DevOps med gode intensjoner

Krav til leveransen

- Eksamensoppgaven er gitt på GitHub repository ; https://github.com/PGR301-2022/eksamen_2022
- Du skal ikke lage en fork av dette repositoryet, men kopiere innholdet til et nytt. Årsaken er at sensor vil lage en fork av ditt repo, og arbeidsflyten blir lettere hvis ditt

README.md

etter innleveringsfrist hvis du er bekymret for plagiat fra medstudenter.

Når sensor evaluerer oppgaven vil han/hun se på

- Ditt repository og "Actions" fanen i GutHub for å bekrefte at Workflows faktisk virker
- AWS miljøet i klassens AWS konto for å bekrefte at oppgaver som beskrevet er utført
- Vurdere drøftelsesoppgavene. Du må lage en "Readme" for besvarelsen i ditt repo.
- Sensor vil Lage en fork av ditt repo og tester ut pipelines med egen AWS bruker/github bruker.

Ved innlevering via WiseFlow, lager du et *tekstdokument* som kun inneholder link til dit repository

Litt om GitHub free tier

- I oppgaven blir du bedt om å lage GitHub actions workflows.
- Med GitHub "Free tier" har du 2,000 minutter med gratis byggetid per måned, dersom du bruker et privat repo.
- Dersom dere i en ekstrem situasjon skulle trenge mer byggetid, kan dere gjøre repository public. Da er byggetiden ubegrenset.
- Hvis dere da er bekymret for at andre skal kopiere arbeidet deres, kan dere lage en ny GitHub bruker med et tilfeldig navn.

OBS!

- I "Free" planen til GitHub er "branch protection" ikke tillat når et repository er privat. Det vil si at dere ikke kan konfigurere GitHub til å hindre push mot for eksempel *main* branch direkte, eller konfigurere regler for godkjenning før merge av pull request osv.
- I denne oppgaven blir dere bedt om å beskrive *hvordan* dette kan gjøres, men dere trenger altså ikke konfigurere dette for repoet dere leverer.

Evaluering

- Del 1 DevOps-prinsipper - 20 poeng
- Del 2 CI - 20 poeng
- Del 3 Docker - 20 poeng
- Del 4 Del - Metrics med Micrometer 20 poeng
- Del 5 Del - Terraform og CloudWatch Dashboards - 20 poeng

Utvikling i Cloud 9

Dere kan bruke et utviklingsmiljø i Cloud 9.
https://244530008913.signin.aws.amazon.com/console - logg på med studentnavn.
Cloud9 miljøene er ble laget på nytt i løpet av helgen før eksamen starter, passord er det "vanlige"

- Siden Cloud 9 miljøet blir laget på nytt før eksamen; må du installere Maven, sette opp "credential helper" osv. Se på en av øvingene vi har gjort i semesteret.

Hvis dere får følgende feilmelding når dere bygger koden med maven i Cloud9, må dere bare gjøre en "mvn clean"

```
java.lang.Error:
Unresolved compilation problem:
        The method builder() is undefined for the type Cart
        at
```

```
no.shoppifly.CartServiceTest.shouldRemoveCartAfterCheckout(CartServiceTest.jav
```

Bonusoppgave - 5 Poeng

Vi fant aldi ut av hvorfor ovnernevnte problem oppstår av og til med Maven i Cloud9. Hvis du klarer å reprodusere feilen konsekvent og kan komme med en forklaring på hvorfor dette skjer, og hva vi kan gjøre for å fikse det, gis 5 ekstra poeng.

Scenario

Som DevOps-ekspert, ferdig utlært fra Høgskolen Kristiania blir du ansatt i en bedrift, "Shopifly" som selger droner, men også andre varer nå som det nærmer seg jul.

Shopifly har store utfordringer med utviklingsprosessen sin

- De deployer kode første mandag i kvartalet.
- De pleide å deploye oftere før- men dette førte til ustabilitet. Selskapet ansatte flere testere, og startet en prosess der utviklingsledere måtte se over og godkjenne alle leveranser. De senket samtidig frekvensen på leveransene sine for å få bedre stabilitet.
- Når de deployer, feiler det fortsatt ofte. Da ruller de tilbake til forrige versjon, og ny funksjonalitet blir derfor ofte forsinket i månedsvis
- Leveransen skjer ved at Utviklingsteamet bruker FTP til å overføre en Spring boot JAR sammen med dokumentasjon i en ZIP. En egen avdeling tar i mot disse filene og installerer i AWS / Produksjon.

