

Skysikkerhet Eksamen

Oppgave A:

Informasjon gitt:

- Lite penger
- Høy fleksibilitet trengs.
- «Legacy» applikasjoner.
- Stram tidsramme.
- Kan ikke patche sårbarheter på selve systemet.
- Beskytte mot normale angrep ved bruk av WAF.

Løsningen må inneholde:

- Backup og recovery.
- Bruk av Terraform.
- Fleksibilitet.
- WAF

Det første jeg må gjøre er å finne ut av hvilken migrering strategi å bruke.

Basert på scenarioet og informasjonen over, vil den beste migrerings strategien være «lift and shift». Dette er basert på tidsrammen, og siden vi ikke får noe relevant informasjon om hva som faktisk kjører på selve maskinen.

Det er også andre alternativer som kunne vært brukt. Her er en kort oppsummering av de resterende alternativene:

- **Retire:**
 - Hva: Fjerne applikasjonen helt fra drift.
 - Relevans: Kunne vært relevant om applikasjonen ikke var kritisk, men dette er ikke mulig siden applikasjonen er avgjørende for virksomheten.
- **Retain:**
 - Hva: La applikasjonen bli der den er, på eksisterende infrastruktur.
 - Relevans: Ikke relevant fordi applikasjonen må flyttes grunnet tidskritisk risiko (flom) og manglende mulighet til å oppgradere infrastrukturen.
- **Relocate:**
 - Hva: Flytte eksisterende virtuelle maskiner direkte til skyen uten modifikasjon (også kjent som "VMware on Cloud").
 - Relevans: Ikke relevant, fordi det krever en eksisterende virtualisert løsning, som vi ikke har informasjon om.
- **Repurchase:**
 - Hva: Erstatte den gamle applikasjonen med en skybasert løsning (f.eks. SaaS).
 - Relevans: Potensielt mulig, men ikke realistisk på grunn av tidspress og spesifikke krav til legacy-applikasjonen.

- **Replattform:**
 - Hva: Gjøre minimale endringer for å optimalisere applikasjonen for skyen (f.eks. migrere til en PaaS-plattform).
 - Relevans: Kunne vært en god løsning for å dra nytte av skyens funksjonalitet, men tidsrammen gjør dette utfordrende.
- **Refactor or Re-Architect:**
 - Hva: Skrive om applikasjonen for å optimalisere for skyen og moderne teknologier.
 - Relevans: Ikke mulig innen tids- og budsjettbegrensningene.

Oppsummert: Av alle disse alternativene er lift and shift det mest praktiske og realistiske valget for situasjonen, gitt tidsrammen, og budsjettet.

Selv om «lift and shift» er den beste strategien for dette scenarioet, er det viktig å være klar over begrensningene:

- Ved bruk av denne metoden vil alle sårbarheter i selve applikasjonen fortsatt være til stede.
- Applikasjonen vil ikke bli optimalisert for skyytelse.

Det anbefales på det sterkeste at når det er mulig anbefales det og patche applikasjoner og optimalisere for skytjenester.

Kilder som er brukt her: <https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html>

Hvilken Cloud modell skal vi bruke?

Valget av IaaS (Infrastructure-as-a-Service) virker helt riktig for en "lift and shift"-strategi. Dette er fordi denne strategien innebærer å flytte eksisterende applikasjoner og arbeidsmengder til skyen med minimal endring. La oss utforske hvorfor IaaS passer godt i dette tilfellet:

1. Full kontroll over applikasjoner: IaaS gir kontroll over operativsystemet og applikasjoner, noe som lar dere opprettholde eksisterende programvare og innstillinger uten omfattende tilpasning.
2. Ingen bekymringer for maskinvare: Leverandøren håndterer maskinvareadministrasjon, noe som betyr at dere slipper å tenke på vedlikehold, oppgradering og skalering av fysisk infrastruktur.
3. Fleksibilitet i oppsett: Dere kan tilpasse infrastrukturen (som CPU, RAM, lagring og nettverk) til deres spesifikke behov, akkurat som på egne servere, men med fordelene fra skyen.
4. Kostnadseffektivitet i migrasjonen: Lift-and-shift krever minimalt med redesign, noe som gjør det kostnadseffektivt og raskt å implementere.
5. Skalerbarhet etter behov: IaaS-plattformer lar dere enkelt skalere ressursene opp eller ned basert på arbeidsmengde og krav, noe som gir bedre kostnadskontroll over tid.

