

SkySikkerhet Konte Eksamen

Oppgave A

For å sikre en rask, skalerbar og sikker migrasjon til skyen, samtidig som vi tar hensyn til NovaTechs økonomiske og tekniske begrensninger, må løsningen oppfylle følgende krav:

- **Rask migrasjon** – Flytte kritiske legacy-applikasjoner til skyen før den kommende varmebølgen ødelegger on-premise-infrastrukturen.
- **Minimal kostnad** – Siden NovaTech har begrenset kapital, må løsningen være kostnadseffektiv og unngå unødvendige ressurser.
- **Skalerbarhet og global ekspansjon** – Infrastruktur skal kunne utvides globalt, men kun etter at betalende kunder er sikret.
- **Sikkerhet for sårbare legacy-applikasjoner** – Ettersom applikasjonene er usikre og ikke kan patches, må vi implementere en skybasert sikkerhetsstrategi.
- **Infrastructure as Code (IaC)** – Bruke Terraform for automatisering.
- **Backup** – Implementere en robust backup-strategi for å unngå fullstendig datatap ved feil

Github Actions Pipeline

GitHub Actions vil bli satt opp slik at hver gang kode pushes til repositoriet, vil en arbeidsflyt kjøres for å validere og teste Terraform-konfigurasjonene. Den første testen vil bruke tfsec for å skanne koden for sikkerhetsproblemer. Hvis tfsec finner feil, vil arbeidsflyten stoppe umiddelbart for å forhindre at usikre endringer blir implementert. I tillegg vil terraform validate bli kjørt for å sikre at konfigurasjonsfilene ikke har syntaks feil. Terraform fmt vil også bli brukt for å formatere koden etter beste praksis. Hvis alle disse sjekkene består, kan terraform plan og terraform apply bli utført, med en manuell godkjenningsmekanisme før endringene implementeres.

Hvilken Migrasjonsstrategi skal vi bruke?

Kilde brukt ved valg av migrasjons strategi: <https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html>

1. Retire

Beskrivelse:

- Retire-strategien innebærer å dekommisjonere eller arkivere applikasjoner som ikke lenger har forretningsverdi, eller som ikke er verdt å flytte til skyen.

Vanlige brukstilfeller:

- Applikasjonen har ikke lenger forretningsverdi.
- Ønsker å eliminere kostnader forbundet med vedlikehold og hosting.
- Redusere sikkerhetsrisikoen ved å fjerne systemer med utdatert operativsystem eller komponenter.
- Applikasjoner med svært lav ressursbruk (f.eks. "zombie"- eller "idle"-applikasjoner) kan identifiseres og deaktiveres.
- Ingen innkommende tilkoblinger til applikasjonen over en lengre periode (f.eks. 90 dager).

Hvorfor velge:

- Denne strategien reduserer unødvendige kostnader og forenkler IT-miljøet ved å fjerne overflødige applikasjoner.

2. Retain

Beskrivelse:

- Retain-strategien betyr å beholde applikasjoner i kilde-miljøet fordi de enten er underlagt spesielle sikkerhetskrav, har høyrisiko ved migrering, eller har kritiske avhengigheter som må løses før migrasjon.

Vanlige brukstilfeller:

- Oppfyllelse av sikkerhets- og compliance-krav (f.eks. datalagringslokasjon).
- Applikasjoner med høy migrasjonsrisiko som krever omfattende vurderinger.
- Avhengigheter til andre applikasjoner som må migreres først.
- Nylig oppgraderte systemer der neste tekniske oppfriskning er planlagt.
- Applikasjoner med få brukere eller lav ytelse, som kan være lite verdifulle å migrere.
- Applikasjoner med fysiske avhengigheter som ikke har et direkte skyekvivalent.
- Mainframe- eller midrange-applikasjoner som krever spesialvurderinger.

Hvorfor velge:

- Ved å beholde slike applikasjoner kan man sikre at kritiske krav og avhengigheter blir adressert før en full migrasjon gjennomføres.

3. Rehost (Lift and Shift)

Beskrivelse:

- Rehost, også kjent som lift and shift, innebærer å flytte applikasjonen fra kilde-miljøet til skyen uten å gjøre endringer i applikasjonens struktur. I vårt tilfelle migreres applikasjonen til GCPs Compute Engine.

Fordeler:

- **Rask implementering:** Minimal omstrukturering gir rask migrasjon.
- **Reduserte kostnader:** Mindre utviklingsarbeid reduserer oppstartskostnader.
- **Minimal risiko:** Bevarer den opprinnelige arkitekturen, noe som reduserer risikoen for feil under migrasjonen.
- **Sømløs overgang:** Applikasjonen fortsetter å betjene brukere mens migreringen pågår.

Hvorfor valgt:

- Gitt de økonomiske begrensningene og behovet for en rask løsning, er lift and shift med Compute Engine det mest hensiktsmessige alternativet. Etter migrasjonen kan man vurdere ytterligere optimaliseringer når budsjettet tillater det.

4. Relocate

Beskrivelse:

- Relocate-strategien innebærer å overføre et stort antall servere – gjerne som en samlet gruppe – fra on-premises til en skyplattform, eller fra ett skyområde (VPC, region, konto) til en annen.
-

Fordeler:

- **Minimalt omarbeid:** Krever ingen ny maskinvare eller større endringer i applikasjonen.
- **Kontinuerlig drift:** Applikasjonen forblir operativ under overføringen, noe som minimerer nedetid.
- **Fleksibilitet:** Brukbart når man ønsker å omplassere ressurser innenfor skyen for bedre administrasjon eller kostnadsoptimalisering.

Hvorfor velge:

- Denne strategien er ideell hvis man ønsker å omstrukturere infrastrukturen din uten å endre den underliggende applikasjonsarkitekturen.

5. Repurchase (Drop and Shop)

Beskrivelse:

- Repurchase-strategien innebærer å erstatte den eksisterende applikasjonen med en ny, skybasert løsning. Dette kan ofte innebære å bytte til en SaaS-løsning som gir bedre funksjonalitet og reduserte driftskostnader.

Fordeler:

- **Reduserte vedlikeholdskostnader:** Overgang fra tradisjonell lisensiering til en pay-as-you-go-modell.
- **Moderne funksjonalitet:** Tilgang til nyere teknologier, bedre integrasjon med sky-tjenester og økt tilgjengelighet.
- **Forenklet administrasjon:** Leverandøren tar seg av infrastrukturen, noe som reduserer den tekniske byrden.

Hvorfor velge:

- Denne tilnærmingen er gunstig hvis den eksisterende applikasjonen er kostbar å vedlikeholde og det finnes en moderne, mer effektiv skybasert løsning på markedet.

6. Replatform (Lift, Tinker, and Shift / Lift and Reshape)

Beskrivelse:

- Replatform-strategien innebærer å flytte applikasjonen til skyen med mindre endringer for å utnytte noen av skyens fordeler, for eksempel ved å bruke administrerte tjenester eller serverløse løsninger.

Fordeler:

- **Kostnadsbesparelser:** Ved å migrere til en administrert tjeneste kan man redusere driftskostnader.
- **Forbedret ytelse:** Optimalisering for skyens infrastruktur kan gi bedre ytelse.
- **Redusert administrasjonsbyrde:** Skyleverandøren håndterer mange administrative oppgaver.

Hvorfor velge:

- Hvis man ønsker å forbedre applikasjonens drift uten å gjennomføre en fullstendig refaktorering, er replatform en ideell løsning. Dette alternativet kombinerer elementer av lift and shift med noen optimaliseringer.

7. Refactor / Re-architect

Beskrivelse:

- Refactor- eller re-architect-strategien innebærer en fullstendig omstrukturering av applikasjonen for å dra full nytte av sky-native arkitektur. Dette kan inkludere oppdeling av en monolitt i mikrotjenester, bruk av serverløse funksjoner eller andre optimaliseringer.

Fordeler:

- **Skalerbarhet og ytelse:** En moderne, sky-native applikasjon kan utnytte skyens fulle potensial.
- **Økt fleksibilitet:** Forbedret agilitet og mulighet til raskere å implementere nye funksjoner.
- **Bedre vedlikehold:** En refaktorert applikasjon er ofte enklere å oppdatere og sikre.

Ulemper:

- **Ressurskrevende:** Krever betydelig tid, kostnader og omstrukturering.
- **Høy risiko:** Overgangen kan introdusere nye feil og utfordringer underveis.

Hvorfor velge:

- Denne strategien er best egnet for applikasjoner som har kritiske begrensninger i sin eksisterende arkitektur, eller som ikke lenger møter forretningskravene. Det er en langsiktig investering for å oppnå maksimal ytelse og fleksibilitet.

Hvorfor Velges Rehost (Lift and Shift med Compute Engine) i Vårt Tilfelle?

Gitt de økonomiske begrensningene og behovet for en rask migrasjon, er **Rehost** – altså å flytte applikasjonen “som den er” til Compute Engine det mest levedyktige alternativet akkurat nå. Denne strategien gir:

- **Rask implementering:** Minimal omstrukturering gir rask overføring.
- **Kostnadseffektivitet:** Reduserte utviklings- og tilpasningskostnader.
- **Lav risiko:** Bevarer den opprinnelige arkitekturen, noe som reduserer potensielle feil og driftstans.
- **Fleksibilitet:** Etter en stabil migrasjon kan applikasjonen senere optimaliseres, containeriseres eller refaktoreres dersom forretningsbehovene endres eller budsjettet øker.

Denne helhetlige tilnærmingen gir oss en solid, kostnadseffektiv overgang til GCP, samtidig som vi har en plan for fremtidige forbedringer basert på de øvrige migrasjonsstrategiene

Valg av Cloud løsning

Valg av Cloud-løsning: IaaS for Lift-and-Shift

Når man skal migrere en eksisterende infrastruktur til skyen med en **lift-and-shift**-tilnærming, er **Infrastructure as a Service (IaaS)** den mest hensiktsmessige cloud-modellen. IaaS gir full kontroll over virtuelle maskiner, nettverk og lagring, samtidig som man eliminerer behovet for fysisk maskinvare og datasentre.

Hvorfor velge IaaS for Lift-and-Shift?

Lift-and-shift innebærer å flytte eksisterende applikasjoner til skyen uten å endre dem vesentlig. Dette betyr at applikasjonene fortsatt vil kreve underliggende infrastruktur, som servere og nettverkskonfigurasjoner, noe som gjør IaaS til den beste modellen. Med IaaS kan man:

- **Minimere utviklingskostnader:** Fordi applikasjonene ikke trenger større omstrukturering, unngår man omfattende kodeendringer.
- **Opprettholde eksisterende driftsmodeller:** IT-teamet kan fortsette å administrere servere, nettverk og lagring på en måte som ligner det de gjør on-premise.
- **Skalere ressursene ved behov:** IaaS gjør det enkelt å tilpasse kapasiteten dynamisk, noe som gir større fleksibilitet sammenlignet med tradisjonell fysisk infrastruktur.
- **Automatisere infrastrukturhåndtering:** Bruk av Infrastructure as Code (IaC) med Terraform muliggjør standardisert og reproducerbar infrastrukturadministrasjon.

Selv om IaaS gir stor kontroll, medfører det også ansvar for administrasjon av operativsystemer, sikkerhet og konfigurasjoner. Dette kan være en ulempe sammenlignet med mer administrerte cloud-løsninger.

Oversikt over Cloud-modeller

Cloud computing er vanligvis delt inn i tre hovedmodeller: **IaaS (Infrastructure as a Service)**, **PaaS (Platform as a Service)**, og **SaaS (Software as a Service)**. Valget av riktig modell avhenger av hvor mye kontroll organisasjonen ønsker, og hvor mye av infrastrukturen de ønsker å administrere selv.

1. Infrastructure as a Service (IaaS)

IaaS gir tilgang til virtuelle maskiner, lagring og nettverksressurser uten at man trenger å administrere fysisk maskinvare. Brukere har full kontroll over operativsystemene og applikasjonene som kjører på infrastrukturen.

Fordeler med IaaS

- **Fleksibilitet og kontroll:** Brukere har full administrativ tilgang til servere og nettverk.
- **Skalerbarhet:** Enkle justeringer av kapasitet basert på behov.
- **Reduserte kapitalutgifter:** Ingen behov for fysiske servere eller datasentre.
- **Egnet for Lift-and-Shift:** Gir mulighet til å migrere eksisterende applikasjoner raskt uten omstrukturering.

Ulemper med IaaS

- **Krever administrasjon:** Operativsystemer, sikkerhet og programvare må fortsatt vedlikeholdes av brukeren.
- **Høyere kompleksitet:** Kan kreve spesialisert kompetanse for optimal konfigurasjon og administrasjon.
- **Kostnader ved ineffektiv ressursbruk:** Hvis ressursene ikke administreres godt, kan kostnadene øke raskt.

Når bruke IaaS?

- Når man ønsker **kontroll over servere og nettverk**.
- Når man skal **migrere eksisterende applikasjoner (lift-and-shift)**.
- Når man har behov for **høy grad av tilpasning og fleksibilitet**.

2. Platform as a Service (PaaS)

PaaS gir utviklere et administrert miljø for å bygge, distribuere og administrere applikasjoner uten å håndtere underliggende infrastruktur.

Fordeler med PaaS

- **Redusert operasjonelt ansvar:** Skyleverandøren administrerer operativsystemer, databaser og nettverk.
- **Raskere utvikling:** Utviklere kan fokusere på kode fremfor infrastruktur.
- **Automatisert skalering:** PaaS-plattformer skalerer ressurser dynamisk basert på bruk.

Ulemper med PaaS

- **Begrenset kontroll:** Mindre fleksibilitet i konfigurasjon og infrastrukturvalg.
- **Vendor lock-in:** Applikasjoner kan bli tett knyttet til en spesifikk leverandør.
- **Kan være dyrt for visse brukstilfeller:** Kostnader kan øke hvis man ikke optimaliserer bruken.

Når bruke PaaS?

- Når utviklingsteamet ønsker å **fokusere på kode, ikke infrastruktur**.
- Når man utvikler **nye cloud-native applikasjoner**.
- Når man trenger **automatisert skalering og integrerte utviklingsverktøy**.

PaaS ville vært mer aktuelt hvis vi ønsket å modernisere applikasjonene ved å containerisere dem eller bruke administrerte tjenester som Google App Engine eller Cloud Run.

3. Software as a Service (SaaS)

SaaS leverer ferdige programvareløsninger som kan brukes direkte av sluttbrukere uten behov for installasjon eller administrasjon.

Fordeler med SaaS

- **Enkel implementering:** Krever ingen lokal installasjon eller vedlikehold.
- **Automatiske oppdateringer:** Leverandøren sørger for vedlikehold, sikkerhet og oppdateringer.
- **Kostnadseffektivt for mange bedrifter:** Betaling skjer ofte per bruker eller forbruk.

Ulemper med SaaS

- **Begrenset tilpasning:** Brukere må tilpasse seg de funksjonene som tilbys.
- **Vendor lock-in:** Overgang til en annen plattform kan være utfordrende.
- **Sikkerhets- og personvern hensyn:** Data lagres eksternt, noe som kan være en utfordring for sensitive systemer.

Når bruke SaaS?

- Når man **trenger en ferdig løsning uten vedlikeholdsansvar**.
- Når applikasjoner som **CRM, e-post og dokumentbehandling** er hovedfokus.
- Når **brukervennlighet og rask tilgang er viktigere enn fleksibilitet**.

SaaS ville vært aktuelt hvis vi ønsket å erstatte en eksisterende applikasjon med en skytjeneste som allerede tilbyr tilsvarende funksjonalitet, for eksempel å bytte ut en lokal database med en SaaS-basert CRM-plattform.

Oppsummering: Hvorfor IaaS for Lift-and-Shift?

I vår situasjon, hvor vi ønsker en lift-and-shift-migrasjon, er IaaS det beste valget fordi vi får fleksibiliteten til å beholde eksisterende applikasjoner med minimal omstrukturering. PaaS og SaaS kunne vært aktuelle dersom vi ønsket en mer omfattende modernisering eller erstatning av eksisterende løsninger. IaaS gir oss kontroll over infrastrukturen, mulighet til å migrere raskt, og fleksibilitet til å optimalisere løsningen videre etter migrasjonen.

Kilder som er brukt: <https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=nb>

Plan for terraform kode

Først vil infrastrukturen etableres med et globalt VPC-nettverk, der fire subnetter opprettes, ett for hvert kontinent: Asia, Nord-Amerika, Europa og Australia. Disse subnettene vil være plassert i strategiske regioner innenfor hvert kontinent for å optimalisere ytelse og tilkobling. Hvert subnet vil få egne nettverksressurser og nødvendige IAM-roller for å styre tilgangskontroll.

For hver region vil det opprettes en VM-mal (Instance Template), som vil definere konfigurasjonen for de virtuelle maskinene, inkludert operativsystem, maskinvareprofil, oppstartsskript og nettverkstilgang. Disse malene vil brukes av Instance Managers som administrerer opprettelse, vedlikehold og skalering av instanser i hver region.

Instance Groups vil opprettes for hvert kontinent basert på de respektive VM-malene. Disse gruppene vil starte med en instans i hver region, men autoskalering vil aktiveres for Europa og Nord-Amerika basert på det faktum at jeg estimerer at disse regionene vil ha mest pågang. Dersom dette ikke er tilfellet, kan dette justeres etter behov. Videre vil Autoskaleringen basere seg på CPU-belastning og nettverkstrafikk, slik at kapasiteten justeres automatisk etter behov.

For å distribuere trafikken effektivt mellom instansene, vil det opprettes en global lastbalanser som opererer over HTTPS. Lastbalanseren vil dirigere forespørsler til de ulike instansgruppene basert på nærhet og belastning. Den vil være konfigurert med et SSL-sertifikat for å sikre HTTPS-trafikk, samt en helsekontroll for kontinuerlig overvåking av instansenes tilgjengelighet og ytelse.

Hver virtuelle maskin vil få automatiske sikkerhetskopier for å sikre dataintegritet og rask gjenoppretting ved feil. Backup-konfigurasjonen vil ta øyeblikksbilder (snapshots) av diskene til hver VM på en jevnlig basis.

Til slutt vil det settes opp sikkerhetsmekanismer for å beskytte infrastrukturen. En brannmur (firewall) vil definere regler for innkommende og utgående trafikk. En Web Application Firewall (WAF) vil konfigureres for å beskytte webapplikasjonen mot vanlige angrep som SQL-injeksjon og XSS (Cross-Site Scripting).

Illustrasjon av hvordan oppsettet blir:



Oppgave A. 1

OBS! Dersom jeg hadde hatt bedre tid ville jeg også implementert Cloud CDN.

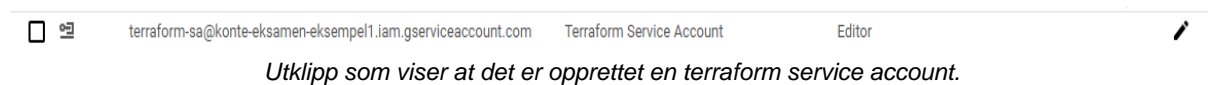
Hovedformålet med en CDN er å redusere latenstid (forsinkelse) og forbedre lastetider for nettsider, applikasjoner og annet digitalt innhold ved å lagre kopier av innholdet nærmere sluttbrukerne.

