Eksamen DevOps – Kandidat 34

Oppgave 1

- Opprettet main.tf, variables.tf, provider.tf, backend.tf og outputs.tf i "/infra-s3".
- Fulgte denne guiden for å installere terraform siden "terraform –v" var ukjent: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
- Jeg å comittet .terraform mappen med uhell, som var for stor for Github, men fikk heldigvis fikset det.
- Satte opp github actions som kjører ved push til Main fra "infra-s3/**" og ".github/**": https://github.com/andrerishovd/DevOps-Exam/blob/main/.github/workflows/terraform-s3.yml

Hvordan kjøre hvis man kloner repository:

- Sett AWS ID og secret i "repository secrets" som AWS_ACCESS_KEY og AWS_SECRET_KEY.
- Jeg valgte å sette default-variabler i variables.tf, så disse kan endres før man starter github actions. Eventuelt kan man teste å sette variabler ved å bytte ut strengene på blokkene under og lime dem inn i terminalen. Sørg for at prefix_midlertidig slutter på "/" for at det skal bli en mappe/prefix.

- **Plan:**
  terraform plan -var="aws_region=eu-west-1" -var="analysis_bucket_name=analysis-bucket-candidate-34" -var="prefix_midlertidig=midlertidig/" -var="transition_time=20" -var="expiration_time=60"

- **Apply:**
  terraform apply -var="aws_region=eu-west-1" -var="analysis_bucket_name=analysis-bucket-candidate-34" -var="prefix_midlertidig=midlertidig/" -var="transition_time=20" -var="expiration_time=60"

- Commit og push til main.

Oppgave 2 - A

Endret S3BucketName, key og secret i template.yaml.

Installerte sam og la til i .gitignore

Testet sam local start-api

Kjørte sam deploy –guided

Kan nå brukes med POST (f.eks i POSTMAN) til:

https://9yjo6eg50g.execute-api.eu-west-1.amazonaws.com/Prod/analyze

Gå i Body -> raw -> json

{

"text": "Amazon has announced a new AI model that rivals OpenAI's GPT series. Microsoft and Google are also expanding their AI research teams."

}

Resultatet kom i Postman, og man kan finne resultatet i bucketen “kandidat-34-data” -> “midlertidig/” på AWS (slettes etter 60 dager):

https://eu-west-1.console.aws.amazon.com/s3/object/kandidat-34-data?region=eu-west-1&prefix=midlertidig/comprehend-20251120-111650-d5f28591.json

Resultat åpnet fra lenken over:

```
{
  "timestamp": "2025-11-20T11:16:50.475820",
  "overall_sentiment": "NEUTRAL",
  "sentiment_scores": {
    "Positive": 0.07575350999832153,
    "Negative": 0.0002698953030630946,
    "Neutral": 0.9238883852958679,
    "Mixed": 8.819688082439825e-05
  },
  "companies_detected": [
    {
      "name": "Amazon",
      "confidence": 0.9954857230186462
    },
    {
      "name": "OpenAI",
      "confidence": 0.6516576409339905
    },
    {
      "name": "Microsoft",
      "confidence": 0.9944602251052856
    },
    {
      "name": "Google",
      "confidence": 0.9954705238342285
    }
  ],
  "method": "Amazon Comprehend (Statistical)",
  "text_length": 133
}
```


Oppgave 2 – B

Oppdatert sam-deploy.yml:

Hadde et problem som kostet meg mye tid. Deploy feilet fordi opprettelese av IAM regler ble stoppet fordi maksgrensen på 120 var nådd, måtte manuelt bytte fra EDGE til REGIONAL. Det er grunnen til at det ble laged 4 stacks med kandidatnummer 34.

Vellykket deploy action fra push til main:

https://github.com/andrerishovd/DevOps-Exam/actions/runs/19547696749

Velykket skip deploy action fra pull request fra en annen branch:
https://github.com/andrerishovd/DevOps-Exam/actions/runs/19548184646

Instruksjoner ved kloning:

- Sett AWS ID og secret i “repository secrets” som AWS_ACCESS_KEY og AWS_SECRET_KEY. (Samme som i oppgave 1)
- Sett følgende variabler under “env” i sam-deploy.yml
    - Stack_Name
    - AWS_region
    - S3Bucket_Name
- Commit og push til main
- Sjekk Actions for validering

Oppgave 3 - A

- Lagde Dockerfile i /sentiment-docker

Oppsett ved clone for å kjøre lokalt

- Åpne codespace fra main
- cd sentiment-docker
- Build lokalt
  docker build -t sentiment-docker .
- Fyll inn access key id, secret access key og s3-bucket navn i teksten under og kjør i terminalen:

docker run -e AWS_ACCESS_KEY_ID=<din aws access key id> \

-e AWS_SECRET_ACCESS_KEY=<din aws secret access key> \

-e S3_BUCKET_NAME=<ditt s3 bucket navn> \

-p 8080:8080 sentiment-docker

- Åpne en annen terminal og kjør:
  curl -X POST http://localhost:8080/api/analyze -H "Content-Type: application/json" -d '{"requestId": "test-123", "text": "NVIDIA soars while Intel struggles with declining sales"}'