Dette valget gir en god balanse mellom kontroll og fleksibilitet og sikrer en smidig overgang til skyen uten å tvinge store endringer i applikasjonsarkitekturen. I senere faser, når applikasjonene eventuelt optimaliseres for skyen, kan dere vurdere å bevege dere mot PaaS (Platform-as-a-Service) eller SaaS (Software-as-a-Service), dersom det passer for deres behov.

Kilde brukt: <https://www.ibm.com/topics/iaas-paas-saas>

Hvor skal vi deploye?

Vi skal deploye i regionen europe-north1, som er GCPs datasenter i Finland, med sonen europe-north1-a. Dette valget er basert på at selskapet har planer om å levere produkter globalt, men samtidig antas det at selskapet i hovedsak opererer i Norge. Finland er den nærmeste GCP-regionen, noe som gir lavere latency for norske brukere og sikrer samtidig at datahåndteringen er i samsvar med GDPR-regelverket. Denne regionen gir en god balanse mellom lokal ytelse og fremtidig mulighet for global ekspansjon uten betydelige endringer i infrastrukturen.

Hvordan skal vi sørge for en fleksibel og skalerbar løsning?

For å sikre en fleksibel og skalerbar løsning, benytter vi oss av Infrastructure as Code (IaC) ved hjelp av Terraform. Dette gir oss muligheten til å definere, administrere og versjonskontrollere infrastrukturen på en effektiv måte. Terraform muliggjør automatiserte utrullinger og enkel oppdatering av ressurser, noe som reduserer manuelt arbeid og minimerer risikoen for feil. Ved å implementere lastbalansører i arkitekturen, fordeles trafikken dynamisk mellom flere serverinstanser. Dette sikrer både høy tilgjengelighet og forbedret ytelse, samtidig som systemet enkelt kan skalere opp eller ned i takt med belastningen. Kombinasjonen av IaC og lastbalansører gir en robust, smidig og fremtidsrettet løsning.

Hvordan skal vi implementere en sikker løsning?

For å implementere en sikker løsning skal vi bruke GCP Cloud Armour til å beskytte applikasjonen mot vanlige angrep som SQL-injeksjon og XSS. Dette oppnås ved å aktivere predefinerte WAF-regler som blokkerer mistenkelig trafikk før den når applikasjonen. Applikasjonen vil plasseres i et isolert Virtual Private Cloud (VPC)-nettverk med brannmurregler som begrenser tilgang til nødvendige porter som HTTP og HTTPS.

Backup og recovery sikres gjennom automatiserte snapshots av applikasjonens data, med daglige sikkerhetskopier.

Terraform plan

Jeg vil dele terraform koden opp i 4 forskjellige filer for effektivitet og skalerbarhet.