Implementering av Github Actions

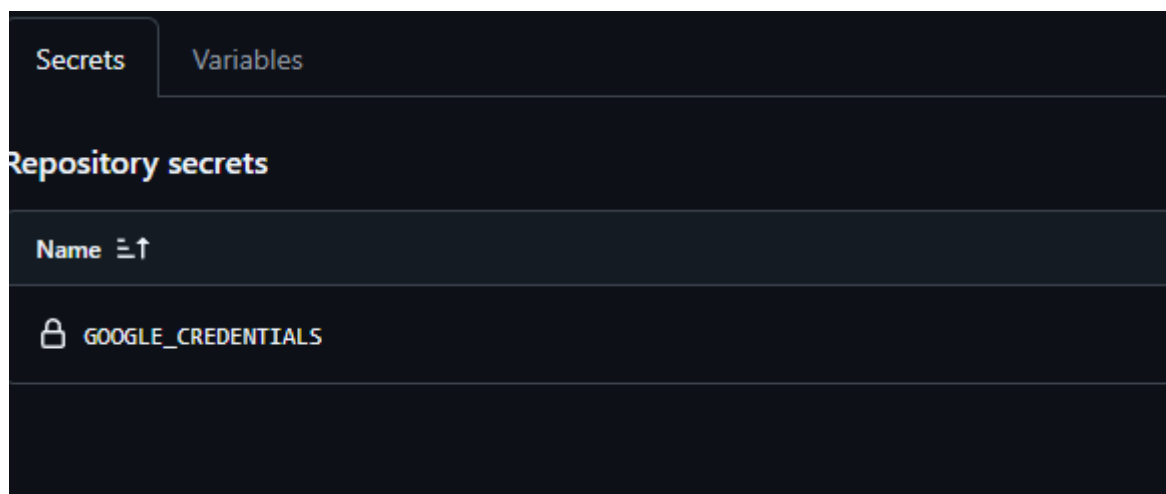
Som første steg implementerte jeg GitHub Actions for å automatisere arbeidsflyten og sikre at all kode som pushes, gjennomgår omfattende tester og sikkerhetssjekker.

Deretter opprettet jeg en servicekonto i Google Cloud Platform (GCP) via konsollen. Denne kontoen ble nøye konfigurert etter prinsippet om minste privilegium, slik at den kun har de nødvendige rettighetene for å opprette og slette ressurser. Ved å begrense tilgangen på denne måten reduseres risikoen for uautorisert bruk og utilsiktede endringer i miljøet.

Disse tiltakene bidrar sammen til en tryggere og mer robust deploy-prosess, der både koden og infrastrukturen blir håndtert med høy sikkerhetsstandard.



Deretter opprettet jeg en GitHub Secret for å lagre nøkkelen. Denne secreten integreres trygt i GitHub Actions, slik at autentiseringen mot GCP skjer uten at sensitive data eksponeres.



Deretter skal jeg opprette workflowen. Jeg har valgt å ikke følge den opprinnelige planen, ettersom jeg ønsker å integrere flere tiltak som øker sikkerheten og sikrer at koden er feilfri.

Jeg har valgt å dele opp koden i tre forskjellige workflows.

- Terraform-destroy.yml

- Terraform-plan.yml
- Terraform-apply.yml

Denne inndelingen gir bedre oversikt og kontroll over de ulike Terraform-operasjonene, og bidrar til en sikrere deploy-prosess.

Terraform-destroy.yml

Dersom det skulle bli nødvendig å fjerne alle Terraform-ressurser, kan dette gjøres, men prosessen må manuelt trigges av eieren av repositoriet – altså meg. Dette sikrer at destruktive handlinger kun utføres med godkjenning.

```
.github > workflows > 📄 destroy.yml
1  name: Terraform Destroy
2
3  on:
4    workflow_dispatch:
5
6  jobs:
7    terraform-destroy:
8      runs-on: ubuntu-latest
9
10     steps:
11       - name: Checkout code
12         uses: actions/checkout@v3
13
14       - name: Authenticate to GCP
15         uses: google-github-actions/auth@v1
16         with:
17           credentials_json: ${ secrets.GOOGLE_CREDENTIALS }
18
19       - name: Set up Terraform
20         uses: hashicorp/setup-terraform@v2
21         with:
22           terraform_version: 1.5.5
23
24       - name: Terraform Init
25         run: terraform init
26
27       - name: Terraform Destroy
28         run: terraform destroy -auto-approve
```

Utklipp av destroy.yml

Terraform-plan.yml

Terraform-Plan kjører automatisk når det gjøres et push til main-grenen. Workflowen definerer en jobb kalt terraform_plan som kjører på en Ubuntu-basert runner. Jobben inneholder flere steg som utfører ulike oppgaver relatert til Terraform og sikkerhetssjekker før Terraform-planen genereres.

Først sjekkes kildekoden ut med actions/checkout@v3, slik at workflowen får tilgang til prosjektets filer. Deretter kjøres Gitleaks, et verktøy som skanner kildekoden for hemmelige

nøkler og sensitive data, ved hjelp av [zricethezav/gitleaks-action@v1.5.0](https://github.com/zricethezav/gitleaks-action@v1.5.0). Dette hjelper med å forhindre at sensitive opplysninger blir sjekket inn i repositoryet.

Videre installeres tfsec, et sikkerhetsverktøy for Terraform, ved å laste ned den nyeste versjonen fra GitHub, gjøre den kjørbart og flytte den til /usr/local/bin/. Deretter kjøres tfsec . for å analysere Terraform-koden for sårbarheter, slik at eventuelle sikkerhetsrisikoer kan oppdages tidlig.

Terraform settes deretter opp ved å bruke hashicorp/setup-terraform@v2, som sikrer at den spesifikke Terraform-versjonen 1.5.5 blir installert. For at Terraform skal kunne jobbe med Google Cloud, autentiseres workflowen mot Google Cloud ved hjelp av google-github-actions/auth@v1, hvor nødvendige autentiseringsopplysninger hentes fra secrets.GOOGLE_CREDENTIALS.

Etter at autentiseringen er på plass, kjøres terraform fmt -check -recursive for å sjekke at Terraform-konfigurasjonen følger riktig formatering. Deretter kjøres terraform init, som initialiserer Terraform ved å laste ned nødvendige moduler og providers. Når initialiseringen er fullført, kjøres terraform validate for å verifisere at konfigurasjonen er gyldig og ikke inneholder syntaksfeil eller logiske feil.

Til slutt kjøres terraform plan -out=tfplan, som lager en plan for hvilke endringer Terraform vil gjøre dersom infrastrukturen blir anvendt. Denne planen gir en forhåndsvisning av endringene og lagres i en fil kalt tfplan, slik at den kan brukes senere i en eventuell "apply"-prosess. Workflowen sikrer dermed at Terraform-koden er formatert riktig, sikker, gyldig og klar for videre bruk i infrastrukturadministrasjon. Denne workflowen er hovedsaklig for å sjekke hva koden potensielt ville endret. Dette vil man da se når terraform plan kjøres.

```

.github > workflows > terraform-plan.yml
1  name: Terraform Plan
2
3  on:
4    push:
5      branches:
6        - main
7
8  jobs:
9    terraform_plan:
10     runs-on: ubuntu-latest
11
12     steps:
13       # Sjekk ut kildekoden
14       - name: Checkout code
15         uses: actions/checkout@v3
16
17       # Hemmelig-nøkkelskanning med Gitleaks
18       - name: Run Gitleaks secret scan
19         uses: zricethezav/gitleaks-action@v1.5.0
20
21       # Installer tfsec for sikkerhetskontroll av Terraform-koden
22       - name: Install tfsec
23         run: |
24           wget https://github.com/aquasecurity/tfsec/releases/latest/download/tfsec-linux-amd64 -O tfsec
25           chmod +x tfsec
26           sudo mv tfsec /usr/local/bin/
27
28       # Kjør tfsec for å sjekke for sårbarheter
29       - name: Run tfsec
30         run: tfsec .
31
32       # Sett opp Terraform
33       - name: Set up Terraform
34         uses: hashicorp/setup-terraform@v2
35         with:
36           terraform_version: 1.5.5
37
38       # * **Autentiser mot Google Cloud før Terraform Init**
39       - name: Authenticate with Google Cloud
40         uses: google-github-actions/auth@v1
41         with:
42           credentials_json: ${ secrets.GOOGLE_CREDENTIALS }
43
44       # Sjekk Terraform-format
45       - name: Terraform FMT Check
46         run: terraform fmt -check -recursive
47
48       # Initialiser Terraform før validering
49       - name: Terraform Init
50         run: terraform init
51
52       # Valider Terraform-konfigurasjonen
53       - name: Terraform Validate
54         run: terraform validate
55
56       # Lag en plan for endringer
57       - name: Terraform Plan
58         run: terraform plan -out=tfplan

```

Utklipp som viser terraform-plan.yml.

Terraform-apply.yml

Terraform-apply er konfigurert for å automatisere utrulling av Terraform-konfigurasjonen, men i motsetning til terraform-plan, må denne workflowen trigges manuelt via workflow_dispatch. Dette betyr at infrastrukturen ikke oppdateres automatisk ved hver commit eller merge til hovedgrenen, men krever en handling fra en utvikler eller administrator. Denne tilnærmingen øker sikkerheten og kontrollen, da infrastrukturendringer kun gjennomføres etter grundig vurdering og testing.

Workflowen starter med å sjekke ut den nyeste versjonen av kodebasen fra GitHub-repositoriet ved hjelp av actions/checkout@v3. Deretter utføres en sikkerhetsskanning med Gitleaks for å identifisere eventuelle eksponerte hemmelige nøkler i kodebasen, noe som bidrar til å redusere risikoen for utilsiktet lekkasje av sensitiv informasjon. Videre installeres og kjøres tfsec, et verktøy som analyserer Terraform-koden for potensielle

sikkerhetssårbarheter og brudd på beste praksis, slik at eventuelle svakheter kan avdekkes tidlig i prosessen.

Terraform settes deretter opp ved hjelp av hashicorp/setup-terraform@v2, som sikrer at riktig versjon av Terraform (1.5.5) er tilgjengelig i miljøet. Etter dette autentiserer workflowen seg mot Google Cloud ved hjelp av google-github-actions/auth@v1, hvor GOOGLE_CREDENTIALS-hemmeligheten fra GitHub Secrets brukes til å gi Terraform tilgang til nødvendige ressurser i Google Cloud-prosjektet. Dette er en kritisk del av prosessen, da den muliggjør infrastrukturendringer samtidig som sensitive autentiseringsdetaljer holdes sikre.

Før Terraform kjøres, gjennomføres en formateringssjekk med terraform fmt -check -recursive, som sikrer at koden følger Terraform sine formateringsstandards for å opprettholde en enhetlig og lesbar kodebase. Deretter initialiseres Terraform med terraform init, som laster ned nødvendige providere og moduler. Etter initialisering valideres konfigurasjonen med terraform validate, noe som kontrollerer at koden har riktig syntaks og følger Terraform sine regler.

Neste steg er å generere en Terraform Plan med terraform plan -out=tfplan, som skaper en detaljert plan for hvilke ressurser som vil bli opprettet, oppdatert eller slettet. Til slutt utføres terraform apply -auto-approve tfplan, som implementerer de planlagte endringene i Google Cloud-miljøet. Bruken av -auto-approve betyr at Terraform automatisk godkjenner endringene uten menneskelig inngripen, noe som er nyttig for automatisering, men bør håndteres med forsiktighet i produksjonsmiljøer.

Det faktum at denne workflowen ikke kjøres automatisk, men må trigges manuelt, gir en betydelig sikkerhetsfordel. Ved å kreve en eksplisitt handling fra en autorisert bruker, reduseres risikoen for utilsiktede endringer i infrastrukturen, enten det skyldes feil i kode, kompromitterte secrets eller uforutsette sikkerhetshendelser. Denne tilnærmingen gir også teamet bedre mulighet til å gjennomgå Terraform-planen før endringene tas i bruk, slik at eventuelle feil eller uønskede konsekvenser kan oppdages og korrigeres før de implementeres.

Dersom flere personer arbeider på prosjektet, ville jeg satt opp denne workflowen slik at den kunne utløses av andre teammedlemmer, men med en mekanisme for at den endelige godkjenningen må komme fra den personen som er ansvarlig for prosjektet eller repositoriet. Dette sikrer at selv om flere kan initiere prosessen, forblir kontrollen med kritiske infrastrukturendringer hos den ansvarlige, noe som bidrar til økt sikkerhet og robusthet i driftsprosessen.

For å implementere en løsning der andre kan trigge Terraform Apply-workflowen, men den faktiske deployen må godkjennes av prosjektansvarlig, kan du benytte GitHubs miljøfunksjonalitet (environments). Med denne løsningen kan du opprette et miljø, for eksempel «production», som krever manuell godkjenning før workflowen fortsetter. Dette oppsettet sikrer at selv om flere teammedlemmer har mulighet til å utløse workflowen via *workflow_dispatch*, vil ikke endringene settes ut i live-miljøet uten at den ansvarlige eksplisitt godkjenner deployen.

For å gjøre dette må du først opprette et miljø i GitHub-repositoriets innstillinger, og konfigurere en beskyttelsesregel (environment protection rule) som krever at en bestemt

person eller et sett med godkjennerne gir sin tillatelse før deployen fortsetter. Deretter legger du til et environment-felt i workflow-filen din under jobben.

Hadde repoet vært offentlig, ville jeg konfigurert workflowen slik at den utnyttet GitHubs miljøfunksjonalitet for å kreve manuell godkjenning før deploy. Dette gir en ekstra sikkerhetsmekanisme der andre teammedlemmer kan trigge prosessen, men den endelige deployen må godkjennes av den ansvarlige. Siden det imidlertid ikke er mulig å sette opp miljøer for private repositories uten å oppgradere til en annen plan, valgte jeg å implementere workflowen slik jeg har gjort det nå. Denne løsningen opprettholder en robust og kontrollert prosess, selv om den ikke benytter den ekstra godkjenningsmekanismen som er tilgjengelig for offentlige repositorier.

```
.github > workflows > terraform-apply.yml
1  name: Terraform Apply (Manual Trigger)
2
3  on:
4    workflow_dispatch:
5
6  jobs:
7    terraform_apply:
8      runs-on: ubuntu-latest
9
10     steps:
11       # Sjekk ut kildekoden
12       - name: Checkout code
13         uses: actions/checkout@v3
14
15       # Hemmelig-nøkkelskanning med Gitleaks
16       - name: Run Gitleaks secret scan
17         uses: zricethezav/gitleaks-action@v1.5.0
18
19       # Installer tfsec for sikkerhetskontroll av Terraform-koden
20       - name: Install tfsec
21         run: |
22           wget https://github.com/aquasecurity/tfsec/releases/latest/download/tfsec-linux-amd64 -O tfsec
23           chmod +x tfsec
24           sudo mv tfsec /usr/local/bin/
25
26       # Kjør tfsec for å sjekke for sårbarheter
27       - name: Run tfsec
28         run: tfsec .
29
30       # Sett opp Terraform
31       - name: Set up Terraform
32         uses: hashicorp/setup-terraform@v2
33         with:
34           terraform_version: 1.5.5
35
36       # Autentiserer mot GCP
37       - name: Authenticate with Google Cloud
38         uses: google-github-actions/auth@v1
39         with:
40           credentials_json: ${ secrets.GOOGLE_CREDENTIALS }
41
42       # Sjekk Terraform-format
43       - name: Terraform FMT Check
44         run: terraform fmt -check -recursive
45
46       # Initialiser Terraform før validering
47       - name: Terraform Init
48         run: terraform init
49
50       # Valider Terraform-konfigurasjonen
51       - name: Terraform Validate
52         run: terraform validate
53
54       # Lag en plan for endringer
55       - name: Terraform Plan
56         run: terraform plan -out=tfplan
57
58       # Utfør Terraform Apply
59       - name: Terraform Apply
60         run: terraform apply -auto-approve tfplan
61
```

Utklipp som viser terraform-apply.yml.

Terraform kode / Infrastruktur oppsett

Her vil jeg dokumentere koden som ble brukt for å sette opp infrastruktur i produksjonsmiljøet. Dokumentasjonen vil inkludere de ressursene som er opprettet, hvordan disse er konfigurert, og hvilke alternativer som kan justeres etter videre testing eller endrede krav.

Det første som ble gjort var å opprette nettverket, og sub nettene.

Network.tf

```
network.tf > ...
1  resource "google_compute_network" "vpc_https_servers" {
2      name                = "vpc-https-servers"
3      auto_create_subnetworks = false
4  }
5
6  resource "google_compute_subnetwork" "subnet_europe" {
7      name                = "subnet-europe"
8      ip_cidr_range       = "10.10.0.0/24"
9      network              = google_compute_network.vpc_https_servers.name
10     region               = "europe-west1"
11 }
12
13 resource "google_compute_subnetwork" "subnet_north_america" {
14     name                = "subnet-north-america"
15     ip_cidr_range       = "10.20.0.0/24"
16     network              = google_compute_network.vpc_https_servers.name
17     region               = "us-central1"
18 }
19
20 resource "google_compute_subnetwork" "subnet_asia" {
21     name                = "subnet-asia"
22     ip_cidr_range       = "10.30.0.0/24"
23     network              = google_compute_network.vpc_https_servers.name
24     region               = "asia-east1"
25 }
26
27 resource "google_compute_subnetwork" "subnet_australia" {
28     name                = "subnet-australia"
29     ip_cidr_range       = "10.40.0.0/24"
30     network              = google_compute_network.vpc_https_servers.name
31     region               = "australia-southeast1"
32 }
```

Først ble det opprettet et Virtual Private Cloud (VPC)-nettverk kalt `vpc_https_servers`, som fungerer som en logisk isolert nettverksinfrastruktur. Dette nettverket er konfigurert til ikke å automatisk opprette subnett, slik at vi kan oppnå full kontroll over subnettene som defineres manuelt. Deretter ble det opprettet flere subnett, hvor hvert subnett representerer en spesifikk geografisk region og har en unik IP-adresseringsplan. For eksempel ble subnettet `subnet-europe` opprettet i regionen `eu-west-1` med IP-adresseområdet `10.10.0.0/24`. Dette betyr at subnettet har et begrenset antall interne IP-adresser som kan tildeles ressurser i subnettet, og alle ressurser i dette subnettet vil bli knyttet til VPC-nettverket. På samme måte ble det opprettet andre subnett i regionene `us-central-1` (Nord-Amerika), `ap-east-1` (Asia) og `ap-southeast-1` (Australia) med henholdsvis IP-adresseområdene `10.20.0.0/24`, `10.30.0.0/24` og `10.40.0.0/24`. Til tross for at subnettene er identiske i oppsett, varierer de i region og IP-adressering. Dette gir flere fordeler, blant annet global tilgjengelighet der applikasjoner og ressurser kan plasseres nærmere sluttbrukerne for å redusere ventetid. Det gir også bedre feiltoleranse, ettersom trafikk kan rutes til andre regioner dersom en region blir utilgjengelig, samt forbedret sikkerhetskontroll ved å implementere spesifikke brannmurregler for hver region. Totalt sett gir denne infrastrukturen muligheten til å distribuere applikasjoner globalt, segmentere nettverkstrafikk for økt sikkerhet og optimalisere kostnadene ved å kunne utnytte de mest kostnadseffektive regionene.

Derimot, etter å ha forsøkt å pushe koden, avdekket `tfsec` at VPC Flow Logs ikke var aktivert for subnettene. Dette er en viktig sikkerhetsfunksjon som gir detaljert innsikt i nettverkstrafikken og muliggjør både sanntidsanalyse og historisk gjennomgang av trafikkmønstre. Ved å aktivere VPC Flow Logs kan man identifisere uventede eller potensielt skadelige tilkoblinger, noe som er essensielt for å styrke sikkerheten i infrastrukturen. Dette gir mulighet for å avdekke uautorisert tilgang, overvåke intern trafikk mellom systemer, samt analysere potensielle forsøk på angrep som DDoS eller scanning av åpne porter. Videre muliggjør loggene bedre feilsøking ved å gi detaljert informasjon om hvilke IP-adresser som kommuniserer, hvilke protokoller som benyttes, og om det er spesifikke mønstre som bør undersøkes nærmere. Ved å inkludere logging på subnettnivå sikrer man også at nettverksanalyser ikke kun er begrenset til brannmurregler eller applikasjonsnivået, men at all trafikk, både inngående og utgående, blir grundig logget og kan analyseres. Dette er spesielt viktig i en distribuert global infrastruktur, hvor dataflyten skjer mellom flere regioner og over flere nettverk. Ved å aktivere VPC Flow Logs på alle subnett øker man dermed både sikkerheten og transparensen i nettverksinfrastrukturen, samtidig som man legger til rette for bedre compliance og revisjonsmuligheter ved å ha en detaljert loggføring av all nettverkstrafikk.

```

1 ▶ Run tfsec .
4 =====
5 tfsec is joining the Trivy family
6 tfsec will continue to remain available
7 for the time being, although our engineering
8 attention will be directed at Trivy going forward.
9 You can read more here:
0 https://github.com/aquasecurity/tfsec/discussions/1994
1 =====
6 Result #1 LOW Subnetwork does not have VPC flow logs enabled.
7
8 network.tf:13-18
9
10 13 resource "google_compute_subnetwork" "subnet_north_america" {
11 14     name           = "subnet-north-america"
12 15     ip_cidr_range = "10.20.0.0/24"
13 16     network        = google_compute_network.vpc_https_servers.name
14 17     region         = "us-central1"
15 18 }
16
17 ID google-compute-enable-vpc-flow-logs
18 Impact Limited auditing capability and awareness
19 Resolution Enable VPC flow logs
20
21 More Information
22 - https://aquasecurity.github.io/tfsec/v1.28.13/checks/google/compute/enable-vpc-flow-logs/
23 - https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute\_subnetwork#enable\_flow\_logs
24
25 Result #2 LOW Subnetwork does not have VPC flow logs enabled.
26
27 network.tf:20-25
28
29 20 resource "google_compute_subnetwork" "subnet_asia" {

```

Utklipp fra CI/CD Pipelinen som viser at tfsec oppdaget at VPC logs ikke var aktivert.

