Oppgave 1

A) Levering av lenke til test i postman
https://vb9e9pvt7i.execute-api.eu-west-1.amazonaws.com/Prod/imageGen/
testet med {"prompt" : "prompttekst"} body i postman
B) Levering av lenke til vellykket workflow
[https://github.com/LouiseHjuler/DevOpsEksamen24/actions/runs/11921441935]

Oppgave 2

- workflow til main med apply
https://github.com/LouiseHjuler/DevOpsEksamen24/actions/runs/11998849255
- workflow til alt som ikke er main med plan
https://github.com/LouiseHjuler/DevOpsEksamen24/actions/runs/11998884373
- sqs queue
https://sqs.eu-west-1.amazonaws.com/244530008913/kandidat24_sqs_queue

Oppgave 3

- Tagge strategi

Tagger med github version count, da dette gir oversigt over hvilken iterasjon av koden som er benyttet, basert på antall commits via github.

Denne kan utvides med semantic versioning om det er behov, men for hvor simpel koden er føler jeg dette blir overkill.

Mulig å ligge til senere om nødvendig.

Samtidig ligger jeg opp en versjon med latest taggen, sådan at seneste image er lett å finne for de som skal benytte imaget.

- Docker container image:

Name: louisehjuler/23imagegen

https://hub.docker.com/repository/docker/louisehjuler/23imagegen/general

SQS = https://sqs.eu-west-1.amazonaws.com/244530008913/kandidat24_sqs_queue

Oppgave 4

Mail modtatt med subscription godkjenning. Klarte ikke å trigger alarmen selv.
Lenke til alarm her:

https://eu-west-1.console.aws.amazon.com/cloudwatch/home?region=eu-west-1#alarmsV2:alarm/kandidat_24AproxAgeOfOldestMsg

Oppgave 5

Automatisering og kontinuerlig levering (CI/CD)

I et gitt miljø med automatisering for utrulling av tjenester i en CI/CD pipeline, vil overordnet medføre mye bra. Å implementere en slik løsning fra bunn, vil generelt sett kreve mere planlegging, og testing, for å opprettholde kode-kvalitet og minimere mulige feil.

Automatisert levering vil i prinsippet føre til et sømløst og smertefritt utrulling av enhver oppdatering og forbedring av koden. Dette fører til høy hastighet for del-leveranser og milepæler målt mot en gitt kundes behov, og gjør det samtidig mulig å få feedback på spesifikke endringer, sammenlignet med gårsdagens større utrullinger av månedsvis av kode. Dette gir mulighet for hurtig å respondere til disse, sådan at f.eks. bruks-trender i løsningen, sikkerhetshull eller nye behov fort blir dekket inn.
CI/CD gir også større mulighet for å tilpasse retningen for utviklingen med mye høyere hastighet, sådan at man bedre kan utnytte arbeidskraften sin på best mulig måte.
Det øker også eierskap hos hver utvikler, da ikke godkjent kode ikke skal passere tests og bli rullet ut. Altså høyner dette insentivet for å skrive bra kode, og vil tvinge en gitt utvikler til å rette koden sin for å få det ut. Samtidig, skulle noe kode som ikke virker slippe igjennom testing, er det mindre barriere for å rulle koden tilbake, da hver endring er minimal.

Dog gjør dette også behovet for testing av koden innen den rulles ut ekstra viktig. Disse må være bygget inn i den automatiserte prosessen, og gjør derfor arbeidet med individuelle kodebiter mere tidskrevende, da tests ofte er noe som ellers bliver nedprioritert, men her får en kritisk posisjon.
I samme vene, krever automatisering at man overvåker løsningen sin nøye, og at rapporter fra alarmer tas seriøst og rettes opp i så fort som mulig, dette også spesielt fordi små endringer i f.eks. én microservice kan påvirke andre områder og services uten at disse nødvendigvis er åpenlyse når endringen blir lagd.
I tillegg er det ikke alle brukere som liker at løsningene dem bruker bliver endret ofte, dog føler jeg at dette sannsynligvis er mest tydelig når det gjeller UI eller UX endringer, da brukere da blir presentert med en læringskurve for bruk av løsningen.