- loadbalancer.tf
- Compute-vm.tf
- Compute-vpc.tf
- Provider.tf
- Security.tf

Sammendrag

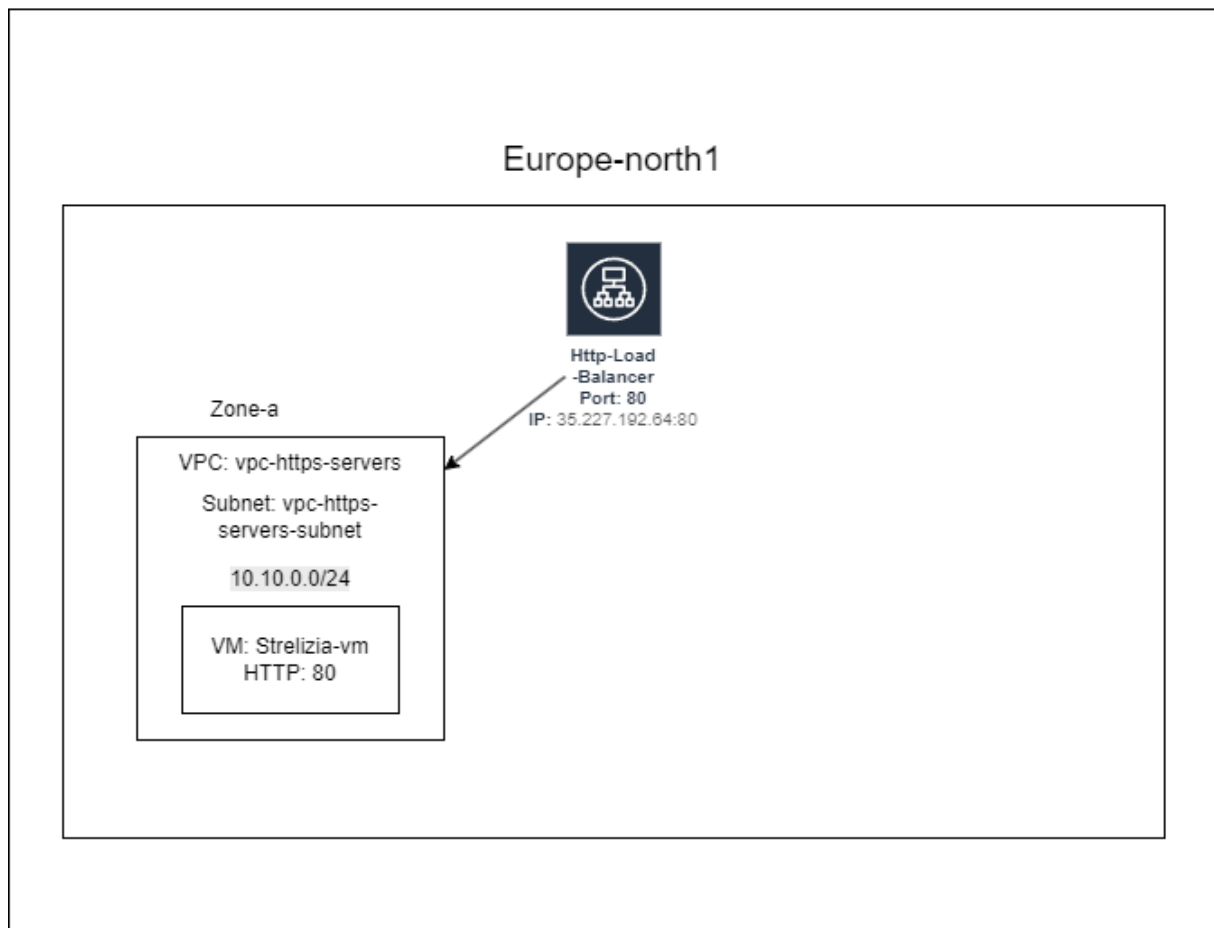
Vi skal bruke Terraform til å sette opp en billig virtuell maskin (VM) som kjører Nginx for å simulere en legacy-applikasjon, hvor installasjonen og konfigurasjonen håndteres via et script. Denne VM-en vil bli plassert i et eget dedikert nettverk for økt isolasjon, og vi velger den rimeligste løsningen for å holde kostnadene nede. Videre skal vi integrere en Layer 7 Web Application Firewall (WAF) for å beskytte applikasjonen mot avanserte trusler og demonstrere hvordan vi kan koble infrastrukturen til et SIEM-system for overvåking og

logganalyse. I tillegg vil vi bruke en lastbalanser for å håndtere trafikk mot applikasjonen. Alt dette implementeres og automatiseres ved hjelp av Terraform-kode.

Diagram

Her er et diagram som viser hvordan det vil se ut.

GCP

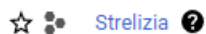


Dokumentasjon

Kode fra dette GitHub-repositoriet, [mInrOz/SKY2100](https://github.com/mInrOz/SKY2100), ble brukt som inspirasjon og hjelp i løpet av arbeidet.

Prosessen er dokumentert med bilder av Terraform-koden for å gi en visuell fremstilling av oppsettet og infrastrukturen. I tillegg er både Terraform-koden og tfstate-filen tilgjengelige i den vedlagte mappen, slik at det er mulig å etterprøve og validere løsningen.

Det første som ble gjort var å opprette et nytt prosjekt i GCP med navn «Strelizia».



strelizia-443612

Utklipp som viser opprettet prosjekt.

Det ble opprettet 5 forskjellige terraform filer istedenfor 4 som jeg egentlig skulle ifølge planen:

- loadbalancer.tf
- Compute-vm.tf
- Compute-vpc.tf
- Provider.tf
- Security.tf

Dette blir gjort basert på fleksibilitet, ryddighet og skalerbarhet.