VPC Flow Logs ble deretter implementert, og som nevnt vil dette bidra til å øke sikkerheten i infrastrukturen.

```

network.tf > resource "google_compute_subnetwork" "subnet_north_america" > log
1  resource "google_compute_network" "vpc_https_servers" {
2      name                = "vpc-https-servers"
3      auto_create_subnetworks = false
4  }
5
6  resource "google_compute_subnetwork" "subnet_europe" {
7      name                = "subnet-europe"
8      ip_cidr_range       = "10.10.0.0/24"
9      network              = google_compute_network.vpc_https_servers.name
10     region              = "europe-west1"
11
12     log_config {
13         aggregation_interval = "INTERVAL_10_MIN"
14         flow_sampling        = 0.5
15         metadata              = "INCLUDE_ALL_METADATA"
16     }
17 }
18
19 resource "google_compute_subnetwork" "subnet_north_america" {
20     name                = "subnet-north-america"
21     ip_cidr_range       = "10.20.0.0/24"
22     network              = google_compute_network.vpc_https_servers.name
23     region              = "us-central1"
24
25     log_config {
26         aggregation_interval = "INTERVAL_10_MIN"
27         flow_sampling        = 0.5
28         metadata              = "INCLUDE_ALL_METADATA"
29     }
30 }
31
32 resource "google_compute_subnetwork" "subnet_asia" {
33     name                = "subnet-asia"
34     ip_cidr_range       = "10.30.0.0/24"
35     network              = google_compute_network.vpc_https_servers.name
36     region              = "asia-east1"
37
38     log_config {
39         aggregation_interval = "INTERVAL_10_MIN"
40         flow_sampling        = 0.5
41         metadata              = "INCLUDE_ALL_METADATA"
42     }
43 }
44
45 resource "google_compute_subnetwork" "subnet_australia" {
46     name                = "subnet-australia"
47     ip_cidr_range       = "10.40.0.0/24"
48     network              = google_compute_network.vpc_https_servers.name
49     region              = "australia-southeast1"
50
51     log_config {
52         aggregation_interval = "INTERVAL_10_MIN"
53         flow_sampling        = 0.5
54         metadata              = "INCLUDE_ALL_METADATA"
55     }
56 }

```

Utklipp som viser at VPC flow logs har blitt implementert.

Valgene for hvor ofte den logger og hvor mye er basert på at det skal være budsjett vennlig.

Først er `aggregation_interval` satt til `"INTERVAL_10_MIN"`, noe som betyr at loggene samles opp og skrives til Cloud Logging hvert tiende minutt. Dette reduserer mengden data som genereres, sammenlignet med kortere intervaller som `"INTERVAL_1_MIN"`, som ville økt kostnadene for logglagring betydelig. Deretter er `flow_sampling` satt til 0.5, noe som betyr at kun 50 % av trafikken logges, i motsetning til full logging (1.0). Dette gir tilstrekkelig innsikt i nettverkstrafikken uten at kostnadene blir for høye. Hadde dette vært et oppdrag med høyere budsjett, kunne full logging vært vurdert for enda bedre innsikt, men for vårt bruk gir en sampling på 50 % en god balanse mellom detaljnivå og kostnadseffektivitet.

Til slutt er metadata satt til `"INCLUDE_ALL_METADATA"`, noe som betyr at loggene inkluderer all tilgjengelig informasjon om nettverkstrafikken, som kilde- og destinasjons-IP, portnumre og nettverksprotokoller. Dette gir verdifull kontekst for sikkerhetsanalyser og feilsøking, men kan også medføre høyere lagringskostnader sammenlignet med alternativet `"INCLUDE_NONE"`. Gitt vårt behov for en global, distribuert infrastruktur hvor vi ønsker å ha innsikt i trafikkmønstre på tvers av regioner, er denne tilnærmingen en fornuftig løsning.

Etter dette, ved neste forsøk på å pushe koden, stoppet CI/CD-pipelinen på grunn av at Terraform-koden ikke var riktig formatert. For å løse dette ble kommandoen `terraform fmt -recursive` kjørt lokalt i mappen der Terraform-koden ligger. Dette sikret at alle Terraform-filene ble formatert i henhold til standardene, noe som forhindre ytterligere feil i pipelinen.

Compute-instance-groups.tf

```
compute-instance-groups.tf > resource "google_compute_instance_group_manager" "igm_north_america" > zone
1 resource "google_compute_instance_group_manager" "igm_europe" {
2   name           = "igm-europe"
3   base_instance_name = "vm-europe"
4   zone           = "europe-west1-b"
5
6   version {
7     instance_template = google_compute_instance_template.vm_template_europe.id
8   }
9
10  target_size = 1
11
12  named_port {
13    name = "http"
14    port = 80
15  }
16 }
17
18 resource "google_compute_instance_group_manager" "igm_north_america" {
19   name           = "igm-north-america"
20   base_instance_name = "vm-north-america"
21   zone           = "us-central1-a"
22
23   version {
24     instance_template = google_compute_instance_template.vm_template_north_america.id
25   }
26
27   target_size = 1
28
29   named_port {
30     name = "http"
31     port = 80
32   }
33 }
34
35 resource "google_compute_instance_group_manager" "igm_asia" {
36   name           = "igm-asia"
37   base_instance_name = "vm-asia"
38   zone           = "asia-east1-a"
39
40   version {
41     instance_template = google_compute_instance_template.vm_template_asia.id
42   }
43
44   target_size = 1
45
46   named_port {
47     name = "http"
48     port = 80
49   }
50 }
51
```

Utklipp av første del av Compute-instance-groups.tf

```

resource "google_compute_instance_group_manager" "igm_australia" {
  name           = "igm-australia"
  base_instance_name = "vm-australia"
  zone           = "australia-southeast1-a"

  version {
    instance_template = google_compute_instance_template.vm_template_australia.id
  }

  target_size = 1

  named_port {
    name = "http"
    port = 80
  }
}

```

Utklipp som viser del 2 av compute-instance-groups.tf

Compute-instance-groups.tf definerer flere Google Compute Instance Group Managers (IGM), som brukes til å administrere grupper av virtuelle maskiner. Hver instansegruppe er ansvarlig for å opprette og vedlikeholde en spesifisert mengde virtuelle maskiner (VMs) i en bestemt geografisk region. For å forstå hvordan dette fungerer, ser vi nærmere på en av instansegruppene, nemlig `igm_europe`. De andre instansegruppene følger samme struktur, men er tilpasset forskjellige regioner.

Instansegruppen `igm_europe` har navnet "igm-europe" og er konfigurert til å operere i sonen "europe-west1-b". Dette betyr at alle VM-instanser i denne gruppen vil bli plassert i denne spesifikke regionen i Google Cloud. I tillegg har gruppen en basisnavnkonfigurasjon satt til "vm-europe", noe som betyr at instansene som opprettes vil ha navn som starter med dette prefikset.

For å bestemme hvordan instansene skal opprettes, bruker instansegruppen en instansmal (`instance_template`). I dette tilfellet peker `igm_europe` på `google_compute_instance_template.vm_template_europe.id`, som er en forhåndsdefinert mal som spesifiserer konfigurasjonen av de virtuelle maskinene. Denne malen inneholder detaljene om operativsystem, maskinvareprofil og eventuelle nettverksinnstillinger som kreves for VM-instansene.

Videre er mål størrelsen (`target_size`) satt til 1, noe som betyr at instansegruppen alltid skal ha en aktiv instans. Hvis den eksisterende instansen feiler eller blir terminert, vil Google Cloud automatisk opprette en ny instans for å opprettholde dette antallet. Dette gir en grunnleggende form for redundans og høy tilgjengelighet.

Til slutt er en named port definert, der `name = "http"` og `port = 80`. Dette gjør at tjenesten som kjører på disse instansene kan eksponere HTTP-trafikk på port 80, slik at den blir tilgjengelig for eksterne forespørsler. Denne konfigurasjonen er nyttig for applikasjoner som kjører webtjenester og trenger en spesifikk portbinding for lastbalansering eller brannmurregler.

De andre instansegruppene i konfigurasjonen (igm_north_america, igm_asia, igm_australia) fungerer på nøyaktig samme måte, men er tilpasset forskjellige soner og bruker tilsvarende instansmaler for sine respektive regioner. Dette oppsettet sikrer en distribuert og geografisk redundant infrastruktur, der tjenesten er tilgjengelig på flere kontinenter.

Compute-instance-templates.tf

```
compute-instance-templates.tf > resource "google_compute_instance_template" "vm_template_australia" > metadata > start
1 resource "google_compute_instance_template" "vm_template_europe" {
2   name           = "vm-template-europe"
3   machine_type   = "e2-micro"
4
5   tags = ["http", "lb"]
6
7   disk {
8     source_image      = "debian-cloud/debian-12"
9     auto_delete       = true
10    resource_policies = [google_compute_resource_policy.backup_policy_europe.id]
11  }
12
13  shielded_instance_config {
14    enable_secure_boot = true
15  }
16
17  metadata = {
18    startup-script = <<-EOT
19    #!/bin/bash
20    apt-get update
21    apt-get install -y nginx
22    cat <<EOF > /var/www/html/index.nginx-debian.html
23    <html>
24    <head><title> Nova Tech Solutions</title> </head>
25    <body>
26      <h1>Welcome to NovaTech Solutions</h1>
27      <h2>You are connected to the <strong>European</strong> server</h2>
28    </body>
29    </html>
30    EOF
31    systemctl enable nginx
32    systemctl start nginx
33  EOT
34  }
35
36  network_interface {
37    network      = google_compute_network.vpc_https_servers.self_link
38    subnetwork   = google_compute_subnetwork.subnet_europe.self_link
39    access_config {}
40  }
41 }
42
```

Utklipp fra compute-instance-templates.tf

For å spare plass har jeg valgt og bare legge ved et utklipp ettersom at alle instanse templatesene er helt like utenom at de bruker forskjellige subnet basert på regionen de skal deployes i.

Instansmalen vm_template_europe får navnet "vm-template-europe" og spesifiserer at den virtuelle maskinen skal bruke maskintypen "e2-micro", en kostnadseffektiv instans med lav ressursbruk. I tillegg er det angitt tags ["http", "lb"], som kan brukes til å anvende brannmurregler eller lastbalansering.

Videre defineres konfigurasjonen for den primære disken til VM-en. Diskens `source_image` er satt til `"debian-cloud/debian-12"`, noe som betyr at instansen vil kjøre Debian 12 som operativsystem. Den er også konfigurert til å automatisk slettes (`auto_delete = true`) når instansen fjernes. I tillegg er en ressurspolicy for backup (`resource_policies`) tilknyttet, som sørger for at instansen har en definert backup-plan.

For å forbedre sikkerheten aktiveres Secure Boot gjennom `shielded_instance_config`. Secure Boot bidrar til å forhindre at ondsinnet programvare laster inn under oppstarten av systemet.

Instansmalen inkluderer også et startup-script, som automatisk kjører når instansen opprettes. Dette skriptet oppdaterer pakkeindeksen, installerer Nginx som webserver og oppretter en HTML-side med en velkomstmelding. Siden spesifiserer at brukeren er tilkoblet den europeiske serveren, slik at hver region har en unik melding.

Til slutt konfigureres nettverksgrensesnittet. Instansen kobles til et spesifikt VPC-nettverk (`google_compute_network.vpc_https_servers.self_link`) og tildeles en subnet-adresse for Europa (`google_compute_subnetwork.subnet_europe.self_link`). En `access_config` legges også til, noe som sikrer at instansen får en offentlig IP-adresse og dermed kan nå eksternt.

Denne instansmalen sikrer at alle VM-er som opprettes i Europa har samme konfigurasjon. De andre instansmalene (`vm_template_north_america`, `vm_template_asia`, `vm_template_australia`) fungerer på samme måte, men bruker ulike subnett og viser en tilpasset velkomstmelding for hver region.

Backup.tf

```

backup.tf > ...
1  resource "google_compute_resource_policy" "backup_policy_europe" {
2      name     = "daily-backup-policy-eu"
3      region  = "europe-west1"
4
5      snapshot_schedule_policy {
6          schedule {
7              daily_schedule {
8                  days_in_cycle = 1
9                  start_time    = "03:00"
10             }
11         }
12
13         retention_policy {
14             max_retention_days = 7
15             on_source_disk_delete = "KEEP_AUTO_SNAPSHOTS"
16         }
17
18         snapshot_properties {
19             labels = {
20                 environment = "production"
21                 service      = "http-backend"
22             }
23         }
24     }
25 }

```

Utklipp som viser 1 av 4 backup policies.

Backup-policyen `backup_policy_europe` får navnet `"daily-backup-policy-eu"` og er satt til å operere i regionen `"europe-west1"`. Denne policyen definerer en snapshot-skjema under `snapshot_schedule_policy`, som spesifiserer når og hvordan sikkerhetskopiene skal tas. Planen er konfigurert med en daglig backup (`daily_schedule`), hvor `days_in_cycle = 1` betyr at en ny snapshot blir opprettet hver dag. Tidspunktet for backup er satt til 03:00, for å minimere påvirkning på systemytelsen ved å kjøre backupen på et tidspunkt med lav trafikk.

For å administrere lagringsbehov defineres en retensjonspolicy (`retention_policy`). Her er `max_retention_days = 7`, noe som betyr at maksimalt 7 dagers verdt av snapshots blir lagret. Når nye snapshots tas, vil eldre snapshots automatisk slettes for å sikre at systemet ikke bruker unødvendig lagringsplass. Videre er `on_source_disk_delete` satt til `"KEEP_AUTO_SNAPSHOTS"`, noe som sikrer at snapshots bevares selv om den originale disken blir slettet. Dette kan være nyttig for å gjenopprette en slettet instans fra en tidligere snapshot.

Backup-policyen inkluderer også metadata-labels under `snapshot_properties`, hvor snapshotene blir merket med `environment = "production"` og `service = "http-backend"`. Disse labelene gjør det enklere å organisere og identifisere snapshots basert på hvilken tjeneste de tilhører, noe som er spesielt nyttig i større infrastrukturer med flere tjenester og miljøer.

De andre backup-policyene (`backup_policy_north_america`, `backup_policy_asia`, `backup_policy_australia`) er identiske i funksjonalitet, men er satt opp for ulike regioner. Hver av disse policyene sikrer at de respektive VM-instansene har daglige snapshots, som gir en pålitelig backup-løsning med en 7-dagers historikk. Dette oppsettet sikrer høy tilgjengelighet og rask gjenoppretting dersom noe skulle gå galt med en instans i en bestemt region.

Loadbalancer.tf

```
loadbalancer.tf > ...
1  resource "google_compute_url_map" "https_url_map" {
2      name          = "global-url-map"
3      default_service = google_compute_backend_service.http_backend.id
4  }
5
6  resource "google_compute_target_https_proxy" "https_proxy" {
7      name          = "global-https-proxy"
8      url_map        = google_compute_url_map.https_url_map.id
9      ssl_certificates = [google_compute_managed_ssl_certificate.default.self_link]
10 }
11
12 resource "google_compute_global_forwarding_rule" "https_forwarding_rule" {
13     name          = "https-forwarding-rule"
14     target        = google_compute_target_https_proxy.https_proxy.id
15     port_range    = "443"
16 }
```

Utklipp som viser loadbalancer.tf

Denne Terraform-konfigurasjonen setter opp en global HTTPS-lastbalanser i Google Cloud, som sikrer at innkommende HTTPS-trafikk håndteres på en strukturert måte. Oppsettet består av tre hovedkomponenter: en URL Map, en Target HTTPS Proxy, og en Global Forwarding Rule, som sammen sørger for at forespørsler blir terminert på HTTPS, analysert, og deretter videresendt til riktig backend-tjeneste.

Først defineres en URL Map gjennom `google_compute_url_map`. Denne ressursen bestemmer hvordan innkommende forespørsler skal rutes til backend-tjenester. I dette tilfellet er `default_service` satt til `google_compute_backend_service.http_backend.id`, noe som betyr at all trafikk som ikke samsvarer med spesifikke regler automatisk blir sendt til en standard backend-tjeneste. En URL Map kan utvides med mer avanserte regler, som for eksempel å sende forskjellige URL-stier til forskjellige backend-servere.

Deretter opprettes en Target HTTPS Proxy ved hjelp av `google_compute_target_https_proxy`. Denne proxyen fungerer som et mellomledd mellom klientene og backend-serverne, hvor den tar imot HTTPS-trafikk, terminerer SSL, og ruter forespørslene videre basert på URL-kartleggingen. For å sikre kryptert kommunikasjon, er proxyen konfigurert med et SSL-sertifikat, som i dette tilfellet er et Google-administrert sertifikat (`google_compute_managed_ssl_certificate.default.self_link`). Dette gjør det enklere å administrere HTTPS, ettersom Google håndterer fornyelse og distribusjon av sertifikatene automatisk.

Til slutt defineres en Global Forwarding Rule ved hjelp av `google_compute_global_forwarding_rule`. Denne regelen sørger for at innkommende HTTPS-forespørsler på port 443 blir sendt videre til `google_compute_target_https_proxy.https_proxy.id`. Forwarding-regelen fungerer som en inngangsport for all HTTPS-trafikk og sikrer at forespørslene blir behandlet av lastbalanseren på en effektiv måte.

Når en klient sender en HTTPS-forespørsel til en nettside som bruker dette oppsettet, blir forespørselen først fanget opp av Global Forwarding Rule, som lytter på port 443. Den sender deretter trafikken videre til Target HTTPS Proxy, som håndterer SSL-terminering og sørger for at forespørselen er kryptert og gyldig. Proxyen bruker URL Map til å bestemme hvilken backend-tjeneste forespørselen skal sendes til, og sender den videre til riktig server. Til slutt returnerer backend-serveren et svar som sendes tilbake til klienten via den samme ruten.

SSL.tf

```
ssl.tf > ...
1 resource "google_compute_managed_ssl_certificate" "default" {
2   name = "managed-ssl-cert"
3
4   managed {
5     domains = [
6       "wrathbyte.com",
7       "www.wrathbyte.com"
8     ]
9   }
10 }
```

Utklipp fra ssl.tf

Jeg har valgt å bruke et av mine egne domener til dette prosjektet som et eksempel på hvordan man kan sette opp HTTPS på en last-balanser. Jeg har brukt cloud flare til å koble domenet opp mot last-balanseren sin IP.

<input type="checkbox"/>	A	wrathbyte.com	34.8.205.4	 DNS only	Auto	Edit
<input type="checkbox"/>	A	www	34.8.205.4	 DNS only	Auto	Edit

Utklipp fra CloudFlare som viser at domenet peker mot last-balanseren sin IP.

SSL.tf definerer en Google Managed SSL Certificate, som brukes til å aktivere HTTPS uten å måtte administrere SSL-sertifikater manuelt. Ved å bruke Google-administrerte SSL-sertifikater, håndterer Google både utstedelse, fornyelse og distribusjon av sertifikatene automatisk.

