov.yaml` og skal konfigureres til å scanne Terraform-rammeverket med compact output format.

Når du har opprettet konfigurasjonsfilene, kjør `tflint --init` etterfulgt av `tflint` for å teste TFLint. Du skal se enten "0 issues found" eller en liste over warnings og errors. Kjør deretter `checkov -d . --framework terraform` for å teste Checkov. Du vil få en oversikt over hvilke sikkerhetsjekker som passerte og hvilke som feilet. Velg minst én warning eller error fra enten TFLint eller Checkov og fiks den i din Terraform-kode. Her er god praksisk å se dokumentasjon på hvordan en løste det, og gjennomfører en ny kjøring for å verifisere at feilen/advarselen er borte.

Neste steg er å automatisere valideringsprosessen med et script. Lag et script som automatisk kjører alle valideringssteg i rekkefølge: Terraform format check, Terraform validate, TFLint (med init) og Checkov. Kall scriptet `validate.ps1` hvis du er på Windows, eller `validate.sh` hvis du er på macOS eller Linux.

Test scriptet ved å kjøre det og verifiser at alle tester passerer. Introduser deretter en feil med vilje, for eksempel ved å lage en variabel med camelCase (som `variable "myVariable"`), og kjør scriptet igjen. Scriptet skal nå feile og vise tydelig hvilken test som ikke passerte. Fiks feilen og verifiser at scriptet kjører grønt igjen.

BONUSOPPGAVE: Nå skal du implementere samme validering i en GitHub Actions workflow. Opprett mappestrukturen `.github/workflows/` og lag en fil som heter `terraform-validation.yml`. Workflow-en skal trigges på pull requests mot main branch. Den skal inneholde følgende steg: checkout av kode, setup av Terraform (latest versjon), initialisering av Terraform, kjøring av Terraform format check, Terraform validate, setup og kjøring av TFLint, og til slutt setup og kjøring av Checkov. Bruk actions fra GitHub Marketplace som `actions/checkout@v4` og `hashicorp/setup-terraform@v3`.

Verifiser at alle steg blir grønne. For å teste at workflow-en faktisk fanger feil, opprett en ny branch kalt `test-validation`, introduser en feil (for eksempel ved å fjerne description fra en variabel), gjennomfør en ny pull request. Workflow-en skal nå feile og vise hva som er galt. Fiks feilen, og se at workflow-en blir grønn. Når pull request er grønn kan den merge til main som trigger CD for selve deploymenten.

Til slutt kan (ikke noe vi skal gjøre) en opprette en Workflow som gjennomfører tester på et kjørende miljø i Microsoft Azure (Drift detection – Kjør en nattlig terraform plan for å sjekke om det er forskjell på kode og infrastruktur i Azure). Disse testene vil så klart være påvirket av hvilke ressurser en har

kjørende, men for å gjøre testene så raske og ikke for kompleks, forholder vi oss fortsatt til en storage account og container.

Til slutt skal du reflektere over arbeidet:

- Hva er fordelen med å kjøre validering lokalt før du pusher kode?
- Hvorfor er det viktig å ha validering i CI/CD pipeline også?
- Hvilken type feil fanget TFLint versus Checkov?
- Hva skjer hvis noen i teamet ditt prøver å merge kode som ikke passerer valideringen?
- MERK! Dette er en veldig liten og enkel infrastruktur hvor en har en mappe som inneholder alle .tf-filene.
    - Forsøk å reflekter rundt hvordan det ville blitt med et større prosjekt, både med lokal testing og workflows, som har:
        - Flere moduler med .tf filer.
        - Flere stacks, typisk nettwork-stack, compute-stack etc..