Utklipp som viser at filene er opprettet.

```
1 provider "google" {  
2   project = "strelizia-443612"  
3   region  = "europe-north1"  
4 }
```

Utklipp som viser provider.tf

```
Strelizia > compute-vm.tf > resource "google_compute_resource_policy" "backup_
1  resource "google_compute_resource_policy" "backup_policy" {
2      name      = "daily-backup-policy"
3      region    = "europe-west4"
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 }
26
27 resource "google_compute_instance_template" "vm_template" {
28     name      = "strelizia-template"
29     machine_type = "e2-micro"
30
31     tags = ["http", "Strelizia", "lb", "17fw"]
32
33     disk {
34         source_image = "debian-cloud/debian-12"
35         auto_delete  = true
36         resource_policies = [
37             google_compute_resource_policy.backup_policy.id
38         ]
39     }
40 }
```

```

shielded_instance_config {
  enable_secure_boot = true
}

network_interface {
  subnetwork = google_compute_subnetwork.vpc-http-servers-subnet.id
}
}

resource "google_compute_instance_group_manager" "igm" {
  name = "igm-strelizia"
  zone = "europe-north1-a"
  version {
    instance_template = google_compute_instance_template.vm_template.id
  }
  base_instance_name = "strelizia"
  target_size        = 1
}

```

Utklipp som viser compute-vm.tf.

Denne koden kombinerer robusthet og skalerbarhet med sikkerhets- og backup-funksjoner for en fremtidssikker infrastruktur. `google_compute_instance_template` definerer en mal for VM-er med maskintypen e2-micro, Debian 12 som operativsystem, og aktiverte sikkerhetsinnstillinger som Secure Boot for å beskytte mot uautoriserte oppstarter. Diskene tilknyttet instansene er konfigurert med en backup-policy (`google_compute_resource_policy`) som automatisk tar daglige snapshots kl. 03:00 UTC, og beholder dem i opptil 7 dager. Backup-policyen legger også til spesifikke metadata for enkel identifikasjon av snapshots.

Instansene administreres av `google_compute_instance_group_manager`, som sikrer at minst en instans alltid kjører, samtidig som det legger til rette for enkel skalering i fremtiden. Dette oppsettet er i tråd med budsjettbegrensninger i oppstartsfasen, med kun en aktiv instans, men er designet for å kunne utvides når trafikken og behovene øker. Samlet sett gir dette oppsettet både pålitelighet, sikkerhet og fleksibilitet, samtidig som det ivaretar kostnadskontroll.

```

strelizia > compute-vpc.tf > ...
1  resource "google_compute_network" "vpc-http-servers" {
2      name                = "vpc-legacy-http-servers"
3      auto_create_subnetworks = false
4  }
5
6  resource "google_compute_subnetwork" "vpc-http-servers-subnet" {
7      name                = "vpc-legacy-http-servers-subnet"
8      ip_cidr_range       = "10.10.0.0/24"
9      network             = google_compute_network.vpc-http-servers.name
10     region              = "europe-north1"
11 }
12

```

Utklipp som viser compute-vpc.tf.

Compute-vpc.tf-filen oppretter en Virtual Private Cloud (VPC) og et tilknyttet subnet i Google Cloud. Ressursen `google_compute_network` definerer en VPC kalt `vpc-legacy-http-servers`,