Ressursen `google_compute_managed_ssl_certificate "default"` oppretter et nytt administrert SSL-sertifikat med navnet `"managed-ssl-cert"`. Dette sertifikatet kan deretter knyttes til en HTTPS-proxy eller en lastbalanser for å muliggjøre kryptert trafikk mellom klienter og tjenesten.

I konfigurasjonen er det en `managed`-blokk som spesifiserer domenene som sertifikatet skal gjelde for. Her er `"wrathbyte.com"` og `"www.wrathbyte.com"` oppgitt, noe som betyr at sertifikatet vil dekke både hoveddomenet og dets `www`-subdomene. Google vil automatisk hente og validere sertifikatet ved å sjekke at disse domenene peker til Google Cloud.

Når dette oppsettet er på plass, sørger Google for at sertifikatet fornyes automatisk før det utløper, slik at tjenesten alltid har et gyldig SSL-sertifikat. Dette eliminerer behovet for manuell sertifikatadministrasjon og gir en sikker, vedlikeholdsfri HTTPS-løsning for domenet.

Firewall.tf

```
firewall.tf > ...
1 resource "google_compute_firewall" "allow_http" {
2   name     = "allow-http"
3   network = google_compute_network.vpc_https_servers.name
4
5   allow {
6     protocol = "tcp"
7     ports    = ["80"]
8   }
9
10  # tfsec:ignore:google-compute-no-public-ingress Reason: Only Google Cloud Load Balancer IP ranges are allowed.
11  source_ranges = ["130.211.0.0/22", "35.191.0.0/16"]
12  target_tags   = ["http"]
13 }
```

Utklipp av firewall.tf

Ressursen `google_compute_firewall "allow_http"` oppretter en ny regel med navnet "allow-http", og den er knyttet til et eksisterende VPC-nettverk (`google_compute_network.vpc_https_servers.name`). Dette betyr at regelen kun gjelder for instanser som er en del av dette spesifikke nettverket.

Regelen spesifiserer at den skal tillate TCP-trafikk på port 80. Dette gjøres gjennom allow-blokken, hvor `protocol = "tcp"` og `ports = ["80"]` angir at kun HTTP-trafikk skal være tillatt. Dette gjør det mulig for HTTP-servere, som for eksempel en Nginx- eller Apache-instans, å motta forespørsler fra eksterne brukere.

Under en tfsec-sikkerhetsskanning ble det rapportert en potensiell sikkerhetsrisiko knyttet til tillatelse av offentlig innkommende trafikk (`google-compute-no-public-ingress`). Dette skyldes at regelen åpner for eksterne IP-adresser. Likevel er `source_ranges` begrenset til Google Cloud Load Balancer IP-adresser, noe som sikrer at kun trafikk som rutes gjennom lastbalanseren får tilgang. Dette samsvarer med Google Cloud best practices og utgjør ikke en sikkerhetsrisiko i dette tilfellet. For mer informasjon om disse IP-adressene, se den offisielle Google-dokumentasjonen:

[Google Cloud Load Balancer IP-adresser](#)

For å begrense hvem som kan sende trafikk til instansene, er `source_ranges` satt til "130.211.0.0/22" og "35.191.0.0/16". Disse IP-adressene er spesifikke for Google Cloud Load Balancers, noe som betyr at kun trafikk som kommer via lastbalanseren vil bli tillatt. Dette forhindrer direkte tilgang til VM-instanser fra tilfeldige eksterne kilder, og sikrer at all trafikk går gjennom en kontrollert og overvåket inngangsvei.

Til slutt brukes `target_tags = ["http"]` for å begrense regelen til kun å gjelde for instanser som har "http"-merket. Dette sørger for at ikke alle instanser i nettverket blir påvirket av regelen, men kun de som spesifikt er konfigurert for å håndtere HTTP-trafikk. Dette bidrar til en mer granulær sikkerhetsstyring, der brannmurregler bare gjelder for relevante ressurser.

Health_check.tf

```
health_check.tf > ...
1  resource "google_compute_http_health_check" "health_check" {
2      name                = "http-health-check"
3      check_interval_sec  = 10
4      timeout_sec         = 5
5      healthy_threshold    = 2
6      unhealthy_threshold = 2
7      request_path        = "/"
8  }
```

Utklipp av health_check.tf.

Ressursen `google_compute_http_health_check "health_check"` oppretter en HTTP-basert helsesjekk med navnet `"http-health-check"`. Denne sjekken brukes til å kontinuerlig overvåke backend-tjenestene ved å sende regelmessige HTTP-forespørsler til dem. Dersom en instans ikke svarer riktig på disse forespørslene, vil den bli ansett som ustabil, og lastbalanseren vil slutte å sende trafikk til den inntil den gjenoprettes.

Parameteren `check_interval_sec = 10` spesifiserer at Google Cloud skal sende en ny helsesjekk hvert 10. sekund. Dette betyr at systemet vil overvåke tjenesten kontinuerlig med korte intervaller for raskt å oppdage eventuelle feil. Samtidig er `timeout_sec = 5`, noe som betyr at en instans får 5 sekunder på seg til å svare på helsesjekken før den anses som ikke responderende.

For å bestemme når en instans regnes som sunn eller usunn, brukes `healthy_threshold = 2` og `unhealthy_threshold = 2`. `healthy_threshold = 2` betyr at en instans må bestå to påfølgende helsesjekker for å bli ansett som sunn. På samme måte betyr `unhealthy_threshold = 2` at en instans må feile to helsesjekker på rad før den regnes som usunn. Disse tersklene hjelper med å unngå falske positive og sikrer at en kortvarig feil ikke automatisk fjerner en instans fra rotasjonen.

Til slutt definerer `request_path = "/"` hvilken URL-sti helsesjekken skal bruke. I dette tilfellet er forespørselen rettet mot rotstien (`/`), noe som betyr at den sjekker om standard websiden eller API-endepunktet på serveren svarer riktig. Om serveren returnerer en 2xx eller 3xx HTTP-statuskode, blir den betraktet som sunn. Om den returnerer en 5xx-statuskode eller ikke svarer innen timeout-grensen, blir den ansett som usunn.

Backend.tf

```
backend.tf > resource "google_compute_backend_service" "http_backend" > backend
1  resource "google_compute_backend_service" "http_backend" {
2      name                = "http-backend-service"
3      description         = "Backend service for HTTP traffic"
4      protocol            = "HTTP"
5      port_name           = "http"
6      timeout_sec         = 30
7      security_policy      = google_compute_security_policy.web_security_policy.id
8
9      backend {
10         group = google_compute_instance_group_manager.igm_europe.instance_group
11     }
12
13     backend {
14         group = google_compute_instance_group_manager.igm_north_america.instance_group
15     }
16
17     backend {
18         group = google_compute_instance_group_manager.igm_asia.instance_group
19     }
20
21     backend {
22         group = google_compute_instance_group_manager.igm_australia.instance_group
23     }
24
25     health_checks = [google_compute_http_health_check.health_check.self_link]
26 }
27
```

Utklipp av Backend.tf

Ressursen `google_compute_backend_service "http_backend"` oppretter en backend-tjeneste med navnet `"http-backend-service"`. Den er konfigurert til å håndtere HTTP-trafikk, angitt med `protocol = "HTTP"`. `port_name = "http"` spesifiserer at trafikken vil rutes gjennom en port som har dette navnet definert i instansgruppene.

For å sikre at tjenesten er robust og kan håndtere forsinkelser i forespørsler, er `timeout_sec = 30` satt. Dette betyr at hvis en forespørsel til en backend-instans tar mer enn 30 sekunder å svare, vil den bli avbrutt.

Et viktig sikkerhetstiltak er `security_policy = google_compute_security_policy.web_security_policy.id`, som knytter backend-tjenesten til en Google Compute Security Policy. Denne policyen er satt opp for å beskytte mot en rekke angrep.

Videre er det fire backend-blokker, som definerer hvilke instansegrupper som skal motta trafikk. Her legges instansegruppene fra fire geografiske regioner til:

- `igm_europe.instance_group` (Europa)

- `igm_north_america.instance_group` (Nord-Amerika)
- `igm_asia.instance_group` (Asia)
- `igm_australia.instance_group` (Australia)

Disse instansegruppene inneholder de virtuelle maskinene (VM-ene) som håndterer HTTP-forespørsler. Lastbalanseren kan dermed fordele trafikken på tvers av regionene, noe som sikrer geografisk redundans og bedre ytelse for brukere over hele verden.

Til slutt er `health_checks = [google_compute_http_health_check.health_check.self_link]` angitt, noe som betyr at backend-tjenesten er knyttet til helsesjekken.

Autoscaler.tf

```

1 resource "google_compute_autoscaler" "autoscaler_europe" {
2   name     = "autoscaler-europe"
3   zone     = "europe-west1-b"
4   target   = google_compute_instance_group_manager.igm_europe.id
5
6   autoscaling_policy {
7     max_replicas = 5
8     min_replicas = 1
9     cpu_utilization {
10      target = 0.6
11    }
12  }
13 }
14
15 resource "google_compute_autoscaler" "autoscaler_north_america" {
16   name     = "autoscaler-north-america"
17   zone     = "us-central1-a"
18   target   = google_compute_instance_group_manager.igm_north_america.id
19
20   autoscaling_policy {
21     max_replicas = 5
22     min_replicas = 1
23     cpu_utilization {
24      target = 0.6
25    }
26  }
27 }

```

Utklipp av autoscaler.tf

Her har jeg kun satt opp to autoskalere, en for Europa (`autoscaler_europe`) og en for Nord-Amerika (`autoscaler_north_america`). Dette er fordi jeg har estimert at disse regionene vil ha den høyeste trafikken i startfasen. Hvis autoskalering også blir nødvendig i Asia og Australia, kan en av kodeblokkene enkelt kopieres og tilpasses ved å peke til de riktige instansegruppene i disse regionene.

Autoskaleren `google_compute_autoscaler "autoscaler_europe"` er konfigurert til å håndtere instanser i Europa, spesifikt i sonen `"europe-west1-b"`. Den peker til `google_compute_instance_group_manager.igm_europe.id`, som er instansegruppen for denne regionen.

Autoskaleringen styres gjennom `autoscaling_policy`. Her er minimum antall instanser satt til 1, noe som betyr at det alltid vil være minsten aktiv instans. Samtidig er maksimum antall instanser satt til 5, slik at systemet kan skalere opp til fem instanser hvis belastningen øker.

Grunnen til at maksimum er satt til 5 er basert på budsjettbegrensninger, men dette kan enkelt justeres ved behov dersom tjenesten krever mer kapasitet.

Autoskaleren bruker CPU-belastning (`cpu_utilization`) som kriterium, hvor `target = 0.6` betyr at autoskaleren vil opprette flere instanser dersom gjennomsnittlig CPU-bruk overstiger 60%. Når belastningen reduseres, vil unødvendige instanser fjernes automatisk for å optimalisere kostnadene.

Den samme konfigurasjonen brukes for `autoscaler_north_america`, men denne peker til instansegruppen for Nord-Amerika, som ligger i sonen "us-central1-a". Autoskaleringen følger de samme prinsippene, med en minimumsgrense på 1 instans, en maksimumsgrense på 5 instanser, og CPU-belastning som avgjør når flere instanser skal opprettes.

```

security.tf > resource "google_compute_security_policy" "web_security_policy" > rule > description
1 resource "google_compute_security_policy" "web_security_policy" {
2   name      = "web-security-policy"
3   description = "Security policy protecting against SQL injection, XSS, RCE, LFI, and brute-force attacks"
4
5   # Block SQL injection attempts
6   rule {
7     priority      = 1000
8     action        = "deny(403)"
9     description   = "Block SQL injection attempts"
10
11     match {
12       expr {
13         expression = "evaluatePreconfiguredWaf('sqli-stable')"
14       }
15     }
16   }
17
18   # Block XSS attacks
19   rule {
20     priority      = 2000
21     action        = "deny(403)"
22     description   = "Block XSS attacks"
23
24     match {
25       expr {
26         expression = "evaluatePreconfiguredWaf('xss-stable')"
27       }
28     }
29   }
30
31   # Block Local File Inclusion (LFI) attacks
32   rule {
33     priority      = 2500
34     action        = "deny(403)"
35     description   = "Block Local File Inclusion (LFI) attacks"
36
37     match {
38       expr {
39         expression = "evaluatePreconfiguredWaf('lfi-stable')"
40       }
41     }
42   }
43
44   # Block Remote Code Execution (RCE) attacks
45   rule {
46     priority      = 2600
47     action        = "deny(403)"
48     description   = "Block Remote Code Execution (RCE) attacks"
49
50     match {
51       expr {
52         expression = "evaluatePreconfiguredWaf('rce-stable')"
53       }
54     }
55   }

```

Utklipp 1 av 2 security.tf.

```

# Throttle rule: if an IP exceeds 10 requests per minute, block with a 403
rule {
  priority    = 3000
  action      = "throttle"
  description = "Throttle requests exceeding 10 per minute per IP"
  match {
    versioned_expr = "SRC_IPS_V1"
    config {
      src_ip_ranges = ["*"]
    }
  }
  rate_limit_options {
    conform_action = "allow"
    exceed_action  = "deny(403)"

    enforce_on_key = ""
    enforce_on_key_configs {
      enforce_on_key_type = "IP"
    }
    rate_limit_threshold {
      count      = 10
      interval_sec = 60
    }
  }
}

# Default allow rule
rule {
  priority    = 2147483647
  action      = "allow"
  description = "Default rule"

  match {
    versioned_expr = "SRC_IPS_V1"
    config {
      src_ip_ranges = ["*"]
    }
  }
}

```

Utklipp 2 av 2 security.tf

Her har jeg gått litt utenfor planen og lagt til flere regler for å øke sikkerheten enda mer.

Den første regelen har en prioritet på 1000 og blokkerer SQL-injeksjonsforsøk ved hjelp av en prekonfigurert Web Application Firewall (WAF). Dette sikrer at forespørsler som inneholder kjente SQL-injeksjonsmønstre blir oppdaget og avvist med en HTTP 403 (Forbidden)-respons. Deretter følger en regel med prioritet 2000 som beskytter mot Cross-Site Scripting (XSS). Dette gjøres ved å evaluere forespørselen med WAF-en, og dersom den inneholder XSS-mønstre, blir den også avvist med en 403-feil.

Videre har policyen en regel med prioritet 2500 som stopper Local File Inclusion (LFI)-angrep. Dette er angrep hvor en angriper prøver å inkludere filer fra serveren for å få tilgang til sensitiv informasjon eller utføre skadelige handlinger. Dersom forespørselen inneholder et mønster som samsvarer med LFI-angrep, blir den blokkert med en 403-feil. Like etterpå følger en regel med prioritet 2600 som blokkerer Remote Code Execution (RCE). RCE-angrep kan gjøre det mulig for en angriper å kjøre vilkårlig kode på serveren, noe som utgjør

en alvorlig sikkerhetsrisiko. WAF-en analyserer forespørselen og avviser den dersom den inneholder kjente RCE-mønstre.

For å begrense muligheten for brute-force-angrep og overbelastning, er det også lagt inn en rate limiting-regel med prioritet 3000. Denne regelen begrenser antallet forespørsler per IP til maksimalt 10 per minutt. Hvis en IP-adresse overskrider denne grensen, blir forespørslene automatisk blokkert med en 403-feil. Dette bidrar til å forhindre misbruk av systemet, spesielt fra automatiserte bots eller angripere som prøver å overvelde tjenesten med forespørsler. Dette er en veldig simpel regel, og dette er bare for å vise at det er mulig gjennom waf. Det anbefales å implementere enda flere regler for og beskytte mot ddos. Ved oppgradering til Cloud Armor enterprise kommer det med DDoS beskyttelse, det er også mulig å bruke CloudFlare hvor det kommer med grunnleggende ddos beskyttelse gratis.

Til slutt inneholder policyen en standard tillatelsesregel med prioritet 2147483647, som sikrer at alle forespørsler som ikke matches av de tidligere reglene, blir tillatt. Dette betyr at legitime brukere fortsatt kan få tilgang til nettsiden uten problemer, samtidig som skadelige forespørsler blir blokkert. Samlet sett gir denne sikkerhetspolicyen en god grunnbeskyttelse mot vanlige webangrep og sikrer at tjenesten forblir tilgjengelig for legitime brukere, samtidig som den stopper forsøk på uautorisert tilgang eller overbelastning.

Provider.tf



```
provider.tf > ...
1  provider "google" {
2      project = "konte-eksamen-eksempel1"
3      region  = "europe-west1"
4  }
```

Utklipp av provider.tf

Først spesifiseres provider "google", som angir at Terraform skal bruke Google Cloud som leverandør. Dette er nødvendig for at Terraform skal kunne administrere ressurser innenfor GCP-økosystemet. Deretter defineres project, som angir hvilket GCP-prosjekt Terraform skal jobbe med. I dette eksempelet er prosjektet satt til "konte-eksamen-eksempel1", noe som betyr at alle Terraform-operasjoner vil bli utført i dette spesifikke prosjektet.

Videre spesifiseres region, som er satt til "europe-west1". Dette betyr at ressursene som Terraform oppretter, som for eksempel virtuelle maskiner, lbøtter eller databaser, vil bli plassert i denne geografiske regionen med mindre en annen region er valgt i konfigurasjonen av ressursen. Regionen europe-west1 refererer til Googles datasentre i Vest-Europa, nærmere bestemt i Belgia. Valg av region påvirker både ytelse og tilgjengelighet, i tillegg til at det kan ha kostnadmessige og regulatoriske implikasjoner.

Bucket.tf

```
bucket.tf > terraform
1 terraform {
2   backend "gcs" {
3     bucket = "sky-konte-eksamen"
4     prefix = "terraform/state"
5   }
6 }
```

Utklipp av bucket.tf

Denne koden konfigurerer Terraform til å bruke en ekstern backend i Google Cloud Storage for å lagre tilstandsinformasjonen. Ved å spesifisere backend-typen "gcs" settes lagringsstedet til en GCS-bøtte med navnet "sky-konte-eksamen". I tillegg definerer "prefix" verdien "terraform/state" den mappen eller sti i bøtten der Terraform tilstandsfilene plasseres, noe som bidrar til å organisere og strukturere lagringen. Denne konfigurasjonen sikrer at tilstandsinformasjonen, som er avgjørende for styring av infrastruktur, blir lagret på en sikker og sentralisert måte, ettersom GCS-bøtter som standard er private og kun tilgjengelige for autoriserte brukere.

Jeg laget bøtten manuelt, men har vedlagt dokumentasjon på hvordan, og hvorfor visse valg er gjort.

Permissions

Access control	Uniform 
Public access prevention 	Enabled via bucket setting
Public access status 	Not public

Protection

Soft delete policy	7 days
Object versioning	Off
Bucket retention policy	None
Object retention	Disabled
Encryption type	Google-managed 

Object lifecycle

Lifecycle rules	None
-----------------	------





Utklipp som viser at bucketen bruker Uniform acces control.

Valget om å bruke Google Managed Keys fremfor Customer Managed Keys (KMS) er basert på en strategisk vurdering av prosjektets behov og prioriteringer. I denne tidlige fasen er det viktig å fokusere ressursene på de områdene av infrastrukturen som gir størst verdi og stabilitet. Selv om KMS gir økt kontroll over nøkkeladministrasjon, krever det også mer operasjonell overhead i form av konfigurasjon, nøkkelrotasjon og tilgangsstyring. Ved å bruke Google Managed Keys overlates denne administrasjonen til Google, som sikrer at nøklene håndteres i tråd med bransjestandarder for sikkerhet og compliance, samtidig som det reduserer risikoen for feilkonfigurerings. Dette gjør at vi kan allokere mer tid og ressurser til andre kritiske deler av prosjektet.

Sikkerheten blir likevel godt ivaretatt, ettersom Google Managed Keys er en del av Google Cloud KMS og følger beste praksis med automatisk rotasjon og beskyttelse av data i samsvar med ISO 27001, SOC 2 og GDPR. Selv om vi på sikt planlegger en overgang til KMS for økt kontroll og tilpasning, er Google Managed Keys et hensiktsmessig valg i denne fasen, ettersom det gir et godt balanseforhold mellom sikkerhet, administrasjon og ressursbruk.