For å løse problemene sine, leide selskapet så inn DevOps-kompetanse fra Gaffel Consulting. Etter å ha sendt fire juniorkonsulenter som fakturerte for en liten formue ble det klart at de aldri kom til å klare å levere, og kontrakten ble sagt opp. "Jim" den "mest senior" av juniorkonsulentene har lagt inn noen kommentarer i koden som kan være til hjelp.

Det Gaffel Consulting klarte å levere på den medgåtte tiden ligger i dette repositoryet.

Nå er det din tur til å ta over!

Beskrivelse av API

Selskapet driver med elektronisk handel, og fokus for denne oppgaven er et API som implementerer en handlekurv. Gjør deg godt kjent med APIet og hvordan det virker - via Postman / Curl før du starter på oppgaven.

Du kan starte applikasjonen, enten i ditt Cloud9 miljø- eller på lokal maskin med kommandoen

```
mvn spring-boot:run
```

Request headers

OBS! For alle reqestene trenger å du sette HTTP header 'Content-Type: application/json'

Opprette handlekurv - POST /cart

Du kan lage ny handlekurv ved å gjøre en HTTP POST til `/cart` Uten "id"

Request body

```
{
  "items": [
    {
      "description": "Ugly christmas sweater",
      "qty": "1",
      "unitPrice": "500"
    }
  ]
}
```

Respons

id blir satt automatisk

```
{
  "id": "fb49e386-7124-4c16-9067-2dde2ee75d4e",
  "items": [
    {
      "description": "Ugly christmas sweater",
      "qty": 1,
      "unitPrice": 500.0
    }
  ]
}
```

Curl-eksempel

```
curl --location --request POST 'http://localhost:8080/cart' \
  --header 'Content-Type: application/json' \
  --data-raw '{
      "items":
```

```
        [
          {
            "description": "Ugly christmas sweater",
            "qty": "1",
            "unitPrice": "500"
          }
        ]
    }'
```

Oppdatere handlekurv - POST /cart

Du kan poste et helt cart-objekt med en "id" for å oppdatere handlekurven

Request

```
{
    "id": "fb49e386-7124-4c16-9067-2dde2ee75d4e",
    "items": [
        {
            "description": "Ugly christmas sweater",
            "qty": 1,
            "unitPrice": 500.0
        },
        {
            "description": "Shark socks",
            "qty": 20,
            "unitPrice": 10.0
        }
    ]
}
```

Response

Samme som request

Eksempel Curl kommando

```
curl --location --request POST 'http://localhost:8080/cart' \
--header 'Content-Type: application/json' \
--data-raw '{
    "id": "fb49e386-7124-4c16-9067-2dde2ee75d4e",
    "items": [
        {
            "description": "Ugly christmas sweater",
            "qty": 1,
            "unitPrice": 500.0
        },
        {
            "description": "Shark socks",
```

```
            "qty": 20,
            "unitPrice": 10.0
        }
    ]
}'
```

Fullføre handel - POST /cart/checkout

Sjekker ut handlekurven, sletter den fra listen over aktive handlekurver og returnerer en ordre ID

request

```
{
    "id": "fb49e386-7124-4c16-9067-2dde2ee75d4e",
    "items": [
        {
            "description": "Cheap 4K Drone with spare parts (needed)",
            "qty": 1,
            "unitPrice": 500.0
        },
        {
            "description": "Shark socks",
            "qty": 20,
            "unitPrice": 10.0
        }
    ]
}
```

Response

```
25d07757-4e56-408c-be30-a0568d35a70d
```

- Eksempel Curl kommando*

```
curl --location --request POST 'http://localhost:8080/cart/checkout' \
--header 'Content-Type: application/json' \
--data-raw '{
    "id": "fb49e386-7124-4c16-9067-2dde2ee75d4e",
    "items": [
        {
            "description": "Ugly christmas sweater with Drone logo",
            "qty": 1,
            "unitPrice": 500.0
        },
        {
            "description": "Shark socks",
```

```
            "qty": 20,
            "unitPrice": 10.0
        }
    ]
}'
```

Hente alle handlekurver - GET /carts

Du kan få en oversikt over alle aktive handlekurver med dette endepunktet.