med parameteren `auto_create_subnetworks` satt til `false`, som sikrer at det ikke opprettes standard subnettverk automatisk. Dette gir bedre kontroll over nettverksoppsettet. Deretter oppretter `google_compute_subnetwork` et subnet med CIDR-adressen `10.10.0.0/24` innenfor den spesifikke VPC-en og regionen `eu-north-1`. Dette undernettet gir en strukturert nettverksinndeling som kan brukes til å plassere ressurser som virtuelle maskiner og andre tjenester. Konfigurasjonen er designet for å gi fleksibilitet, kontroll og sikkerhet i nettverksarkitekturen.

```
Strelizia > load-balancer.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.sql_xss_protection.id
8
9   backend {
10    group = google_compute_instance_group_manager.igm.instance_group
11  }
12
13  health_checks = [google_compute_health_check.http_health_check.self_link]
14 }
15
16 resource "google_compute_health_check" "http_health_check" {
17   name           = "http-health-check"
18   check_interval_sec = 10
19   timeout_sec     = 5
20   healthy_threshold = 2
21   unhealthy_threshold = 2
22
23   http_health_check {
24     request_path = "/"
25     port         = 80
26   }
27 }
28
29 resource "google_compute_url_map" "http_url_map" {
30   name           = "http-url-map"
31   default_service = google_compute_backend_service.http_backend.self_link
32 }
33
34 resource "google_compute_target_http_proxy" "http_proxy" {
35   name       = "http-proxy"
36   url_map    = google_compute_url_map.http_url_map.self_link
37 }
38
39 resource "google_compute_global_forwarding_rule" "http_forwarding_rule" {
40   name       = "http-forwarding-rule"
41   target     = google_compute_target_http_proxy.http_proxy.self_link
42
43   resource "google_compute_global_forwarding_rule" "http_forwarding_rule" {
44     name       = "http-forwarding-rule"
45     target     = google_compute_target_http_proxy.http_proxy.self_link
46     port_range = "80"
47   }
48 }
```

Utklipp som viser load-balancer.tf.

Denne konfigurasjonen setter opp en lastbalanser for HTTP-trafikk, inkludert backend-tjenester, helseovervåking, og trafikkdirigering. `google_compute_backend_service` definerer en backend-tjeneste kalt `http-backend-service`, som mottar HTTP-trafikk på port 80 og bruker en sikkerhetspolicy for å beskytte mot SQL-injeksjon og XSS-angrep. Backend-tjenesten er koblet til en instansgruppe som håndterer forespørsler. Helsekontroller, definert av `google_compute_health_check`, overvåker tilgjengeligheten til instansene ved å sende forespørsler til rotbanen (`/`) hvert 10. sekund og markere instanser som "friske" etter to suksessfulle svar. `google_compute_url_map` kobler innkommende trafikk til backend-tjenesten, mens `google_compute_target_http_proxy` videresender trafikken til riktig tjeneste.

Til slutt definerer `google_compute_global_forwarding_rule` en global regel som lytter på port 80 og dirigerer trafikk fra internett til HTTP-proxyen. Dette oppsettet sikrer effektiv distribusjon og overvåking av trafikk til backend-servere.

```
Strelizia > security.tf > resource "google_compute_security_policy" "sql_xss_protection"
1 resource "google_compute_security_policy" "sql_xss_protection" {
2   name      = "sql-xss-protection-policy"
3   description = "Protect against SQL injection and XSS attacks"
4
5   rule {
6     priority = 1000
7     action   = "deny(403)"
8     match {
9       expr {
10        expression = "evaluatePreconfiguredExpr('sql-injection-stable')"
11      }
12    }
13    description = "Block SQL injection attempts"
14  }
15
16  rule {
17    priority = 2000
18    action   = "deny(403)"
19    match {
20      expr {
21        expression = "evaluatePreconfiguredExpr('xss-stable')"
22      }
23    }
24    description = "Block XSS attacks"
25  }
26 }
```

Utklipp som viser security.tf.

Security.tf-filen oppretter en sikkerhetspolicy i Google Cloud for å beskytte mot SQL-injeksjons- og XSS-angrep. Ressursen `google_compute_security_policy` definerer en policy kalt `sql-xss-protection-policy`, som bruker forhåndsdefinerte regler for å identifisere og blokkere disse truslene. Den første regelen, med prioritet 1000, blokkerer forsøk på SQL-injeksjon ved å evaluere trafikk mot Google Clouds innebygde uttrykk `sql-injection-stable`. Den andre regelen, med prioritet 2000, blokkerer XSS-angrep ved å bruke uttrykket `xss-stable`. Begge reglene returnerer HTTP-statuskode 403 (Forbidden) for trafikk som bryter disse retningslinjene. Denne policyen er ideell for å beskytte applikasjoner ved å filtrere skadelig trafikk før den når backend-tjenestene.