I tillegg har vi valgt å bruke uniform access control for bucketen, noe som er anbefalt som best practice, blant annet av tfsec. Uniform tilgangskontroll sikrer en konsistent og forutsigbar tilgangsstyring ved å håndheve IAM-policyer på bucket-nivå i stedet for på objektnivå. Dette reduserer kompleksiteten i administrasjonen av tilgangsrettigheter og minimerer risikoen for feilkonfigurerte tillatelser, noe som igjen styrker den generelle sikkerheten i prosjektet.

Implementering av CloudFlare

	A	wrathbyte.com	34.54.39.96	 Proxied	Auto	Edit
	A	www	34.54.39.96	 Proxied	Auto	Edit

Utklipp som viser at domenet "wrathbyte.com" peker mot last balanserens sin ip.

Jeg har konfigurert domenet wrathbyte.com til å peke mot lastbalanserens IP-adresse ved hjelp av DNS-innstillinger. Dette innebærer at en A-post for domenet er satt opp slik at trafikk til wrathbyte.com rutes direkte til IP-adressen til lastbalanseren.

I tillegg er det aktivert en proxy-løsning som fungerer som et mellomledd mellom klientforespørsler og backend-tjenestene. Denne proxyen maskerer den faktiske IP-adressen til lastbalanseren, noe som reduserer risikoen for at den blir utsatt for direkte angrep. Ved å skjule den eksponerte IP-adressen, bidrar proxyen til en økt sikkerhet og gjør infrastrukturen mindre sårbar for uautorisert tilgang.

Videre inkluderer løsningen gratis grunnleggende DDoS-beskyttelse. Denne beskyttelsen hjelper til med å dempe effekten av distribuerte tjenestenektangrep, noe som sikrer at tjenesten opprettholder tilgjengelighet selv under angrep. Samlet sett bidrar denne

kombinasjonen av DNS-konfigurasjon, proxy og DDoS-beskyttelse til en robust og sikker drift

Oppgave A. 2

Det er forskjellige typer logger i Google cloud. Disse er som følger:

1. Cloud Audit Logs

Cloud Audit logs sporer handlinger utført av GCP-ressurser og gir innsyn i endringer og tilgangskontroll. De tre hovedtypene av revisjonslogger er:

- **Admin activity logs:** Registrerer endringer på GCP-ressurser (f.eks opprettelse eller sletting av en VM).
- **Data Access Logs:** Fanger opp leseoperasjoner på ressurser (f.eks tilgang til Cloud Storage).
- **System Event logs:** Dokumenterer systemgenererte handlinger i GCP (f.eks autoskalering).

2. VPC Flow Logs

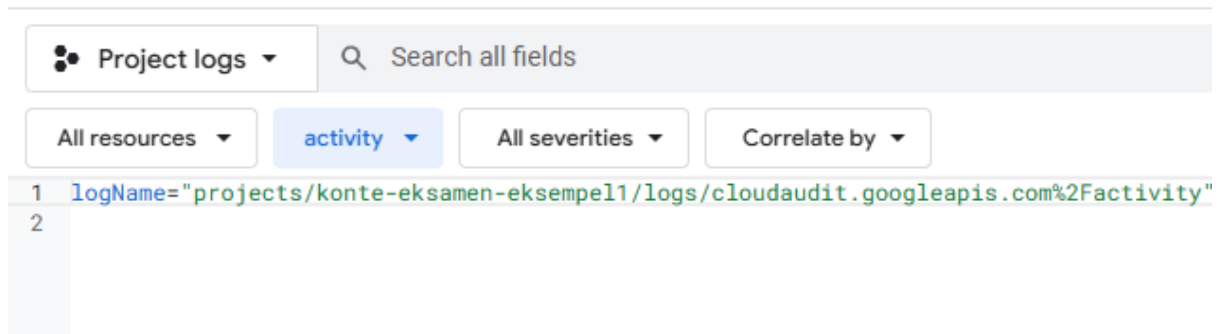
Vpc Flow logs gir innsyn i nettverkstrafikken, inkludert detaljer om source og destination ip, porter, overførte byte og tilkoblingsstatus. Jeg skrudde da på vpc logs i terraform konfigurasjonen.

3. Cloud Load balancer og brannmurlogger

Disse loggene fanger opp HTTP(S)-forespørsler, blokkerte tilkoblinger og tillatt trafikk.

I GCP er det noe som heter log explorer. Log Explorer samler inn logger fra ulike GCP-tjenester, inkludert Compute Engine, Kubernetes Engine, Cloud Functions, Cloud Load Balancing og mange flere. Dette gjør det mulig å overvåke systemytelse, feilsøke problemer og analysere sikkerhetshendelser på ett sentralt sted. Brukere kan filtrere loggene basert på kriterier som tidsperiode, ressursnavn, tjenestetype, alvorlighetsgrad og spesifikke tekststrenger, noe som gjør det enklere å finne relevant informasjon raskt.

Logs Explorer



Utklipp som viser et filter som sorterer etter Cloud Audit logs.

```
{
  insertId: "1zebfydcrs2"
  labels: {1}
  logName: "projects/konte-eksamen-eksempel1/logs/cloudaudit.googleapis.com%2Factivity"
  operation: {3}
  protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    authenticationInfo: {3}
    methodName: "v1.compute.securityPolicies.insert"
    request: {1}
    requestMetadata: {4}
    resourceName: "projects/konte-eksamen-eksempel1/global/securityPolicies/web-security-policy"
    serviceName: "compute.googleapis.com"
  }
  receiveTimestamp: "2025-02-17T14:18:26.327442888Z"
  resource: {2}
  severity: "NOTICE"
  timestamp: "2025-02-17T14:18:25.871630Z"
}
```

Utklipp som viser en Cloud Audit log entry.

Loggoppføringen indikerer at en sikkerhetspolicy med navnet "web-security-policy" ble opprettet i prosjektet konte-eksamen-eksempel1 via compute.googleapis.com API-et. Denne opprettelsen ble logget med NOTICE-nivå, noe som betyr at hendelsen er en normal operasjonell aktivitet og ikke en feil.

NOTICE-nivået brukes ofte til å logge hendelser som er viktige å dokumentere, men som ikke krever umiddelbar oppmerksomhet eller tiltak. I dette tilfellet viser loggen at systemet har utført en forventet administrativ handling, nemlig å opprette en sikkerhetspolicy, noe som bidrar til å forbedre og opprettholde sikkerheten i infrastrukturen. Hendelsen blir registrert for å gi et spor av administrasjonshendelser, slik at man kan spore og revidere viktige endringer i systemet over tid. Dette gir også en oversikt over at de nødvendige sikkerhetstiltakene blir implementert som forventet, uten at det oppstår feil eller uventede problemer.

Project logs ▾

Search all fields

Subnetwork +1 ▾

vpc_flows ▾

All severities ▾

Correlate by ▾

+1 filter

1 logName="projects/konte-eksamen-eksempel1/logs/compute.googleapis.com%2Fvpc_flows"

2 resource.type="gce_subnetwork"

3 resource.labels.subnetwork_name="subnet-north-america"

4 jsonPayload.connection.src_ip="35.191.206.0"

Utklipp som viser et filter som sorterer etter VPC logs på subnett i nord amerika. Er også valgt en ip adresse.

```
{
  insertId: "dwcsd8e47ngf"
  jsonPayload: {
    bytes_sent: "1360"
    connection: {5}
    dest_instance: {5}
    dest_vpc: {4}
    end_time: "2025-02-18T11:14:49.019115053Z"
    network_service: {1}
    packets_sent: "16"
    reporter: "DEST"
    start_time: "2025-02-18T11:14:49.019115053Z"
  }
  logName: "projects/konte-eksamen-eksempel1/logs/compute.googleapis.com%2Fvpc_flows"
  receiveTimestamp: "2025-02-18T11:23:02.560469300Z"
  resource: {2}
  timestamp: "2025-02-18T11:23:02.560469300Z"
}
```

Utklipp som viser en log entry fra VPC logs.

Her vises en nettverksflyt mellom en kilde-IP (35.191.206.0) og en destinasjonsinstans (vm-north-america-1qrv). Det ble sendt 1360 bytes, ved bruk av TCP-port 80 (HTTP) innenfor VPC-en vpc-https-servers. Ved å benytte VPC Flow Logs kan man hente ut viktig informasjon om nettverkstrafikken i miljøet. For eksempel får man oversikt over hvilke IP-adresser som kommuniserer, hvilken mengde data som utveksles, og hvilke protokoller og porter som benyttes. Dette er essensielt for å overvåke normal drift, oppdage uregelmessigheter og identifisere potensielle sikkerhetsproblemer. Ved å analysere slike logger kan man raskt avdekke feilkonfigurasjoner eller mistenkelige aktiviteter, noe som bidrar til en sikrere og mer robust nettverksinfrastruktur.

Integrering av logs i en SIEM

For å sikre en effektiv og skalerbar eksport av logger fra Google Cloud Platform (GCP) til Splunk, kan en arkitektur basert på Cloud Logging, Pub/Sub og Dataflow implementeres. Denne løsningen muliggjør kontinuerlig strømming av loggdata til Splunk Enterprise eller Splunk Cloud Platform. Det er to forskjellige metoder push og pull. Jeg vil gå inn på begge metodene og forklare når det er hensiktsmessig og implementere dem.

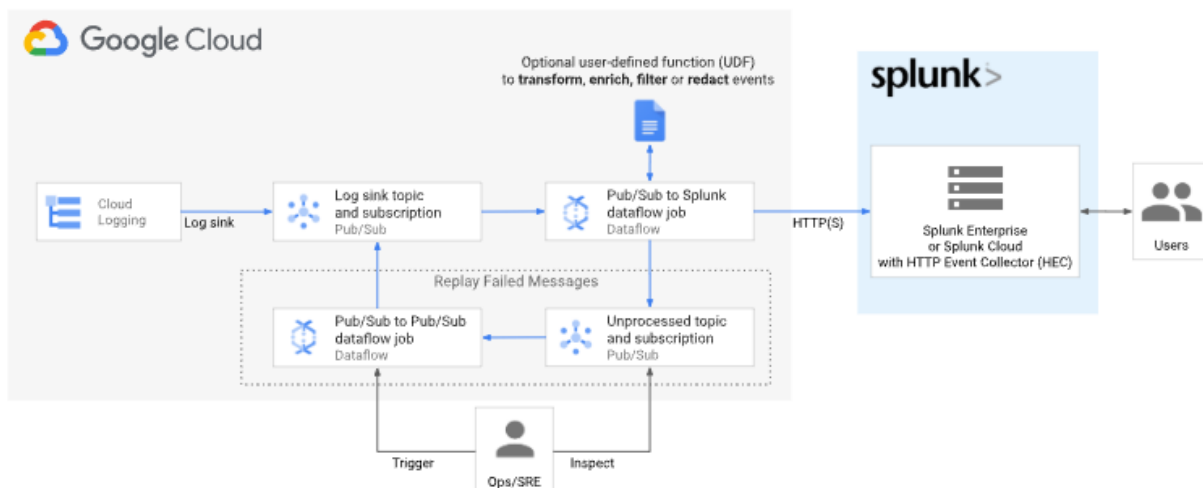
Pull metoden

Eksporten skjer gjennom en loggstrømningspipeline der Cloud Logging samler inn logger fra GCP-ressurser på tvers av organisasjoner, mapper og prosjekter. En aggregert loggsink

videresender loggene til Pub/Sub, som fungerer som en meldingstjeneste for distribuering. Dataflow behandler loggene gjennom to parallelle pipelines, hvor den primære strømmer loggene til Splunk via HTTP Event Collector (HEC), mens den sekundære lagrer ubehandlede meldinger i en backup Pub/Sub-kø for feilhåndtering. Splunk mottar deretter loggene via HEC for analyse og visualisering. Dataflow sikrer automatisk ressursallokering og parallell prosessering av store datamengder, noe som forbedrer ytelsen.

Løsningen er designet med fokus på sikkerhet. En push-basert arkitektur unngår administrasjon av "Service account keys", mens Cloud NAT brukes for utgående trafikk fra Dataflow-"workers", slik at private IP-adresser kan benyttes uten å eksponere interne ressurser. Splunk HEC-token lagres sikkert i Secret Manager for å unngå eksponering av legitimasjon, og en egendefinert Dataflow-tjenestekonto opprettes med minst nødvendige privilegier. Feilhåndtering ivaretas ved bruk av en backup-Pub/Sub-kø for ubehandlede meldinger. Overvåking kan gjøres via Splunk-dashboards eller Cloud Monitoring-metrikker, med varsler konfigurert for rask identifisering og håndtering av problemer.

Diagram av Pull Metoden:



Hentet fra: <https://cloud.google.com/architecture/stream-logs-from-google-cloud-to-splunk>

For å sette opp arkitekturen opprettes en aggregert loggsink i Cloud Logging, som sender logger til en Pub/Sub-topic. Deretter konfigureres Pub/Sub med riktige IAM-policyer for publisering og abonnement. Dataflow-pipelinen settes opp ved bruk av Pub/Sub til Splunk Dataflow-malen, samtidig som en tjenestekonto med nødvendige roller og tilgangsrettigheter opprettes. Til slutt konfigureres Splunk for mottak av logger via HTTP Event Collector (HEC).

Et eksempel på Terraform-kode for å opprette en loggsink og Pub/Sub-topic:

```

splunk.tf > resource "google_pubsub_subscription" "log_subscription"
1  resource "google_logging_project_sink" "log_sink" {
2      name         = "gcp-to-splunk-sink"
3      destination = "pubsub.googleapis.com/projects/YOUR_PROJECT_ID/topics/YOUR_TOPIC"
4      filter       = "resource.type=gce_instance"
5
6      unique_writer_identity = true
7  }
8
9  resource "google_pubsub_topic" "log_topic" {
10     name = "YOUR_TOPIC"
11 }
12
13 resource "google_pubsub_subscription" "log_subscription" {
14     name     = "log-subscription"
15     topic    = google_pubsub_topic.log_topic.name
16 }

```

Det er også mulig å opprette de resterende nødvendige ressursene gjennom Terraform. Dette vil imidlertid kreve opprettelse av flere service accounts, noe som igjen innebærer at Terraform-servicekontoen må tildeles enda flere rettigheter. Siden det ikke anses som en "beste praksis" å gi service kontoer omfattende rettigheter, vil jeg derfor anbefale å opprette disse ressursene manuelt.

Fordeler

- **Administrert tjeneste:** Som en administrert tjeneste håndterer Dataflow de nødvendige ressursene i Google Cloud for databehandlingsoppgaver, for eksempel eksport av logger.
- **Distribuert arbeidsbelastning:** Denne metoden lar deg fordele arbeidsoppgaver på tvers av flere "workers" for parallell behandling, slik at det ikke finnes et enkelt feilpunkt.
- **Sikkerhet:** Siden Google Cloud sender dataene dine til Splunk HEC, unngår man vedlikeholds- og sikkerhetsutfordringene knyttet til opprettelse og administrasjon av tjenestekontonøkler.
- **Autoskalering:** Dataflow-tjenesten skalerer automatisk antall "workers" i henhold til variasjoner i innkommende loggvolum og eventuell backlog.
- **Feiltoleranse:** Hvis det oppstår midlertidige server- eller nettverksproblemer, vil den push-baserte metoden automatisk forsøke å sende dataene til Splunk HEC på nytt. Den støtter også ubehandlede emner (også kjent som dead-letter topics) for eventuelle ikke-leverbare loggmeldinger, slik at datatap unngås.
- **Enkelhet:** Man unngår administrativt arbeid og kostnadene ved å kjøre en eller flere tunge forwardere i Splunk.

Push metoden

En alternativ metode for loggeksport til Splunk er å hente logger fra Google Cloud. I denne pull-baserte metoden bruker man Google Cloud API-er for å hente data gjennom Splunk Add-on for Google Cloud. Denne metoden kan brukes i følgende situasjoner:

- Splunk-installasjonen din tilbyr ikke en Splunk HEC-endepunkt.
- Loggvolumet ditt er lavt.
- Man ønsker å eksportere og analysere Cloud Monitoring-metrikker, Cloud Storage-objekter, Cloud Resource Manager API-metadata, Cloud Billing-data eller logger med lavt volum.
- Man administrerer allerede en eller flere tunge forwardere i Splunk.
- Man bruker den hosted Inputs Data Manager for Splunk Cloud.

Tilleggsfaktorer å vurdere:

- En enkelt arbeider håndterer dataregistreringen, noe som betyr at denne metoden ikke har autoskaleringmuligheter.
- I Splunk kan bruk av en tung forwarder til å hente data skape "one point of failure".
- Den pull-baserte metoden krever at du oppretter og administrerer tjenestekontonøkler for å konfigurere Splunk Add-on for Google Cloud.

Før man bruker Splunk Add-on, må loggoppføringer først rutes til Pub/Sub ved hjelp av en loggsink. Dette kan gjøres på samme måte som det ble gjort i Pull metoden ved bruk av terraform.

Når loggene er eksportert til **Pub/Sub**, må Splunk konfigureres til å hente dem manuelt. Dette innebærer opprettelse av en tjenestekonto, tildeling av nødvendige rettigheter og konfigurering av Splunk Add-on for Google Cloud.

For å gi Splunk tilgang til å hente meldinger fra Pub/Sub, må det opprettes en Google Cloud-tjenestekonto. Dette gjøres via Google Cloud Console under IAM & Admin → Service Accounts. Tjenestekontoen bør gis et beskrivende navn som gjenspeiler formålet, for eksempel splunk-pubsub-reader.

Etter at tjenestekontoen er opprettet, må det genereres en privat nøkkel i JSON-format. Denne nøkkelen vil bli brukt til autentisering når Splunk skal hente loggene fra Pub/Sub. Den må lastes ned og lagres på et sikkert sted, da den gir tilgang til Google Cloud-ressurser.

Tjenestekontoen må gis tilgang til Pub/Sub for å kunne lese meldinger. Dette gjøres ved å tildele følgende roller:

- Pub/Sub Viewer – Gir tillatelse til å vise Pub/Sub-ressurser.
- Pub/Sub Subscriber – Gir tillatelse til å hente meldinger fra en Pub/Sub-subscription.

Disse rollene kan tildeles via Google Cloud Console under IAM & Admin, eller ved hjelp av Google Cloud CLI.

Når tjenestekontoen er konfigurert med nødvendige tillatelser, må Splunk settes opp til å bruke den:

- Åpne Splunk og naviger til Splunk Add-on for Google Cloud.
- Følg instruksjonene for å konfigurere en ny Pub/Sub-input.
- Last opp tjenestekontonøkkelen (JSON-fil) og spesifiser hvilken Pub/Sub-subscription som skal brukes.

Etter at Splunk er konfigurert til å hente loggene, kan det utføres en verifikasjon ved å søke etter loggdata i Splunk. Dette sikrer at oppsettet fungerer som forventet, og at loggene blir korrekt importert fra Google Cloud.

I vårt tilfelle i NovaTech ville nok Pull-metoden vært det beste valget, ettersom den innebærer lavere kostnader og et enklere oppsett sammenlignet med Push-metoden. Dette gjør den ideell for en mindre bedrift som ønsker en kostnadseffektiv løsning for logghåndtering. Eventuelt kan push metoden implementeres dersom det blir nødvendig.

Hvordan sørge for en sikker og optilamisert SIEM arkitektur.

For å sikre at arkitekturen oppfyller kravene til sikkerhet, personvern, samsvar, operasjonell effektivitet, pålitelighet, feiltoleranse, ytelse og kostnadsoptimalisering, er det viktig å følge en rekke sikkerhetsprinsipper og designhensyn.