- Sjekk din s3 bucket for å se at resultatet ble lagret

Resultat i S3 bucket “kandidat-34-data":


```
{
  "requestId": "test-123",
  "timestamp": "2025-11-20T22:01:03.291210628Z",
  "method": "AI-Powered (AWS Bedrock + Claude)",
  "model": "amazon.nova-micro-v1:0",
  "companies": [
    {
      "company": "NVIDIA",
      "sentiment": "POSITIVE",
      "confidence": 0.95,
      "reasoning": "The text uses 'soars' to describe NVIDIA's performance, indicating a positive sentiment."
    },
    {
      "company": "Intel",
      "sentiment": "NEGATIVE",
      "confidence": 0.95,
      "reasoning": "The text mentions 'struggles with declining sales' for Intel, indicating a negative sentiment."
    }
  ]
}
```

Oppgave 3 - B

Docker-build.yml:

https://github.com/andrerishovd/DevOps-Exam/blob/main/.github/workflows/docker-build.yml

Vellykket workflow:
https://github.com/andrerishovd/DevOps-Exam/actions/runs/19555471534

Tagging:

Når man prøver og feiler med docker får mange mange forskjellige versjoner når de bygges. Da kan det være fint å ha en kode man kan koble til commit-loggen til hver build for å finne ut hvilke endringer som skjedde for hver build. For å slippe å huske på hver kode er det også fint å ha en latest, for å vite hvilken man skal bruke når man nettopp har bygget.

Dockerhub repository navn og lenke:

jernkjeven/sentiment-docker

https://hub.docker.com/repository/docker/jernkjeven/sentiment-docker/general

Instruksjoner ved fork:

Opprett docker bruker om nødvendig:

https://hub.docker.com/

Lag docker token:

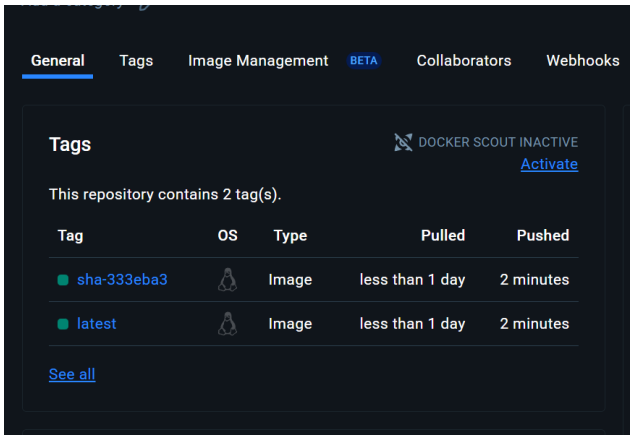
https://docs.docker.com/security/access-tokens/

Sett opp følgende “repository secrets”:

DOCKERHUB_USERNAME og DOCKERHUB_TOKEN

Lag en ny branch, skriv en liten kommentar i sentiment-docker og push til main. Da skal workflow kjøre, hvis den er vellykket vil builden komme opp i Docker Hub som vist i skjermbildet under:

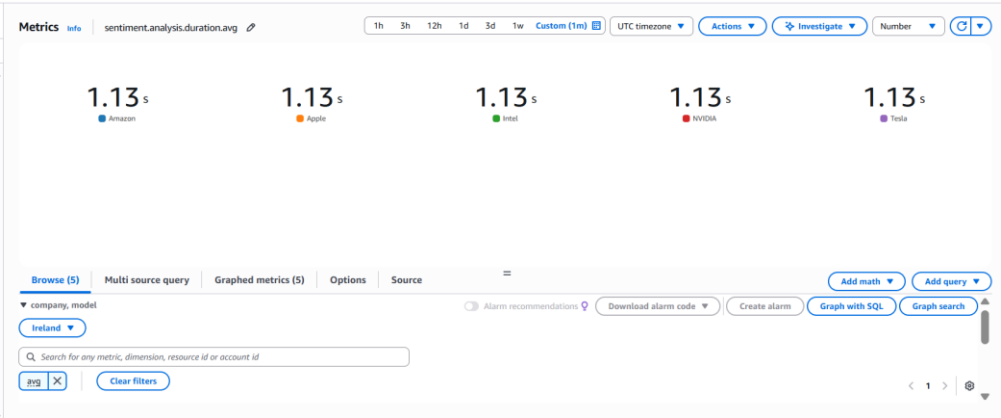
Forventet resultat i Docker Hub etter første vellykket workflow (med annerledes sha-***):