Nå som jeg har klar planen for terraform koden skal jeg kjøre den gjennom `tfsec` for og sørge for at koden er sikker.

```
tfsec is joining the Trivy family
tfsec will continue to remain available
for the time being, although our engineering
attention will be directed at Trivy going forward.
You can read more here:
https://github.com/aquasecurity/tfsec/discussions/1994

Result #1 LOW Subnetwork does not have VPC flow logs enabled.

compute-vpc.tf:6-11
6 resource "google_compute_subnetwork" "vpc-http-servers-subnet" {
7   name           = "vpc-legacy-http-servers-subnet"
8   ip_cidr_range  = "10.10.0.0/24"
9   network        = google_compute_network.vpc-http-servers.name
10  region         = "europe-north1"
11 }

II google-compute-enable-vpc-flow-logs
Impact Limited auditing capability and awareness
Resolution Enable VPC flow logs

More Information
- https://aquasecurity.github.io/tfsec/latest/checks/google/compute/enable-vpc-flow-logs/
- https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_subnetwork#enable_flow_logs
```

Utklipp av Tfsec output.

Tfsec anbefaler å implementere VPC flow logs. VPC logs vil gi et ekstra lag med sikkerhet ved å gi oss detaljert informasjon om nettverks trafikken.

Jeg vil da legge til VPC flow logs.

```
Strelizia > compute-vpc.tf > resource "google_compute_subnetwork" "vpc-http-servers-subnet" > log_config
1  resource "google_compute_network" "vpc-http-servers" {
2    name           = "vpc-legacy-http-servers"
3    auto_create_subnetworks = false
4  }
5
6  resource "google_compute_subnetwork" "vpc-http-servers-subnet" {
7    name           = "vpc-legacy-http-servers-subnet"
8    ip_cidr_range  = "10.10.0.0/24"
9    network        = google_compute_network.vpc-http-servers.name
10   region         = "europe-north1"
11   log_config {
12     aggregation_interval = "INTERVAL_10_MIN"
13     flow_sampling         = 0.5
14     metadata              = "INCLUDE_ALL_METADATA"
15   }
16 }
17
```

Utklipp som viser at VPC logs er lagt til.

Koden aktiverer og konfigurerer VPC Flow Logs for subnettverket, som gir innsikt i nettverkstrafikk. Med `aggregation_interval` satt til `"INTERVAL_10_MIN"`, aggregeres logger hvert 10. minutt, mens `flow_sampling` med verdien 0.5 sikrer at 50 % av trafikken logges, noe som balanserer detaljeringsgrad og kostnader. `metadata` er konfigurert som `"INCLUDE_ALL_METADATA"`, slik at all tilgjengelig metadata om ressursene som genererer trafikk inkluderes. Dette gir detaljerte logger som kan brukes til sikkerhetsanalyse, feilsøking, nettverksoptimalisering og kostnadskontroll ved å overvåke kildes- og destinasjons-IP-er, protokoller, datamengder og brannmurtillatelser for subnettverket.

Deretter kjørte jeg «`terraform init`», og «`terraform plan`».

```
(strelizia-443612)$ terraform init
```

```
/Strelizia (strelizia-443612)$ terraform plan
```

Etter kjøring av terraform plan ble det ikke oppdaget noen error. Deretter ble terraform apply kjørt for å opprette filer osv.

Etter at jeg kjørte «terraform apply» kom errors som indikerte at Compute engine Api ikke var skrudd på.

```
Error: Error creating Network: googleapi: Error 403: Compute Engine API has not been used in project strelizia-443612 before or it is disabled.
```

Utklipp som viser at Compute Engine API ikke er skrudd på.

Det ble skrudd på ved å gå inn på API-siden.

Deretter ble det prøvd å kjøre terraform apply.

Jeg fikk da en ny error som tilsier at jeg trenger å legge til en default rule i security.tf:

```
Error: Error creating SecurityPolicy: googleapi: Error 400: Invalid value for field 'resource.rules[0]': Every security policy must have a default rule at priority 2147483647 with match condition *, invalid
```

Jeg la da til denne regelen:

```
6 rule {
7   priority    = 2147483647
8   action      = "allow"
9   match {
10    versioned_expr = "SRC_IPS_V1"
11    config {
12      src_ip_ranges = ["*"]
13    }
14  }
15  description = "Default rule to allow traffic"
16 }
17 }
```

Utklipp som viser default rule.

Etter at koden ble kjørt med terraform apply, oppdaget jeg at ingen maskiner ble opprettet. Gjennom feilsøking fant jeg ut at backup-regionen måtte samsvare med regionen der maskinene ble deployert. Jeg oppdaterte derfor regionen til europe-north1. I tillegg la jeg til et installasjonsskript som laster ned og konfigurerer nginx. Dette fungerer som en simulering av Strelizias legacy-applikasjon.