For å minimere eksponering mot det offentlige internettet bør Dataflow-VM-er konfigureres til å bruke private IP-adresser. Dette sikrer at all kommunikasjon mellom Dataflow, Pub/Sub og Splunk HEC skjer over et beskyttet nettverk, uten at sensitive data risikerer å bli eksponert eksternt. Videre bør Private Google Access aktiveres, slik at interne VM-er i et privat VPC-nettverk kan kommunisere med Google Cloud-tjenester uten behov for offentlig IP-adresse. Dette bidrar til å redusere angrepsflaten og øker sikkerheten.

For ytterligere å beskytte Splunk HEC mot uautorisert tilgang, bør innkommende trafikk begrenses til kjente og godkjente IP-adresser. En effektiv måte å kontrollere dette på er å bruke Cloud NAT for å regulere hvilke IP-adresser som får tilgang til Splunk HEC. Ved å implementere slike restriksjoner sikrer man at kun autoriserte systemer kan sende data, noe som reduserer risikoen for potensielle angrep.

Autentiseringsmekanismer må også håndteres på en sikker måte. Splunk HEC-token, som gir tilgang til å sende data til Splunk, bør ikke lagres direkte i konfigurasjonsfiler, men heller sikres gjennom Google Cloud Secret Manager. Dette forhindrer utilsiktet eksponering og sørger for at kun tjenester med nødvendige rettigheter har tilgang til tokenet.

Når det gjelder tjenestekontoer, bør Dataflow ikke benytte standard Compute Engine-tjenestekontoen, men i stedet ha en spesialtilpasset tjenestekonto med minst mulige nødvendige rettigheter. Dette følger prinsippet om minst privilegium og bidrar til å begrense potensielle skader dersom kontoen skulle bli kompromittert.

Dersom en privat sertifikatautoritet (CA) benyttes, bør SSL-validering konfigureres for å sikre at kommunikasjon mellom tjenester skjer over krypterte og verifiserte forbindelser. Dette er

spesielt viktig i miljøer der strenge samsvarskrav gjelder, for eksempel i finans- eller helsesektoren.

Ved å følge disse prinsippene sikrer man at infrastrukturen er robust, sikker og i samsvar med gjeldende beste praksis for både skalerbarhet og kostnadseffektivitet. Samtidig reduseres risikoen for sikkerhetsbrudd, og man opprettholder en stabil og effektiv logghåndteringsprosess.

Kilde brukt: <https://cloud.google.com/architecture/stream-logs-from-google-cloud-to-splunk>

Jeg opprettet også en egen Logging Sink og et datasett for å sende logger til BigQuery. Siden jeg ikke ønsker at Terraform-servicekontoen skal kunne opprette eller endre IAM-brukere, ga jeg den kun midlertidige rettigheter til å opprette de nødvendige ressursene. Etter at ressursene var opprettet, fjernet jeg disse rettighetene for å begrense tilgangen til IAM-administrasjon.

```
big_query.tf > resource "google_project_iam_member" "bq_sink_role"
1  resource "google_bigquery_dataset" "log_dataset" {
2    dataset_id = "logs_dataset"
3    project    = "konte-eksamen-eksempel1"
4    location   = "EU"
5
6    labels = {
7      environment = "production"
8    }
9  }
10 resource "google_logging_project_sink" "log_sink" {
11   name                = "bigquery-logging-sink"
12   destination         = "bigquery.googleapis.com/projects/konte-eksamen-eksempel1/datasets/logs_dataset"
13   filter              = "logName:*"
14   unique_writer_identity = true
15 }
16
17 resource "google_project_iam_member" "bq_sink_role" {
18   project = "konte-eksamen-eksempel1"
19   role    = "roles/bigquery.dataEditor"
20   member  = google_logging_project_sink.log_sink.writer_identity
21 }
```

Utklipp som viser big_query.tf.

Fordeler ved å bruke et SIEM verktøy:

Et SIEM-system gir virksomheter bedre oversikt og kontroll over sikkerhetshendelser ved å samle og analysere logger fra ulike systemer i sanntid. Ved å implementere et SIEM oppnår man en mer effektiv overvåkning av IT-miljøet, noe som gjør det enklere å oppdage og håndtere sikkerhetstrusler før de eskalerer til alvorlige hendelser.

En av de største fordelene med et SIEM er evnen til å oppdage trusler i sanntid. Systemet kan samle inn logger fra nettverksenheter, servere, applikasjoner og skytjenester, analysere dem for avvik, og varsle sikkerhetsteamet når mistenkelig aktivitet oppdages. Dette kan for eksempel være forsøk på uautorisert innlogging, uvanlige datatilganger eller nettverkstrafikk som avviker fra normalen.

Automatisert hendelsesrespons er en annen viktig fordel. Et SIEM kan settes opp til å utløse forhåndsdefinerte tiltak når en potensiell sikkerhetshendelse oppdages. For eksempel kan systemet automatisk blokkere en IP-adresse som har flere mislykkede innloggingsforsøk, eller varsle IT-sikkerhetsteamet dersom en bruker forsøker å tilgå sensitive filer uten riktig tilgang. Dette bidrar til å redusere responstiden og begrense skadeomfanget ved angrep.

SIEM spiller også en viktig rolle i samsvar og revisjon. Mange virksomheter er underlagt regulatoriske krav som GDPR, ISO 27001, PCI DSS og NIST, og et SIEM hjelper med å dokumentere og rapportere sikkerhetshendelser på en måte som oppfyller disse kravene. Dette forenkler revisjonsprosesser og sikrer at bedriften har kontroll over sikkerhetsloggene sine.

I tilfeller der en sikkerhetshendelse allerede har skjedd, gir SIEM verdifull innsikt i etterforskningen. Gjennom forensisk analyse kan man spore hendelsesforløpet, se hvilke systemer som ble berørt, og identifisere hvordan angrepet skjedde. Dette gir viktig læring og bidrar til å forbedre sikkerhetstiltakene for fremtiden.

I tillegg gir et SIEM økt oversikt over IT-infrastrukturen. Ved å sentralisere loggdata og presentere dem i et dashboard, blir det enklere å oppdage systemfeil, ytelsesproblemer og feilkonfigurasjoner som kan påvirke driften. Dette gjør ikke bare sikkerheten bedre, men forbedrer også systemtilgjengeligheten og den generelle driftsstabiliteten.

En annen fordel er at SIEM-systemer reduserer mengden falske alarmer. Ved å bruke avansert analyse og maskinlæring kan systemet filtrere bort uviktige hendelser og fremheve de reelle sikkerhetstruslene, noe som sparer tid og ressurser for sikkerhetsteamet.

Kilder brukt: <https://www.blumira.com/glossary/what-is-siem>

Oppgave A. 3

Dokumentasjon fra GCP

vpc-http-servers

<	OVERVIEW	SUBNETS	STATIC INTERNAL IP ADDRESSES	FIREWALLS	FIREWALL ENDPOINTS	ROUTES	VPC NETWORK PEERING	PRIVATE	>
---	----------	---------	------------------------------	-----------	--------------------	--------	---------------------	---------	---

Subnets [+ ADD SUBNET](#) [MANAGE FLOW LOGS](#)

<input type="checkbox"/>	Name ↑	Region	Stack Type	Primary IPv4 range	Secondary IPv4 ranges	IPv6 ranges	Reserved internal ranges	Gateway	
<input type="checkbox"/>	subnet-asia	asia-east1	IPv4 (single-stack)	10.30.0.0/24			None	10.30.0.1	
<input type="checkbox"/>	subnet-australia	australia-southeast1	IPv4 (single-stack)	10.40.0.0/24			None	10.40.0.1	
<input type="checkbox"/>	subnet-europe	eu-west-1	IPv4 (single-stack)	10.10.0.0/24			None	10.10.0.1	
<input type="checkbox"/>	subnet-north-america	us-central1	IPv4 (single-stack)	10.20.0.0/24			None	10.20.0.1	

Utklipp som viser at alle subnettene er opprettet.

←

Firewall rule details

EDIT

DELETE

allow-http

Logs

Off

view in Logs Explorer

Network

vpc-https-servers

Priority

1000

Direction

Ingress

Action on match

Allow

Tags

—

Targets

Target tags

http

Source filters

IP ranges

130.211.0.0/22

35.191.0.0/16

Protocols and ports

tcp:80

Utklipp som viser at brannmur-regelen er opprettet.

VPC Flow Logs configurations for subnets

VIEW IN FLOW ANALYZER

Filter

Enter property name or value

	Subnet name	Region	Configuration details	Action
	subnet-asia	asia-east1	aggregation: 10 min sample rate: 50% metadata	VIEW IN FLOW ANALYZER
	subnet-australia	australia-southeast1	aggregation: 10 min sample rate: 50% metadata	VIEW IN FLOW ANALYZER
	subnet-europe	europe-west1	aggregation: 10 min sample rate: 50% metadata	VIEW IN FLOW ANALYZER
	subnet-north-america	us-central1	aggregation: 10 min sample rate: 50% metadata	VIEW IN FLOW ANALYZER

Utklipp som viser at VPC Flow logs konfigurasjonene.

Instance groups

CREATE INSTANCE GROUP

REFRESH

DELETE

LE

Instance groups are collections of VM instances that use load balancing and automated services, like autoscaling and autohealing. [Learn more](#)

Filter

Enter property name or value

	Status	Name	Instances	Template	Group type	Creation time	Recommendation	Autoscaling	Zone	In Use By
	✓	igm-asia	1	vm-template-asia	Managed	Feb 17, 2025, 3:12:53 PM UTC+01:00	No configuration	No configuration	asia-east1-a	http-backend-service
	✓	igm-australia	1	vm-template-australia	Managed	Feb 17, 2025, 3:12:51 PM UTC+01:00	No configuration	No configuration	australia-southeast1-a	http-backend-service
	✓	igm-europe	1	vm-template-europe	Managed	Feb 17, 2025, 3:12:53 PM UTC+01:00	On: Target CPU utilization 60%	On: Target CPU utilization 60%	europe-west1-b	http-backend-service
	✓	igm-north-america	1	vm-template-north-america	Managed	Feb 17, 2025, 3:12:49 PM UTC+01:00	On: Target CPU utilization 60%	On: Target CPU utilization 60%	us-central1-a	http-backend-service

Utklipp som viser at instanse gruppene er opprettet.

Instance templates

[+ CREATE INSTANCE TEMPLATE](#) [REFRESH](#) [CREATE VM](#) [CREATE INSTANCE GROUP](#) [COPY](#) [DELETE](#)

Instance templates are saved VM configurations used to create identical VMs, either individually or as part of managed instance groups. [Learn more](#)

Filter	Filter instance templates								
<input type="checkbox"/>	Name ↑	Machine type	Image	Disk type	Location	Placement policy	In use by	Creation time	Actions
<input type="checkbox"/>	vm-template-asia	e2-micro	debian-12	Standard persistent disk	global	No policy	igm-asia	Feb 17, 2025, 3:12:36 PM UTC+01:00	⋮
<input type="checkbox"/>	vm-template-australia	e2-micro	debian-12	Standard persistent disk	global	No policy	igm-australia	Feb 17, 2025, 3:12:39 PM UTC+01:00	⋮
<input type="checkbox"/>	vm-template-europe	e2-micro	debian-12	Standard persistent disk	global	No policy	igm-europe	Feb 17, 2025, 3:12:34 PM UTC+01:00	⋮
<input type="checkbox"/>	vm-template-north-america	e2-micro	debian-12	Standard persistent disk	global	No policy	igm-north- igm-north-	Feb 17, 2025, 3:12:32 PM UTC+01:00	⋮

Utklipp som viser at instance templatene er opprettet.

Health checks

[+ CREATE HEALTH CHECK](#) [REFRESH](#) [DELETE](#)

Health checks determine if applications on your VMs respond to requests. They're used for load balancing and with autohealing in managed instance groups. [Learn more](#)

Filter	Enter property name or value							
<input type="checkbox"/>	Name ↑	Scope	Region	Host	Path	Protocol	Port	In use by
<input type="checkbox"/>	http-health-check ⓘ (legacy)	Global			/	HTTP	80	http-backend-service

Utklipp som viser at helse sjekken er opprettet.

Cloud Armor policies

[+ CREATE POLICY](#) [DELETE POLICY](#)

Cloud Armor advanced network DDoS protection is now generally available to protect applications and services using Network Load Balancer, Protocol Forwarding, or VMs with

Security policies let you control access to your Google Cloud resources at your network's edge, including internal Load Balancers.

You can use security policies to protect workloads on external Cloud Load Balancing deployments, Protocol forwarding deployments, or Instances with public IP addresses. [Learn more](#)

Filter	Enter property name or value						
<input type="checkbox"/>	Name ↑	Type	Scope	Rules	Targets	Description	
<input type="checkbox"/>	web-security-policy	Backend security policy	global	6	1	Security policy protecting against SQL injection, XSS, RCE, LFI, and brute-force attacks	⌵ ⋮

Utklipp som viser at web-security-policy er opprettet.

←

Load balancer details

EDIT

DELETE

→ VIEW IN NETWORK TOPOLOGY

global-url-map

Classic Application Load Balancer

Faster web performance and improved web protection with Cloud CDN and Cloud Armor. [Learn more](#)

DETAILS

MONITORING

CACHING

MIGRATION

Frontend

Protocol ↑	IP:Port	Certificate	Certificate Map	SSL Policy	Network Tier ?
HTTPS	34.54.39.96:443	managed-ssl-cert		GCP default	Premium

Host and path rules

Hosts ↑	Paths	Backend
All unmatched (default)	All unmatched (default)	http-backend-service

Backend

Backend services

1. http-backend-service

Endpoint protocol	HTTP
Named port	http
Timeout	30 seconds
Health check	http-health-check
Cloud CDN	Disabled
Logging	Disabled
Backend security policy	web-security-policy

SHOW ADVANCED

Utklipp som viser at last balanseren er opprettet.

VM instances

Filter

Enter property name or value

<input type="checkbox"/>	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	✓	vm-asia-ss9j	asia-east1-a		igm-asia	10.30.0.2 (nic0)	34.80.93.230 (nic0)	SSH ▾ ⋮
<input type="checkbox"/>	✓	vm-australia-jtk1	australia-southeast1-a		igm-australia	10.40.0.2 (nic0)	34.40.141.51 (nic0)	SSH ▾ ⋮
<input type="checkbox"/>	✓	vm-europe-0xhm	europa-west1-b		igm-europe	10.10.0.2 (nic0)	35.189.232.207 (nic0)	SSH ▾ ⋮
<input type="checkbox"/>	✓	vm-north-america-1qrv	us-central1-a		igm-north-america	10.20.0.2 (nic0)	104.197.173.2 (nic0)	SSH ▾ ⋮

Utklipp som viser at Vm-Instansene har blitt opprettet.



Certificate details



DELETE

managed-ssl-cert

Certificate type	MANAGED
Status	ACTIVE
Status description	The Google-managed SSL certificate is obtained from the Certificate Authority

Domain	Status
wrathbyte.com	ACTIVE
www.wrathbyte.com	ACTIVE

DNS Hostnames	wrathbyte.com, www.wrathbyte.com
Expires	May 18, 2025, 4:36:06 PM
Serial number	3A:1A:1E:C4:D9:C3:06:F5:12:A5:99:50:8C:E5:8E:8E
Certificate Issuer	WR3

Utklipp som viser ssl sertifikatet.



Backend service details



DELETE

In order to edit backend service use load balancer mutations, gcloud or REST API.

http-backend-service

General properties

Description	Backend service for HTTP traffic
Load balancer type	Classic Application Load Balancer (EXTERNAL)
Endpoint protocol	HTTP
In use by	global-url-map
Timeout	30 seconds
IP address selection policy	Only IPv4
Health check	http-health-check VIEW HEALTH CHECK DETAILS
Backend security policy	web-security-policy
Session affinity	None
Cloud CDN	Disabled
Connection draining timeout	300 seconds
Custom request headers	Currently there are no custom request headers configured
Custom response headers	Currently there are no custom response headers configured

Backends

Name	Type	IP stack type	Scope	Healthy	Autoscaling	Balancing mode	Selected ports	Capacity	Preference level
lgm-asia	Instance group	IPv4	asia-east1-a	1 of 1	No configuration	N/A	80	100%	None
lgm-australia	Instance group	IPv4	australia-southeast1-a	1 of 1	No configuration	N/A	80	100%	None
lgm-europe	Instance group	IPv4	europe-west1-b	1 of 1	No configuration	N/A	80	100%	None
lgm-north-america	Instance group	IPv4	us-central1-a	1 of 1	No configuration	N/A	80	100%	None

Utklipp som viser at Backend servicen er opprettet.

Snapshot schedules

CREATE SNAPSHOT

CREATE SNAPSHOT SCHEDULE

REFRESH

DELETE

LEARN

Use snapshot schedules to regularly and automatically back up your zonal and regional persistent disks. [Learn more](#)

COMPARE SNAPSHOT TYPES

Snapshots

Archive snapshots

Instant snapshots

Snapshot Schedules

Filter

Enter property name or value

	Status	Name	Region	Storage Location	Schedule frequency (UTC)	Autodelete snapshots after	In use by	Creation time	Application Consistency
<input type="checkbox"/>	✓	daily-backup-policy-asia	asia-east1	asia (Asia Pacific)	Every day, starts between 3:00 AM and 4:00 AM	7 days	vm-asia-ss9j	Feb 17, 2025, 3:11:48 PM UTC+01:00	Disabled
<input type="checkbox"/>	✓	daily-backup-policy-au	australia-southeast1	asia (Asia Pacific)	Every day, starts between 3:00 AM and 4:00 AM	7 days	vm-	Feb 17, 2025, 3:11:49 PM UTC+01:00	Disabled
<input type="checkbox"/>	✓	daily-backup-policy-eu	eu-west1	eu (European Union)	Every day, starts between 3:00 AM and 4:00 AM	7 days	vm-europe-	Feb 17, 2025, 3:11:48 PM UTC+01:00	Disabled
<input type="checkbox"/>	✓	daily-backup-policy-na	us-central1	us (United States)	Every day, starts between 3:00 AM and 4:00 AM	7 days	vm-north-	Feb 17, 2025, 3:11:48 PM UTC+01:00	Disabled

Utklipp som viser at snapshot/backup planene er opprettet.



Utklipp som viser at ved bruk av vpn vil last balansen bruke instansene som er i USA.



Utklipp uten VPN.

Oppgave B

Oppgave 1:

Skytjenester fungerer ved at eksterne servere håndterer lagring, prosessering og administrasjon av data og applikasjoner via Internett. I stedet for å være avhengig av lokal maskinvare, kan brukere og virksomheter få tilgang til disse tjenestene gjennom skyleverandører (CloudFlare, u.å.). Dette gjør det mulig å benytte avansert IT-infrastruktur uten å måtte eie eller vedlikeholde den selv.

Kjernen i skytjenester er virtualisering, en teknologi som tillater flere virtuelle maskiner å kjøre på en enkelt fysisk server. Skyleverandører drifter store datasentre med mange sammenkoblede servere, noe som sikrer skalerbarhet, redundans og høy tilgjengelighet. Brukere får tilgang til skyressurser gjennom nettlesere, API-er eller spesifikke applikasjoner, avhengig av tjenesten. Skytjenester kan typisk kategoriseres i tre hovedmodeller: Infrastructure-as-a-Service (IaaS), hvor virksomheter leier infrastruktur som virtuelle maskiner og lagringsplass; Platform-as-a-Service (PaaS), som gir utviklere en plattform for å bygge og distribuere applikasjoner; og Software-as-a-Service (SaaS), hvor programvare leveres over Internett uten behov for lokal installasjon (Cloudflare, u.å.).

Ved å bruke skyløsninger oppnår virksomheter betydelige besparelser på ressurser, vedlikehold og eiendomskostnader. I tillegg muliggjør skyteknologien optimalisering av arbeidsbelastninger, slik at driftskostnadene reduseres og ressursbruken effektiviseres (Microsoft, u.å.).

Uavhengig av hvilken skytjenestemodell du velger, betaler du kun for de ressursene du faktisk bruker. Dette bidrar til å unngå overinvestering i infrastruktur og forhindrer overdimensjonering av datasentre. Samtidig frigjør det verdifull tid for IT-teamene, slik at de kan fokusere på mer strategiske oppgaver (Google, u.å.).