Oppgave 4 – Del 1

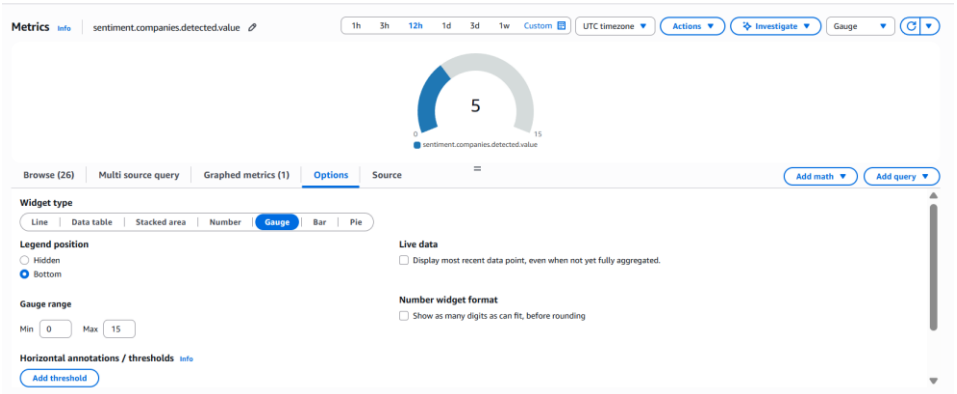
Timer

Her kan man se gjennomsnittlig tid av alle analyserte bedrifter i løpet av det siste minuttet. Dette kan være et viktig verktøy for de som utvikler applikasjonen, der man kan se det oppstår høy latency. Denne tar i bruk Timer metrikken som starter før og etter en analyse har blitt utført:



Gauge

Her kan man se maks oppdagede selskaper fra forrige søk. Man kan lage en alarm som varsler når gaugen er for høy, får man mange alarmer etter hverandre kan det kan være en tidlig varsellampe på unødvendig høy ressursbruk og et tegn på at man må fatte ressurssparende tiltak. For denne verdien brukte jeg en gauge for hver nye registrerte analyse. Gaugen kobles til “AtomicInteger companiesGauge” når programmet starter, og er bundet til den fram til det slutter.



Oppgave 4 – Del 2

Ved fork:

Bytt ut default variabler i infra-cloudwatch med egne

Endre navn i MetricsConfig ved "cloudwatch.namespace", "kandidat34", dette må være samme som i variables.tf