Response

```
[
"4eb4d739-5df9-48b1-84c0-57c039d4fe35",
"cc7068e8-b855-416f-a34c-65dcdf478174",
"9e1e846f-45b7-472d-8bde-af9eba3224a5"
]
```

Eksempel Curl kommando

```
curl --location --request GET 'http://localhost:8080/carts' \
--header 'Content-Type: application/json'
```

Del 1 DevOps-prinsipper

Beskriv med egne ord;

- Hva er utfordringene med dagens systemutviklingsprosess - og hvordan vil innføring av DevOps kunne være med på å løse disse? Hvilke DevOps prinsipper blir brutt?
- En vanlig respons på mange feil under release av ny funksjonalitet er å gjøre det mindre hyppig, og samtidig forsøke å legge på mer kontroll og QA. Hva er problemet med dette ut ifra et DevOps perspektiv, og hva kan være en bedre tilnærming?
- Teamet overleverer kode til en annen avdelng som har ansvar for drift - hva er utfordringen med dette ut ifra et DevOps perspektiv, og hvilke gevinster kan man få ved at team han ansvar for både drift- og utvikling?
- Å release kode ofte kan også by på utfordringer. Beskriv hvilke- og hvordan vi kan bruke DevOps prinsipper til å redusere eller fjerne risiko ved hyppige leveraner.

Del 2 - CI

Konsulentene som har jobbet med innføring av DevOps har startet på en GitHub actions workflow for kontinuerlig integrasjon. GitHub actions workflow (yml) filen ligger i dette repositoryet og heter `ci.yml`

Problemet er at utviklingsteamet må starte jobben manuelt fra GitHub actions brukergrensesnittet. Det er jo ikke bra!

Du kan gjerne teste dette selv ved å gå til "Actions" i ditt repository, du vil se teksten "This workflow has a workflow_dispatch event trigger." Og vil ha et valg som heter "Run workflow"

Oppgave 1

- Start med å få workflowen til å kjøre når det lages en pull request, og på hver push til main branch

Oppgave 2

Det er andre utfordringer med denne flyten også; Workflowen kjører "ok" selv om det åpenbart er unit-testfeil i koden.

- Få først `ci.yml` workflow til å feile fordi enhetstesten feiler.
- Rett deretter enhetstesten og se at pipeline kjører "ok".
- Workflowen skal kompilere javakoden og kjøre enhetstester på hver eneste push, *uavhengig av branch*

Oppgave 3

Branch protection og status sjekker - Beskriv hva sensor må gjøre for å konfigurere sin fork på en slik måte at

- Ingen kan pushe kode direkte på main branch
- Kode kan merges til main branch ved å lage en Pull request med minst en godkjenning
- Kode kan merges til main bare når feature branchen som pull requesten er basert på, er verifisert av GitHub Actions.

Del 3 - Docker

Applikasjonen er laget for å pushe et container image til Docker Hub.

Det ligger en `Dockerfile` i prosjektet, og en workflow fil som heter `docker.yml` .

Oppgave 1

Beskriv hva du må gjøre for å få workflow til å fungere med din DockerHub konto? Hvorfor feiler workflowen?

Oppgave 2

Når du har fikset problemet i oppgave 1, og du forøker å kjøre applikasjonen fra Docker hub med for eksempel; `docker run <dockerhub brukeravn>/shopifly`

Får du en feilmelding

```
Exception in thread "main" java.lang.UnsupportedClassVersionError:
no/shoppifly/CddemoApplication has been compiled by a more recent version
of the Java Runtime (class file version 55.0), this version of the Java
Runtime only recognizes class file versions up to 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
        at
java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:473)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
```

De annsatte i Gaffel consulting tenkte at Maven-bygget kunne kjøres i GitHub Actions, med `mvn ...` , og at JAR filen kunne kopieres inn i Container Image docker under bygging.

Men så bestemte en av utviklerene seg for å oppgradere Javaversjonen i pom.xml, og workflow filen, til Java 11. Alt stoppet å fungere! Som dere ser av Dockerfilen, kjører Spring boot på Java 8...

```
FROM adoptopenjdk/openjdk8
```

Konsulentene ga opp, og hoppet som vanlig over på en annen oppgave. Så for øyeblikket har ikke Shopifly en fungerende applikasjon.

Vi kan få bedre kontroll på byggeprosessen ved også å gjøre maven bygget i en container. For å unngå lignende problemer i fremtiden ønsker vi derfor å bruke Docker til kompilere- og bygge koden.

- Skriv om Dockerfilen. til bruke en *Multi Stage Build*.
- Du må også rydde opp i `docker.yml` workflow filen... Fjern ønødvendige "steps".

Oppave 3

Gaffel consulting var ikke klar over at det var en dårlig idè å ha sitt container image i et offentlig Docker hub repository - og Shopifly har allerede sett at flere hundre har lastet ned deres container image. Et privat ECR repository i AWS er en bedre løsning.

- Lag dit eget ECR repository med kandidatnummer som navn, enten ved hjelp av UI - eller ved hjelp av CLI.