Skytjenester er ikke bare en teknologi i seg selv, men også en katalysator for videre innovasjon. Ved å utnytte skyteknologi kan virksomheter utvikle mer avanserte og forbedrede tjenester for kundene, effektivisere arbeidsprosesser og utforske nye forretningsmodeller med lavere kostnader og raskere implementering (Basalisco et al., 2023, s. 33). I tillegg gir kostnadsbesparelsene og de økte inntektene som følger av skybruk mulighet for reinvestering i forretningsutvikling og andre innovative initiativer, noe som ytterligere styrker vekst og konkurransekraft (Basalisco et al., 2023, s. 33).

Konklusjon:

Skytjenester har revolusjonert måten virksomheter håndterer IT-infrastruktur på ved å tilby fleksible, kostnadseffektive og skalerbare løsninger. Ved å fjerne behovet for lokal maskinvare og vedlikehold, gir skytjenester bedrifter tilgang til avansert teknologi uten store

investeringer. Virtualiseringsteknologi muliggjør effektiv ressursbruk, mens ulike tjenestemodeller som IaaS, PaaS og SaaS gir organisasjoner muligheten til å tilpasse skybruken etter deres spesifikke behov.

I tillegg til økonomiske besparelser gir skytjenester økt operasjonell effektivitet og mulighet for innovasjon. Bedrifter kan optimalisere arbeidsprosesser, redusere driftskostnader og frigjøre IT-ressurser til mer strategiske oppgaver. Videre bidrar skytjenester til utvikling av nye forretningsmodeller og tjenester, noe som styrker konkurranseevnen og fremmer bærekraftig vekst.

Samlet sett representerer skytjenester ikke bare en teknologisk utvikling, men også en strategisk fordel for virksomheter som ønsker å redusere kostnader, øke produktiviteten og drive innovasjon i en stadig mer digitalisert økonomi.

Oppgave 2:

Skytjenester gir virksomheter en rekke tekniske fordeler som forbedrer effektivitet, sikkerhet og tilgjengelighet. En av de viktigste fordelene er raskere utvikling og utrulling av nye tjenester. Med skybaserte løsninger kan utviklere opprette og avslutte instanser i løpet av sekunder, noe som muliggjør raske distribusjoner og kontinuerlig integrasjon. Dette reduserer tiden det tar å lansere nye produkter og tjenester, samtidig som det gjør det enklere å teste nye ideer og utvikle applikasjoner uten begrensninger knyttet til maskinvare eller langsomme innkjøpsprosesser (Google, u.å.).

En annen teknisk fordel er skalerbarhet og fleksibilitet, som gir virksomheter mulighet til å justere ressursbruk etter behov. Med skyteknologi kan selskaper dynamisk øke eller redusere ressursallokeringen uten å måtte investere i dyr fysisk infrastruktur. I stedet for å bygge kapasitet for maksimal belastning, kan bedrifter automatisk tilpasse ressursbruken til faktisk etterspørsel. Dette sikrer optimal ytelse, samtidig som kostnadene holdes lave når mindre kapasitet er nødvendig (Google, u.å.).

Forbedret samarbeid og tilgjengelighet er også en sentral fordel med skytjenester. Ved å lagre data i skyen i stedet for på lokale enheter, kan brukere få tilgang til informasjon fra hvor som helst og på hvilken som helst enhet med internettilkobling. Dette eliminerer behovet for å være knyttet til en bestemt lokasjon eller maskin, noe som øker fleksibiliteten og forbedrer samarbeidet mellom ansatte, uavhengig av geografisk plassering (Google, u.å.).

Når det gjelder sikkerhet, kan skytjenester faktisk styrke en virksomhets digitale forsvar. Ifølge Nasjonal sikkerhetsmyndighet (NSM) bidrar bruk av skytjenester til å redusere risiko, særlig for mindre virksomheter som ikke har kompetanse eller ressurser til å håndtere sikkerhetsarbeid på egen hånd. Store skyleverandører tilbyr omfattende sikkerhetsfunksjoner, automatisert vedlikehold og sentralisert administrasjon, noe som reduserer risikoen for sikkerhetsbrudd. De ansetter også eksperter på cybersikkerhet og implementerer avanserte løsninger for å beskytte data mot trusler. Likevel understreker NSM at flertallet av uønskede hendelser knyttet til skytjenester skyldes feilkonfigurasjoner eller feil bruk snarere enn svakheter i selve tjenesten. Dette viser at skytjenester må brukes riktig, og at virksomheter må ha nødvendig kompetanse for å sikre riktig konfigurasjon og drift (NSM, 2023).

Til slutt er datatapssikring og katastrofegjenoppretting en betydelig teknisk fordel ved skytjenester. Ved å lagre data i skyen i stedet for lokalt, reduseres risikoen for tap av kritisk informasjon ved hendelser som maskinvarefeil, ondsinnede angrep eller menneskelige feil. Skytjenesteleverandører tilbyr innebygde sikkerhetskopierings- og gjenopprettingsløsninger, noe som sikrer at data kan gjenopprettes raskt dersom en uventet hendelse oppstår (Google, u.å.).

Til tross for de mange fordelene har skytjenester enkelte begrensninger. En av de største utfordringene er avhengighet av internettforbindelse, som kan føre til midlertidig tap av tilgang ved nettverksproblemer. Selv de største skyleverandørene kan oppleve nedetid og redusert ytelse på grunn av uforutsette tekniske feil eller eksterne faktorer. Videre kan virksomheter oppleve risiko for leverandørbinding, noe som kan gjøre det utfordrende å bytte plattform uten høye migrasjonskostnader (Google, u.å.).

Sikkerhet og personvern er også et viktig aspekt, da virksomheter må forstå hvilke sikkerhetstiltak som er deres ansvar og hvilke som håndteres av skyleverandøren. I tillegg kan skjulte kostnader og uventede utgifter oppstå hvis bruksmønstre ikke er godt kartlagt på forhånd (Google, u.å.).

Konklusjon:

Skytjenester gir virksomheter en rekke tekniske fordeler som bidrar til raskere innovasjon, bedre ytelse og økt sikkerhet. Skalerbarhet og fleksibilitet gir mulighet for dynamisk ressursstyring, mens sikkerhetsmekanismer hos skyleverandører kan beskytte mot cybertrusler. Samtidig er det avgjørende at virksomheter har riktig kompetanse for å konfigurere og bruke skytjenestene på en sikker måte.

Til tross for enkelte begrensninger, som avhengighet av internett og risiko for leverandørbinding, veier fordelene tungt. Med riktig planlegging og forståelse av skytjenestenes kapasitet og kostnadsstruktur kan virksomheter utnytte skyens potensial til å forbedre drift, sikkerhet og innovasjon på en bærekraftig måte.

Oppgave 3:

Forskjellen mellom de ulike driftsmodellene innen skytjenester – IaaS, PaaS og SaaS – ligger i graden av kontroll og ansvar som virksomheten har over IT-infrastrukturen.

Infrastructure as a Service (IaaS) gir tilgang til grunnleggende IT-ressurser som servere, lagring, nettverk og virtualisering, hvor skyleverandøren håndterer den fysiske infrastrukturen. Samtidig er virksomheten selv ansvarlig for installasjon og vedlikehold av operativsystemer, applikasjoner og sikkerhet. Denne modellen gir stor fleksibilitet og kontroll, men krever også teknisk kompetanse for drift og administrasjon (Google, u.å.).

Platform as a Service (PaaS) går et skritt videre ved å tilby en ferdig utviklingsplattform hvor skyleverandøren ikke bare administrerer infrastrukturen, men også operativsystemer, databaser og utviklingsverktøy. Dette gjør det mulig for virksomheter å fokusere utelukkende på utvikling og distribusjon av applikasjoner uten å bekymre seg for underliggende systemer. PaaS egner seg godt for programvareutvikling hvor rask utrulling og effektiv testing er avgjørende (Google, u.å.).

Software as a Service (SaaS) representerer det mest brukervennlige alternativet, der skyleverandøren tar hånd om hele løsningen – fra infrastruktur og programvare til vedlikehold, oppdateringer og sikkerhet. Virksomheten trenger kun å bruke tjenesten via en nettleser eller en applikasjon, uten å måtte installere eller drifte noe selv. Dette gjør SaaS til et ideelt valg for bedrifter som ønsker en enkel, kostnadseffektiv løsning for forretningsbehov som e-post, CRM, dokumenthåndtering eller økonomisystemer (Google, u.å.).

Hvilken modell som er mest hensiktsmessig, avhenger av hvor mye kontroll virksomheten ønsker over IT-infrastrukturen, samt hvor mye teknisk ansvar de er villige til å ta. IaaS gir maksimal fleksibilitet og kontroll, men krever større teknisk innsats. PaaS reduserer administrasjonsbehovet ved å tilby et komplett utviklingsmiljø, mens SaaS er det mest brukervennlige alternativet, hvor alt er ferdig satt opp av leverandøren. Uansett valg gir skytjenester muligheten til å utnytte moderne teknologi på en mer kostnadseffektiv og skalerbar måte.

Oppgave 4:

Sikring av skytjenester innebærer en kombinasjon av teknologier, retningslinjer og sikkerhetsprosedyrer for å beskytte data, applikasjoner og infrastruktur mot trusler. Dette inkluderer tiltak som identitets- og tilgangsstyring, datakryptering, nettverkssikkerhet og samsvar med regelverk.

En av de viktigste prinsippene i skybasert sikkerhet er delt ansvar, hvor skyleverandøren er ansvarlig for den underliggende infrastrukturen, mens kunden har ansvar for sikkerhetskonsfigurasjoner, tilgangskontroll og beskyttelse av data. Dette varierer basert på skytjenestemodellen. I en IaaS-modell må kunden administrere operativsystemer og nettverkssikkerhet, mens i en SaaS-modell er det hovedsakelig skyleverandøren som sikrer tjenesten (Google, u.å.).

For å sikre skytjenester brukes avanserte sikkerhetsverktøy som identitets- og tilgangsstyring (IAM), som kontrollerer hvem som har tilgang til ressurser, og datatapssikring (DLP) for å oppdage og forhindre uautorisert deling av sensitiv informasjon. SIEM-løsninger (Security Information and Event Management) brukes til å analysere loggdata og oppdage mistenkelige aktiviteter i sanntid. I tillegg benyttes kryptering for å beskytte data både under lagring og i transitt (Google, u.å.).

Til tross for disse tiltakene har flere tilfeller vist hva som skjer når skytjenester ikke er riktig sikret. Et kjent eksempel er Capital One-datasikkerhetsbruddet i 2019, hvor en feilkonfigurert webapplikasjonsbrannmur muliggjorde eksfiltrering av sensitiv kredittkortsøknadsdata. Over 106 millioner kunder i USA og Canada ble berørt, noe som gjorde dette til en av de største datainnbruddene i forrige tiår, på linje med Equifax, Target og Marriott (Khan et al., 2023).

Det som gjør Capital One-innbruddet spesielt interessant, er at tekniske detaljer om angrepsmetoden har blitt offentliggjort, noe som gir et sjeldent innblikk i hvordan slike angrep utføres. I motsetning til mange andre store datainnbrudd var dette ikke et tilfelle av en ukjent sårbarhet, men heller en utnyttelse av feilkonfigurasjon i en skybasert tjeneste. Dette understreker viktigheten av riktig sikkerhetskonsfigurasjon og kontinuerlig overvåking av skytjenester for å minimere risikoen for angrep (Khan et al., 2023).

Dette eksempelet viser at skytjenester, selv hos de største aktørene, kan være sårbare dersom sikkerhetsansvaret ikke tas på alvor. Feilkonfigurasjoner, utilstrekkelig tilgangskontroll og manglende oversikt over sikkerhetsinnstillinger er ofte årsaken til alvorlige datainnbrudd. For å unngå dette må virksomheter ha en grundig **sikkerhetsstrategi**, sikre riktig konfigurasjon av skytjenester, og kontinuerlig overvåke sine skyressurser for mistenkelige aktiviteter.

Oppgave 5:

En skybasert applikasjon er designet fra grunnen av for å utnytte skalerbarheten og den distribuerte naturen til skyen. I motsetning til tradisjonelle monolittiske applikasjoner, som er bygget som en enhet med tett integrerte komponenter, er skybaserte applikasjoner ofte modulære og basert på mikrotjenestearkitektur, containere og automatiserte utviklingsprosesser (Google, u.å.).

I tradisjonelle IT-miljøer utvikles applikasjoner med faste ressurser og må testes og distribueres som en helhet, noe som gjør oppdateringer og skalerbarhet mer utfordrende. Skybaserte applikasjoner, derimot, brytes ned i løst koblede tjenester som kan administreres uavhengig av hverandre. Dette gjør det enklere å oppdatere, distribuere og skalere spesifikke funksjoner uten å påvirke resten av applikasjonen (Google, u.å.). Et eksempel er en netthandelsplattform der handlekurven, betalingssystemet og lagerstyringen hver fungerer som separate tjenester som kan oppdateres individuelt.

For å oppnå disse fordelene er skybaserte applikasjoner bygget ved hjelp av flere teknologier og utviklingsmetoder. DevOps og kontinuerlig integrasjon/kontinuerlig levering (CI/CD) gjør det mulig å raskt bygge, teste og distribuere oppdateringer uten nedetid. Containere og Kubernetes brukes for å sikre at applikasjonene kan kjøre i ethvert miljø uten behov for manuell konfigurasjon, mens deklarativer API-er muliggjør sømløs kommunikasjon mellom tjenestene (Google, u.å.).

Ifølge Amazon er skybaserte applikasjoner ikke bare teknologisk avanserte, men også økonomisk og operasjonelt fordelaktige. AWS påpeker at disse applikasjonene drar nytte av skydrevne tjenester, som serverløs databehandling (AWS Lambda), administrerte databaser (Amazon RDS) og AI/ML-verktøy for automatisert analyse. Dette gir organisasjoner muligheten til å redusere driftskostnader, forbedre pålitelighet og raskere bringe nye produkter til markedet. AWS understreker også viktigheten av infrastruktur som kode (IaC), som lar utviklere administrere og provisionere skyressurser programmatisk, noe som øker effektiviteten og sikrer konsistens i applikasjonsutviklingen (Amazon, u.å.).

Skybaserte applikasjoner gir betydelige fordeler som raskere innovasjon, høy tilgjengelighet, bedre sikkerhet og lavere driftskostnader. De kan også enkelt flyttes mellom ulike miljøer, noe som gjør dem svært portable. Samtidig krever denne arkitekturen en annen tilnærming til utvikling og drift, og virksomheter som ønsker å dra full nytte av skybaserte applikasjoner, må være forberedt på kulturelle og teknologiske endringer for å lykkes i overgangen (Google, u.å.; Amazon, u.å.).

Oppgave 6:

Skymigrering refererer til prosessen med å flytte applikasjoner, data og IT-arbeidsbelastninger fra en lokal infrastruktur til skyen. For å sikre en vellykket overgang har

AWS definert syv forskjellige migreringsstrategier, kjent som "The 7 Rs". Disse strategiene dekker ulike tilnærminger for hvordan applikasjoner og systemer kan overføres til skyen, basert på organisasjonens behov, tekniske krav og forretningsmål (AWS, u.å.).

En av de viktigste grunnene til at virksomheter velger å migrere til skyen er skalerbarhet, fleksibilitet og kostnadsbesparelser. Skybaserte løsninger gir muligheten til å tilpasse ressursbruken etter behov, noe som eliminerer behovet for store investeringer i infrastruktur. I tillegg gir skyen tilgang til de nyeste teknologiske innovasjonene, sikkerhetsoppdateringene og skalerbare løsninger som kan forbedre operasjonell effektivitet. Ved å migrere til skyen kan virksomheter også redusere sitt karbonavtrykk, ettersom store skyleverandører ofte drifter datasentre med energieffektive løsninger (IBM, u.å.).

Retire-strategien brukes for applikasjoner som ikke lenger har forretningsverdi og kan avvikles eller arkiveres. Ved å fjerne slike applikasjoner kan virksomheter eliminere unødvendige kostnader for vedlikehold, infrastruktur og lisensiering. I mange tilfeller identifiseres "zombie-applikasjoner" (med svært lav ressursbruk) eller "idle-applikasjoner" (som har vært inaktive over en lengre periode) som kandidater for avvikling. Dette bidrar til å optimalisere IT-miljøet og redusere sikkerhetsrisikoer knyttet til utdaterte systemer (AWS, u.å.).

Retain-strategien innebærer å holde applikasjoner i det eksisterende miljøet, enten midlertidig eller permanent. Noen applikasjoner krever en detaljert vurdering før migrering, mens andre har regulatoriske eller sikkerhetsmessige krav som gjør at de må forbli lokalt. Applikasjoner med fysiske avhengigheter, som spesialisert maskinvare eller mainframe-løsninger, kan også være kandidater for denne strategien. I noen tilfeller velger virksomheter å beholde applikasjoner frem til en SaaS-erstatning blir tilgjengelig (AWS, u.å.).

Rehost, også kjent som "lift and shift", er den mest direkte migreringsstrategien. Applikasjoner flyttes til skyen uten endringer i arkitektur eller kode. Denne tilnærmingen brukes ofte i store migreringsprosjekter, hvor målet er rask overgang til skyen med minimal forstyrrelse. Rehost kan automatiseres ved hjelp av verktøy som AWS Application Migration Service eller VM Import/Export, noe som gjør det mulig å migrere store mengder arbeidsbelastninger raskt. Selv om denne strategien gir rask gevinst, utnytter den ikke nødvendigvis skyens fulle potensial, og videre optimalisering kan være nødvendig etter migreringen (AWS, u.å.).

Relocate-strategien innebærer å flytte applikasjoner eller hele servermiljøer til en skybasert versjon av samme plattform, uten å endre applikasjonen eller infrastrukturen vesentlig. Dette kan innebære å flytte virtuelle maskiner mellom ulike AWS-regioner eller konti eller å overføre Amazon RDS-databaser til et annet VPC-miljø. Fordelen med relocate er at det gir en rask overgang til skyen uten behov for større omstrukturering, samtidig som det gir muligheter for videre optimalisering etter migreringen (AWS, u.å.).

Repurchase-strategien innebærer å erstatte en eksisterende applikasjon med en ny, skybasert løsning. Dette er spesielt vanlig for programvare som krever omfattende vedlikehold eller har høye lisenskostnader. En vanlig brukssituasjon er å gå fra en lokal installasjon av en forretningsapplikasjon til en Software as a Service (SaaS)-modell, som for eksempel å migrere fra en on-premises CRM-løsning til Salesforce eller Microsoft 365.

Fordelen med repurchasing er at virksomheter kan redusere administrasjonskostnader og raskt dra nytte av moderne, skybaserte funksjoner (AWS, u.å.).

Replatform-strategien innebærer å migrere en applikasjon til skyen med mindre tilpasninger for å forbedre ytelse, sikkerhet eller kostnadseffektivitet. I motsetning til rehost, som er en direkte overføring, optimaliseres applikasjonen til en viss grad under migreringen. Dette kan for eksempel være å migrere en Microsoft SQL Server-database til Amazon RDS for SQL Server, eller å flytte en tradisjonell serverbasert applikasjon til containere ved bruk av AWS App2Container. Ved å implementere noen skyoptimaliseringer tidlig, kan virksomheter oppnå besparelser og forbedret driftseffektivitet uten å måtte gjøre en fullstendig omstrukturering (AWS, u.å.).

Refactor-strategien, også kjent som re-architecting, innebærer en fullstendig modernisering av applikasjonen for å utnytte skyens fulle potensial. Dette kan innebære å bryte ned en monolittisk applikasjon til en mikrotjenestearkitektur, migrere til en serverløs plattform (f.eks. AWS Lambda) eller bruke moderne database- og lagringsløsninger som Amazon Aurora eller DynamoDB. Refactoring gir store fordeler når det gjelder skalerbarhet, kostnadsbesparelser og ytelse, men er også den mest tidkrevende og komplekse strategien. Derfor anbefales denne strategien ofte etter at en applikasjon allerede er migrert til skyen gjennom en enklere strategi som rehost eller replattform (AWS, u.å.).