Overvåkning

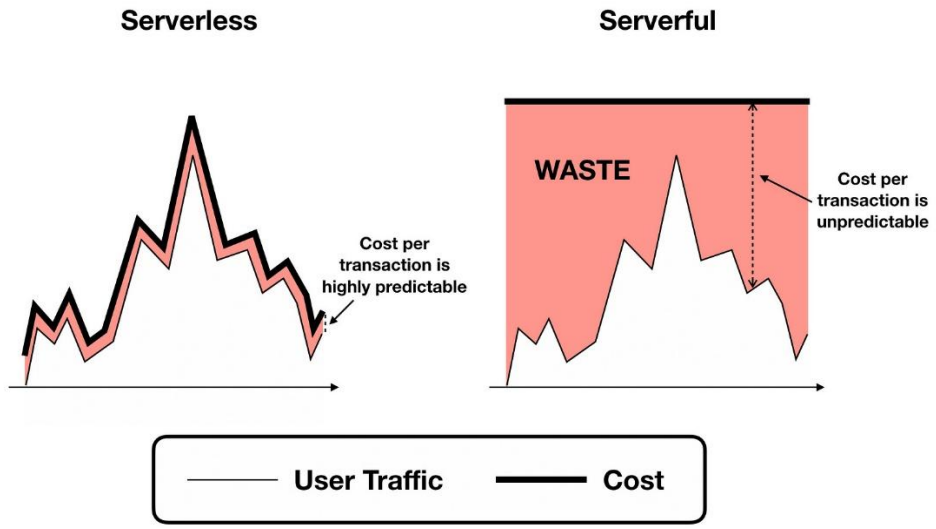
Som nevnt over er overvåking en hjørnesten når det snakkes om kontinuerlig levering og automatisering av serverløse løsninger. For fort å kunne finne feil kreves overvåkning med alarmer for spesifiserte nivå av feil og korrekt logging av disse, så det er mulig å oppdage og finne disse feilene så fort som mulig. Rettelsen av feil, og utrulling av patches og fixes, må skje i samme hast, om ikke fortere, som utrulling av vanlig kode i pipelinen.

Dette gjeller også for arkitektur hvor funksjoner er implementert som separate services, f.eks. lambda funksjoner og kø-systemer som SQS.
I cases med lambda funksjoner, vil disse bare kjøre når dem bliver kallet. Derfor kan det oppstå et problem, hvor oppståtte feil i disse ikke har blitt oppdaget, om den ikke har blitt kjørt på en stund. Samtidig er det ikke plausibelt å teste enhver lambda funksjon hver gang noe bliver rullet ut, noe som igjen understreker behovet for å ha bra oppsett med overvåkning, kombinert med en sjapp reaksjonstid på alarmer som utløses.

Skalerbarhet og kostnadskontroll

I løsninger som benytter serverløs arkitektur, kan nyte godt av «fri» skalerbarhet (så lenge det er penge på konto) og et miljø uten vedlikehold som ikke vedgår kodebasen. Dette fordi dem benytter eksterne leverandører til å ivareta eksterne servere som koden bliver både hostet på og kjørt fra. Derfor er samtidig noe som må vurderes fra et sikkerhetsstandpunkt, hvor f.eks. plasseringen av de fysiske servere skal tas opp, og i det heletatt, om man stoler på sin utbyder nok til at man mener at de ivaretar en høy nok digital og fysisk standard for sikkerhet rundt servicen dem tilbyr.

Serverløs- og mikrotjenestebasert arkitektur kommer og med hver sine kostnadsprognoser. Mikrotjenestebasert arkitektur står vanligvis og kjører uavbrutt og gir en forutsigbar, høy kostnad, hvorimot serverløs arkitektur vil resultere i en «pay-as-you-go» løsning, med uforutsigbare kostnader, som pr bruk er lave.
Det vil sige at om produktet bliver brukt nok vil det komme et punkt, hvor serverløs arkitektur og tilhørende kostnader overstiger kostnadene til en microtjenestebasert løsning. Dog er dette typisk et veldig høyt nivå av bruk som skal oppnås, og derfor finns det oftest gode argumenter for å velge serverløs arkitektur. Min pointe er godt illustrert av grafene under:



https://lumigo.io/blog/save-money-on-serverless-common-costly-mistakes-and-how-to-avoid-them/

Eierskap og ansvar

Ved kontinuerlig utrulling av kode til et serverløs miljø bliver mye av ansvaret for kodekvalitet flyttet tilbake på hver enkelt utvikler, da hver enkelt må sikre seg at koden dem ruller ut til produksjon kan gå igjennom tests. Hvis ikke koden gjør dette, er det nemlig dem selv som vil bli bedt om å rette opp på feil og mangler, og derfor blir insentivet for å skrive kode som er fungerende og optimalisert mye større for første ferd.
Dette gjør samtidig, at hver enkelt utvikler har god grunn til å dyktiggjøre seg, og får rik mulighet for dette, da dem naturlig kommer til å lære av egne og andres feil.
Da vil man også se en trend hvor feil som bliver oppdaget er mindre og mindre kritiske i takt med, at kodekvalitet og DevOps pipelinen bliver bedre og bedre.