- Endre `docker.yml`, workflow til å pushe docker container til Amazon ECR, istedet for docker hub
- Beskriv deretter med egne ord hva sensor må gjøre for å få sin fork til å laste opp container image til sitt eget ECR repo.
- Docker workflow skal pushe et container image med en tag som er lik GitHub commit hash (id); for eksempel `244530008913.dkr.ecr.eu-west-1.amazonaws.com/glenn_exam_practice:8234efc`

Del 4 - Metrics, overvåkning og alarmer

Cloud9 er ikke verdens beste IDE. Det anbefales å gjøre den følgende oppgaven på lokal PC. Husk å kjøre
`aws configure` ;-)

Oppgave 1

Gjør nødvendige endringer i `pom.xml` - og koden, slik at applikasjonen kan levere Metrics til CloudWatch ved hjelp av Spring Boot Micrometer. Konfigurer applikasjonen til å bruke ditt eget ClodWatch Metrics Namespace - ditt Kandidatnummer.

OBS! Når dere innfører Micrometer i koden deres, vil enhetstesten slutte å fungere. Dere får lov til å slette enhetstesten når dere starter å jobbe med denne oppgaven. I "virkeligheten" ville vi brukt et rammeverk som feks Mockito
til å "mocke" micrometer fra enhetstestene, men det er ikke ønskelig at dere skal bruke tid på dette under eksamen!

Oppgave 2

Endre Javakoden slik at den rapporterer følgende Metrics til CloudWatch

- "carts" - Antall handlekurver på et gitt tidspunkt i tid - verdien kan gå opp og ned ettersom kunder sjekker ut handlekurver og nye blir laget.
- "cartsvalue" - Total sum med penger i handlekurver på et gitt tidspunkt i tid - verdien kan gå opp og ned ettersom kunder sjekker ut handlekurver og nye blir laget.
- "checkouts" - Totalt antall handlevogner er blitt sjekket ut
- "checkout_latency" - Gjennomsnittlig responstid for Checkout metoden i Controller-klassen.

Del 5 - Terraform og CloudWatch Dashboards

Konsulentene i Gaffel consulting hadde ambisiøse planer om å få Terraform-koden i dette repoet til å kjøre i GitHub Actions. Workflowen kjørte bra første gang, men nå feiler den hver gang, og klager over at en bucket med samme navn allerede eksisterer. Shopifly har tenkt på bruke denne bucketen til data-analyse.

```
Error: creating Amazon S3 (Simple Storage) Bucket (analytics-jim):
BucketAlreadyOwnedByYou:
Your previous request to create the named bucket succeeded and you already
own it.
```

De kommenterte derfor bare ut S3 bucket koden, og gikk videre til neste oppgave.

Oppgave 1

Se på `provider.tf filen` .

- Forklar med egne ord. Hva er årsaken til dette problemet? Hvorfor forsøker Terraform å opprette en bucket, når den allerede eksisterer?
- Gjør nødvendige Endre slik denne slik at Terraform kan kjøres flere ganger uten å forsøke å opprette ressurser hver gang den kjører.
- Fjern kommentarene fra `databacket.tf` slik at Terraform-koden også lager en S3 bucket.

Oppgave 2

Et annet problem er at "terraform apply" bare blir kjørt hver gang noen lager en Pull request. Vi ønsker bare å kjøre apply når noen gjør en push mot main branch.

Fullfør workflow filen `cloudwatch_dashboard.yml` filen slik at

- Apply kjører mot push mot main branch
- Terraform plan når det lages en Pull request

Oppgave 3

- Fullfør cloudwatch_dashboard.tf slik at koden lager et CloudWatch Dashboard med *fire widgets*. Disse skal vise metrikkene fra oppgave 2, Del 4.
- Antall handlekurver på et gitt tidspunkt i tid - verdien kan gå opp og ned ettersom kunder sjekker ut handlekurver og nye blir laget.
- Total sum med penger i handlekurver på et gitt tidspunkt i tid - verdien kan gå opp og ned ettersom kunder sjekker ut handlekurver og nye blir laget.
- Totalt antall handlevogner er blitt "sjekket ut" per time
- Gjennomsnittlig responstid for Checkout metoden i Controller-klassen.

Alarmer

Lag Terraform-kode som oppretter

- En CloudWatch Alarm som løses ut dersom antall handlekurver over tre repeternde perioder,på fem minutter, overstiger verdien 5
- Alarmen skal sendes som e-post til en addresse som gis i workflow filen `cloudwatch_dashboard.yml`

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Java 74.3% **HCL** 23.1% **Dockerfile** 2.6%