Cloud-migrering byr på flere utfordringer som organisasjoner må være forberedt på å håndtere. Blant de vanligste utfordringene er kostnadsoverskridelser, som kan oppstå hvis skyforbruket ikke overvåkes og optimaliseres kontinuerlig. Ytelsesproblemer kan også oppstå, spesielt hvis applikasjoner ikke er optimalisert for skyen eller hvis auto-skalering ikke er konfigurert riktig. Vendor lock-in er en annen risiko, der virksomheter kan bli avhengige av en enkelt skyleverandør, noe som kan begrense fleksibiliteten og øke kostnadene på lang sikt (IBM, u.å.).

En vellykket migrering krever grundig planlegging og en robust strategi for kostnadsstyring, ytelsesoptimalisering og risikohåndtering. IBM Turbonomic tilbyr løsninger som kan hjelpe virksomheter med å forutse migrasjonsutfordringer og optimere kostnader og ytelse. Plattformen gir innsikt i migreringsscenarioer, inkludert "lift and shift" eller en mer optimalisert migrering, hvor ressurser tilpasses for maksimal kostnadseffektivitet og ytelse (IBM, u.å.).

Valget av riktig migreringsstrategi avhenger av virksomhetens forretningsmål, tekniske krav og ressurskapasitet. Rehost, relocate og replattform er ofte de mest praktiske strategiene for store migreringer, siden de krever minimal endring i applikasjonene og gir en rask overgang til skyen. Refactor gir de største fordelene på lang sikt, men krever en grundig modernisering og utvikling av applikasjonen. Retire og retain hjelper virksomheter med å fjerne eller utsette migrering av applikasjoner som ikke gir tilstrekkelig forretningsverdi.

Oppgave 7:

En Web Application Firewall (WAF) er en avansert sikkerhetsløsning som beskytter webapplikasjoner, API-er og mobile applikasjoner mot cybertrusler ved å overvåke, filtrere og blokkere skadelig trafikk. WAF er spesielt utviklet for å forhindre angrep som SQL-

injeksjoner, cross-site scripting (XSS), distribuerte tjenestenektangrep (DDoS), cookie-manipulasjon og filinkludering. Ved å operere på applikasjonslaget (Layer 7) i OSI-modellen, gir WAF en spesialisert forsvarsmekanisme som utfyller tradisjonelle brannmurer og inntrengingsforebyggende systemer (IPS) (Cisco, u.å.; Palo Alto Networks, u.å.).

I takt med økende bruk av skybaserte applikasjoner og API-er har behovet for spesifikke sikkerhetsløsninger økt. Tradisjonelle nettverksbrannmurer er ikke designet for å håndtere de unike sikkerhetsutfordringene knyttet til webapplikasjoner, da disse må være tilgjengelige for brukere over hele verden. WAF fyller dette gapet ved å fungere som en barriere mellom internett og webapplikasjonen, der den filtrerer trafikk basert på forhåndsdefinerte regler og sikkerhetspolicyer (Palo Alto Networks, u.å.).

WAF-teknologien oppstod på slutten av 1990-tallet som et svar på økende angrep mot webapplikasjoner. De første WAF-ene var enkle løsninger som primært beskyttet mot innsending av ulovlige tegn i input-felter. I dag har WAF utviklet seg til å være en intelligent sikkerhetsplattform som bruker maskinlæring, signaturbasert gjenkjenning og avviksdeteksjon for å identifisere både kjente og ukjente trusler (Palo Alto Networks, u.å.).

Trusselbildet har også endret seg dramatisk. Moderne angrep er ofte automatisk genererte og rettet mot API-er, skyapplikasjoner og distribuerte systemer. De nyeste OWASP Top 10-listene viser en økning i sårbarheter knyttet til ødelagt tilgangskontroll, kryptografiske feil og feilkonfigurasjoner. WAF gir et ekstra beskyttelseslag som bidrar til å mitigere disse risikoene ved å filtrere mistenkelige forespørsler før de når applikasjonen (Cisco, u.å.; Palo Alto Networks, u.å.).

En WAF beskytter webapplikasjoner ved å analysere HTTP-forespørsler og -svar for mistenkelig aktivitet. Den kan blokkere eller begrense trafikk basert på IP-adresser, HTTP-headere, forespørselsparametere, sesjonsdata og adferdsmønstre. WAF-er kan konfigureres til å følge enten en positiv sikkerhetsmodell (allowlist), en negativ sikkerhetsmodell (blocklist), eller en kombinasjon av begge.

Den positive sikkerhetsmodellen tillater kun godkjent trafikk, mens den negative modellen blokkerer trafikk som matcher kjente angrepsmønstre. Mange moderne WAF-er kombinerer disse metodene for å oppnå best mulig balanse mellom sikkerhet og funksjonalitet (Cisco, u.å.).

WAF-er kan også utføre avansert analyse av inn- og utgående trafikk for å forhindre datalekkasjer. De kan stoppe forsøk på dataeksfiltrering ved å overvåke sensitiv informasjon i HTTP-svar og blokkere uautorisert deling av kredittkortnumre, personopplysninger (PII) og konfidensielle data (Palo Alto Networks, u.å.).

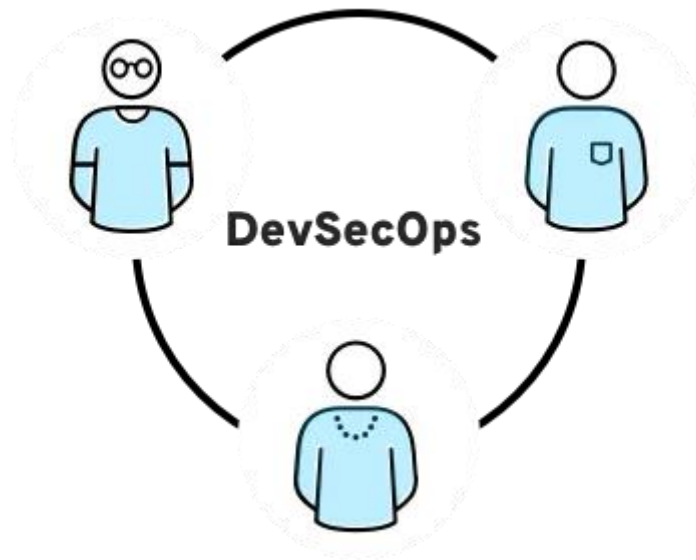
Oppgave 8:

Tradisjonell programvareutvikling følger ofte en sekvensiell modell, hvor prosjekter deles inn i faser for planlegging, design, utvikling, integrasjon og testing. Disse stegene kan ta måneder eller til og med år før et produkt er klart for lansering. Selv om denne tilnærmingen

er metodisk, har mange organisasjoner erfart at den er for treg og ikke møter kundenes forventninger om kontinuerlige forbedringer. I tillegg blir sikkerhet ofte lagt til helt til slutt, noe som øker risikoen for sårbarheter og sikkerhetsbrudd (Microsoft, u.å.).

For å møte disse utfordringene har mange virksomheter gått over til DevOps, hvor utviklings- og driftsteam samarbeider for å levere mindre, hyppigere oppdateringer av kode med høy kvalitet. Automatisering, standardiserte prosesser og tverrfaglig samarbeid gjør det mulig å bevege seg raskere uten at kvaliteten ofres.

DevSecOps bygger videre på DevOps ved å integrere sikkerhet i alle aspekter av utviklingsprosessen, fra planlegging til produksjon. I stedet for at sikkerhet først vurderes i slutfasen, tar hele teamet ansvar for sikkerhet, kvalitetssikring og kodeintegrasjon gjennom hele livssyklusen. Dette innebærer at sikkerhetsimplikasjoner diskuteres allerede i planleggingsfasen, og at sikkerhetstesting starter i utviklingsmiljøet, ikke kun etter at produktet er ferdigstilt. Denne tilnærmingen omtales ofte som "shift left security", hvor sikkerhetsarbeid flyttes så tidlig som mulig i utviklingsløpet (Microsoft, u.å.; Red Hat, 2023).



Utklipp hentet fra RedHat sin side.

Ved å kombinere prinsippene fra DevOps med et sterkt sikkerhetsfokus, gir DevSecOps virksomheter muligheten til å levere sikre, robuste og skalerbare applikasjoner uten at utviklingshastigheten reduseres. I en tid der trusselbildet stadig utvikler seg og applikasjoner distribueres i skybaserte miljøer og mikrotjenestearkitekturer, er DevSecOps avgjørende for å opprettholde en balanse mellom innovasjon og sikkerhet.

Oppgave 9:

For å bruke AWS, Azure eller GCP på en sikker måte, er det viktig å forstå de grunnleggende sikkerhetsprinsippene og beste praksisene som beskytter skyinfrastrukturen mot trusler og feilkonfigurasjoner. Ifølge Varghese (2025) er det flere viktige aspekter man må mestre for å sikre en trygg skyplattform.

En av de viktigste komponentene er Identitet- og tilgangsstyring (IAM), som styrer hvem som har tilgang til hvilke ressurser. IAM bør konfigureres med prinsippet om minst privilegium (PoLP) og Zero Trust, slik at brukere og tjenester kun får den nødvendige tilgangen de trenger. AWS tilbyr IAM med muligheter for å definere detaljerte roller og tillatelser, mens Azure bruker Azure Active Directory og Role-Based Access Control (RBAC) for å administrere tilganger. På Google Cloud fungerer IAM på en annen måte ved å differensiere mellom Google-kontoer (for brukere) og tjenestekontoer (for applikasjoner) (Varghese, 2025).

Videre er flerfaktorausautentisering (MFA) en kritisk sikkerhetsmekanisme for å hindre uautorisert tilgang. AWS krever at MFA aktiveres for alle IAM-brukere med konsolltilgang, mens Azure og GCP tilbyr tilsvarende MFA-løsninger med sterk autentisering. MFA gir en ekstra sikkerhetsbarriere ved å kreve en sekundær bekreftelse, for eksempel en engangskode sendt til en mobilenhet (Varghese, 2025).

For å beskytte data, må kryptering av data i transitt og i hvile være på plass. AWS bruker Transport Layer Security (TLS) for å kryptere data under overføring og tilbyr Amazon S3 med server-side encryption for å beskytte data i hvile. Azure sikrer dataoverføring gjennom VPN Gateway og bruker Azure Key Vault til å håndtere krypteringsnøkler, mens GCP anvender HTTPS for sikker overføring og Cloud KMS for nøkkelhåndtering (Varghese, 2025).

Nettverkssikkerhet er en annen viktig del av skysikkerhet. AWS bruker Virtual Private Cloud (VPC) til å segmentere nettverk og har dedikerte brannmurer på flere lag. Azure tilbyr et omfattende nettverkssikkerhetsrammeverk gjennom Azure Firewall og DDoS Protection, mens GCP sikrer trafikkflyten med Cloud Armor og en global IP-infrastruktur med minimal eksponering for det offentlige internettet (Varghese, 2025).

En av de største sikkerhetsutfordringene i skyen er feilkonfigurerte ressurser, noe som kan føre til datainnbrudd. Derfor er det avgjørende å ha en strukturert patch- og oppdateringsstrategi. AWS Systems Manager Patch Manager automatiserer patching av ressurser, Azure Update Management sikrer at oppdateringer distribueres i hybridmiljøer, og GCPs OS Patch Management gir en samlet plattform for å identifisere og implementere nødvendige sikkerhetsoppdateringer (Varghese, 2025).

For å oppdage og reagere på sikkerhetshendelser er logging og overvåking avgjørende. AWS CloudWatch Logs og CloudTrail gir innsyn i systemaktiviteter, mens Azure Monitor og Sentinel muliggjør avansert trusseldeteksjon. GCP benytter Cloud Logging og Cloud Monitoring for å identifisere og håndtere sikkerhetshendelser (Varghese, 2025).

En omfattende sikkerhetsstrategi må også inkludere sikkerhetskopiering og katastrofegjenoppretting. AWS CloudEndure Disaster Recovery muliggjør rask gjenoppretting av systemer, Azure Site Recovery gir høy tilgjengelighet, og GCP tilbyr tilpassede strategier for sikkerhetskopiering og replikering av data (Varghese, 2025).

Regelmessige sikkerhetsrevisjoner er viktige for å opprettholde en sterk sikkerhetsstilling. AWS Amazon Inspector skanner systemer for sårbarheter, Azure Security Center gir kontinuerlig sikkerhetsvurdering, og GCP Trust and Security Center tilbyr innsikt i sikkerhetsstatusen til skyressursene (Varghese, 2025).

For å forhindre tap av sensitive data er Data Loss Prevention (DLP) en kritisk funksjon. AWS bruker Amazon Macie for å oppdage og beskytte sensitiv informasjon, Azure integrerer DLP i Microsoft 365 Compliance Center, og GCPs Cloud DLP gir verktøy for identifisering og anonymisering av sensitive data (Varghese, 2025).

Et annet grunnleggende prinsipp i skysikkerhet er minste privilegium (PoLP), der brukere og tjenester kun får tilgang til de ressursene de trenger. AWS, Azure og GCP implementerer dette gjennom IAM-roller, RBAC og detaljerte tilgangskontroller for å redusere risikoen for utilsiktet eller ondsinnet tilgang (Varghese, 2025).

Til slutt er opplæring av ansatte avgjørende for å styrke organisasjonens sikkerhetskultur. Mange sikkerhetsbrudd skjer på grunn av menneskelige feil eller sosial manipulering. AWS, Azure og GCP tilbyr alle egne treningsprogrammer for å øke sikkerhetskompetansen til brukere og administratorer (Varghese, 2025).

For å designe sikre skyarkitekturer bør organisasjoner følge de anbefalte sikkerhetsrammeverkene fra hver skyleverandør. AWS har Well-Architected Framework, Azure tilbyr Security Best Practices, og GCP gir Security Foundations Guide. Disse veiledningene gir beste praksis for risikohåndtering, sikkerhetskonfigurasjon og samsvar med bransjestandarder (Varghese, 2025).

Oppgave 10:

Bruken av skytjenester har blitt stadig mer utbredt i næringslivet på grunn av fleksibiliteten, tilgjengeligheten og kostnadseffektiviteten de tilbyr. Likevel er det flere ulemper som må vurderes, og det finnes situasjoner der det kan være fordelaktig å ikke bruke skyen, avhengig av virksomhetens behov og sikkerhetskrav.

En av de største ulempene med skytjenester er den begrensede kontrollen over data og infrastruktur. Siden skyleverandøren eier infrastrukturen, kan virksomheter oppleve utfordringer med å verifisere hvor dataene lagres fysisk, noe som kan ha konsekvenser for både personvern og etterlevelse av regulatoriske krav (Madders, 2021). Ifølge Nasjonal sikkerhetsmyndighet (NSM) er det også viktig å vurdere hvilken sikkerhetsrisiko som oppstår ved at en tredjepart håndterer virksomhetens data. NSM peker på at virksomheter bør ha en klar forståelse av hvordan data lagres, hvem som har tilgang, og hvilke sikkerhetsmekanismer som er på plass for å beskytte informasjonen (NSM, 2023).

Sikkerhet er et annet viktig aspekt. Feilkonfigurasjon av skytjenester kan føre til sårbarheter som gjør det enklere for angripere å få tilgang til sensitive data. Videre kan en avhengighet av internettforbindelse gjøre virksomheter sårbare for nedetid eller tjenestefeil, noe som kan ha store økonomiske konsekvenser dersom en kritisk skytjenesteleverandør opplever driftsproblemer (Madders, 2021).

Til tross for disse utfordringene finnes det fordeler ved ikke å bruke skyen, spesielt for virksomheter med strenge krav til datalagring og sikkerhet. Lokale servere gir full kontroll over dataene og reduserer risikoen for at uvedkommende får tilgang til sensitiv informasjon. For små og mellomstore bedrifter med stabile behov kan det også være mer kostnadseffektivt å ha en lokal løsning fremfor å betale for skybaserte tjenester med variable kostnader (Madders, 2021).

En mulig løsning på utfordringene med skyen er en hybrid tilnærming, der virksomheter kombinerer lokale løsninger med skytjenester. NSM anbefaler en risikobasert tilnærming der virksomheter vurderer hvilke systemer og data som egner seg for skytjenester, og hvilke som bør lagres lokalt for bedre kontroll og sikkerhet (NSM, 2023). En hybrid løsning kan gi det beste fra begge verdener: fleksibilitet og skalerbarhet fra skyen, kombinert med økt kontroll og sikkerhet gjennom lokal lagring av kritiske data.

Skytjenester har mange fordeler, men det er avgjørende at virksomheter gjennomfører en grundig risikovurdering og vurderer en hybrid tilnærming dersom det gir en bedre balanse mellom sikkerhet, kostnader og fleksibilitet.

Kilder:

Amazon. (u.å.). About the migration strategies. Amazon.

<https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html>

Amazon. (u.å.). What is Cloud Native?. Amazon.

<https://aws.amazon.com/what-is/cloud-native/#:~:text=Cloud%20native%20is%20the%20software,quickly%20to%20meet%20customer%20demands>

Basalisco, B., De Michel, F., Salmaso, E., & Okholm, H. B. (2023). *Den økonomiske påvirkningen av cloud i Norge*. Copenhagen Economics.

https://copenhageneconomics.com/wp-content/uploads/2023/10/Copenhagen-Economics-AWS-Nordic-Study_NO.pdf

Cisco. (u.å.). What is a WAF?. Cisco.

<https://www.cisco.com/site/au/en/learn/topics/security/what-is-web-application-firewall-waf.html>

Cloudflare. (u.å.). What is the cloud? Cloudflare.

<https://www.cloudflare.com/learning/cloud/what-is-the-cloud/>

Google. (u.å.). Advantages and Disadvantages of Cloud Computing. Google.

<https://cloud.google.com/learn/advantages-of-cloud-computing?hl=nb>

Google. (u.å.). What are the different types of cloud computing?. Google.

<https://cloud.google.com/discover/types-of-cloud-computing?hl=nb>

Google. (u.å.). What is cloud security?. Google.
<https://cloud.google.com/learn/what-is-cloud-security?hl=nb>

Google. (u.å.). What is Cloud Native?. Google.
<https://cloud.google.com/learn/what-is-cloud-native?hl=nb>

IBM. (u.å.). Cloud migration best practices: Optimizing your cloud migration strategy. IBM.
<https://www.ibm.com/think/insights/cloud-migration-strategy#:~:text=There%20are%20several%20types%20of,%2C%20refactoring%2C%20repurchasing%20and%20retiring>

Khan, H., Kabanov, I., Hua, Y., & Madnick, S. (2023). A systematic analysis of the Capital One data breach: Critical lessons learned. *ACM Transactions on Privacy and Security*, 26(1), 3.
<https://doi.org/10.1145/3546068>

Madders, C. (2021, June 14). *The advantages and disadvantages of cloud computing: Is your head in the cloud?* CyberTec. <https://info.cybertecsecurity.com/advantages-and-disadvantages-of-cloud-computing>

Microsoft. (u.å.). Benefits of Cloud Migration. Microsoft.
<https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/benefits-of-cloud-migration>

Microsoft. (u.å.). What is DevSecOps?. Microsoft.
<https://www.microsoft.com/en-us/security/business/security-101/what-is-devsecops>

Nasjonal sikkerhetsmyndighet. (2020, September 21). *Skytjenester og tjenesteutsetting – muligheter og utfordringer*. Nasjonal sikkerhetsmyndighet.
<https://nsm.no/regelverk-og-hjelp/rapporter/helhetlig-digitalt-risikobilde-2020/skytjenester-og-tjenesteutsetting-muligheter-og-utfordringer/>

Palo Alto Networks. (u.å.). What is a WAF? | Web Application Firewall Explained. Palo Alto Networks.
<https://www.paloaltonetworks.com/cyberpedia/what-is-a-web-application-firewall>

RedHat. (2023). What is DevSecOps?. RedHat.
<https://www.redhat.com/en/topics/devops/what-is-devsecops>

Varghese, J. (2025, January 9). 11 Cloud Security Best Practices For AWS, Azure, And GCP. *Cloud Security Best Practices*. Astra.
<https://www.getastra.com/blog/cloud/cloud-security-best-practices/>