Build og kjør docker som i oppgave 3-A

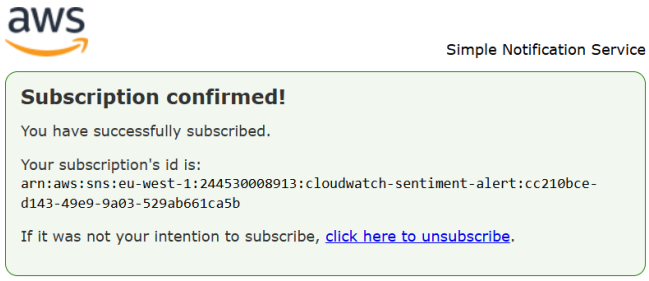
Terraform init

Terraform plan

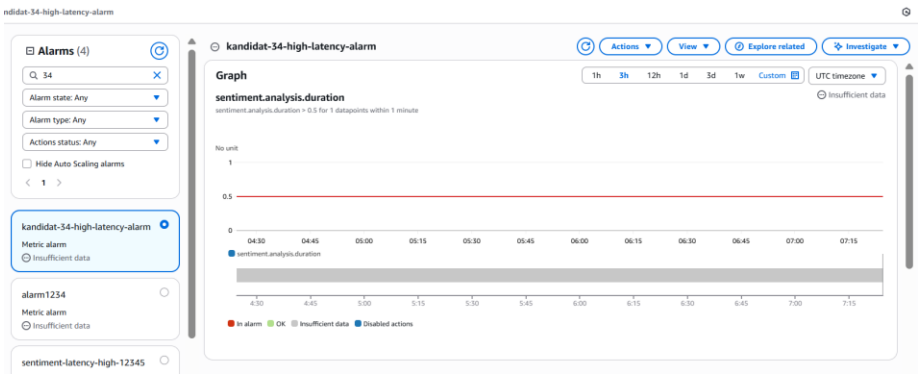
Terraform apply

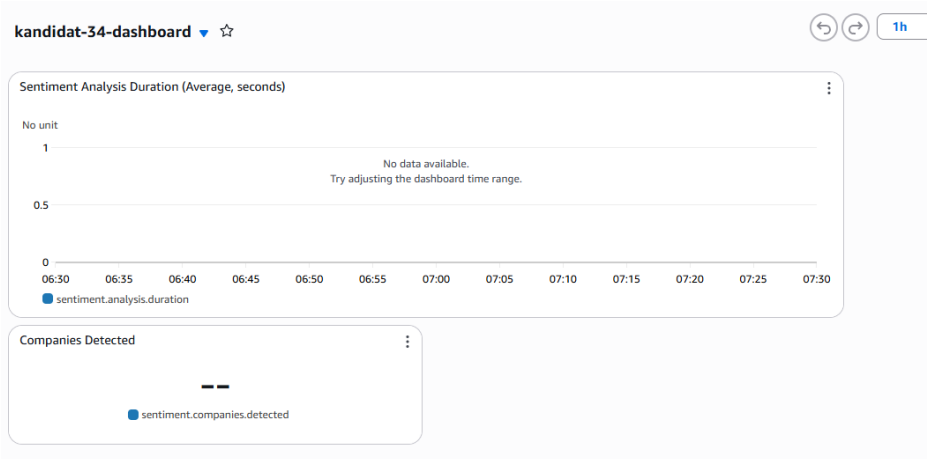
Etter dette gikk det galt. Terraform var vellykket, men:

Jeg fikk e-post om å følge alarmen, svarte på den:



Alarmen ser slik ut, satte latency limit til 0.5 sekunder for å prøve å provosere enn alarm, men det skjedde aldri.





Jeg kan se at dataen oppdaterer seg når jeg tar en POST, men den blir ikke registrert av widgets eller alarm. Widgets ble laget av terraform, men får ikke data inn.

Oppgave 5

Kunstig intelligens, eller KI, er som det sies i navnet kunstig. Språkmodeller som ChatGPT er trent på menneskelig data for å svare på spørsmål så likt som mulig som et menneske. En av KI sine mest menneskelige egenskaper er skråsikkerhet og en uvilje til å innrømme at den ikke vet svaret. Når dette implementeres i programvareutvikling ender det i beste fall opp med å ikke kompilere, men i verste fall ender det opp som implementerte funksjoner på et av verdens største operativsystem.

Det kan være veldig tidseffektivt å bruke KI som en assistent når man utvikler. Det minst kompliserte er å drive med konsentrerte applikasjoner der den har all informasjonen om applikasjonen i et dokument, gjerne skrevet på et eldre programmeringsspråk der det er mye data å hente fra forum som Stack Overflow fra utviklere med lignende problemer. Da er det lett å be den legge til funksjoner her og der, og å endre hele programmet.

Dette kan derimot bli problematisk ved større applikasjoner. Av egen erfaring og observasjoner sliter ChatGPT med feilsøking av større systemer fordi den ikke har nok kontekst og lange samtaler. Etter lengre samtaler kan den begynne å komme med løsninger som ikke fungerer og presenterer dem med sitat som: “AHA! Da vet jeg 100 % sikkert hva som er problemet” og “LØSNING (den riktige):”.

Det KI kan være god på er å lage omfattende fremgangsmåter for hvordan du kan starte et prosjekt eller løse et problem som å deploye et prosjekt. Den kan komme med en grunnmur du kan bygge videre på, eller en steg-for-steg-liste for hele prosjektet.

Der KI ligger i dag vil det være uansvarlig å overlate utvikling til den alene. Et menneske som kan styre og kontrollere bør være tilstede, men etter hvert som den utvikler seg kan det bli mindre skriving og mer “prompting” fra menneskets side. En erfaren utvikler kan se over KI-generert kode og evaluere om den ser trygg ut. Er man uheldig derimot kan debuggingen av KI generert kode ta lenger tid enn hvis man selv skrev den fra bunnen av, og det ender opp med å koste utviklere tid istedenfor å spare dem.

Mange bruker i dag språkmodeller og Copilot til å lære seg selv kode. Så lenge KI blir trent på menneskeskapt kode er dette bærekraftig, men når mer og mer kode genereres kommer vi til et punkt der KI trener på kode skrevet av KI, noe som kan stagnere utviklingen dens og da alle som avhenger av den.

KI kan være et veldig bra verktøy hvis man vet hvordan man bruker det og hva man kan forvente av det. Man kan vanligvis stole på språkmodeller til å hente opp lett tilgjengelig informasjon ved å søke på nettet, men den kan slite med logiske oppgaver som krever metodiske og sammenhengende steg for å løse dem, som matematikk.

Er KI bra for utvikling? Det KAN være bra, dersom man bruker det ansvarlig, ser over og tester, og er til enhver tid kritisk. For nye utviklere kan det være til en grad skadelig for utviklingen deres. Det kan hindre dem i å bygge opp en evne til å debugge og analysere kode, når KI kan gjøre det for dem. For erfarne utviklere kan det være hjelpsomt, helt til det ikke er det.