De endringene som ble gjort, sikrer at infrastrukturen nå fungerer korrekt og effektivt med lastbalansering og nødvendige sikkerhetstiltak. For å tillate HTTP-trafikk ble det lagt til en firewall rule som åpner port 80 for eksterne tilkoblinger, koblet til VPC-en med riktig

nettverkstag. I tillegg ble named ports konfigurert i `google_compute_instance_group_manager` for å mappe portnavnet `http` til portnummer 80, noe som er nødvendig for at lastbalansererer skal kunne rute trafikk til backend-instanser.

Backend-tjenesten og helsesjekken ble også konfigurert slik at lastbalansererer kun ruter trafikk til sunne instanser.

Videre oppdaget jeg gjennom feilsøking at maskinen ble markert som `unhealthy` fordi den manglet internettilgang. Dette ble løst ved å gi maskinen en ekstern IP-adresse ved å legge til `access_config()` i koden. Dette sikrer at VM-en har nødvendig nettverkstilgang for å kunne laste ned og kjøre applikasjonen.

```
network_interface {  
  subnetwork = google_compute_subnetwork.vpc_http_servers_subnet.id  
  access_config {}  
}
```

Utklipp som viser at `access_config {}` er blitt lagt til.

Jeg valgte også å splitte opp noen terraform filer slik at firewall og backend får sine egne filer.

- Backend.tf
- Compute-vm.tf
- Compute-vpc.tf
- Firewall.tf
- Load-balancer.tf
- Provicer.tf
- Security.tf

```

Strelizia > backend.tf > resource "google_compute_http_health_check" "health_check" > # healthy_thre:
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" # Must match the named port in the instance group
6      timeout_sec    = 30
7      security_policy = google_compute_security_policy.sql_xss_protection.id
8
9      backend {
10         group = google_compute_instance_group_manager.igm.instance_group
11     }
12
13     health_checks = [google_compute_http_health_check.health_check.self_link]
14 }
15
16 resource "google_compute_http_health_check" "health_check" {
17     name           = "http-health-check"
18     check_interval_sec = 10
19     timeout_sec     = 5
20     healthy_threshold = 2
21     unhealthy_threshold = 2
22     request_path     = "/"
23 }
24

```

Utklipp som viser backend.tf

```

Strelizia / compute-vmlb / resource "google_compute_instance_template" "vm_template" / 00 metadata / Startup-script
1  resource "google_compute_resource_policy" "backup_policy" {
2      name = "daily-backup-policy"
3      region = "europe-north1"
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 }
26
27 resource "google_compute_instance_template" "vm_template" {
28     name = "strelizia-template"
29     machine_type = "e2-micro"
30
31     tags = ["http", "strelizia", "lb", "l7fw"]
32
33     disk {
34         source_image = "debian-cloud/debian-12"
35         auto_delete = true
36         resource_policies = [
37             google_compute_resource_policy.backup_policy.id
38         ]
39     }
40 }

```

```

shielded_instance_config {
  enable_secure_boot = true
}

metadata = {
  startup-script = <<-EOT
  #!/bin/bash
  apt-get update
  apt-get install -y nginx
  systemctl enable nginx
  systemctl start nginx
  EOT
}

network_interface {
  subnetwork = google_compute_subnetwork.vpc_http_servers_subnet.id
  access_config {}
}
}

resource "google_compute_instance_group_manager" "igm" {
  name = "igm-strelizia"
  zone = "europe-north1-a"
  version {
    instance_template = google_compute_instance_template.vm_template.id
  }
  base_instance_name = "strelizia"
  target_size        = 1

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

```

Utklipp som viser compute-vm.tf.

```

Strelizia > compute-vpc.tf > ...
1  resource "google_compute_network" "vpc_http_servers" {
2    name                = "vpc-http-servers"
3    auto_create_subnetworks = false
4  }
5
6  resource "google_compute_subnetwork" "vpc_http_servers_subnet" {
7    name                = "vpc-http-servers-subnet"
8    ip_cidr_range       = "10.10.0.0/24"
9    network              = google_compute_network.vpc_http_servers.name
10   region              = "europe-north1"
11   log_config {
12     aggregation_interval = "INTERVAL_10_MIN"
13     flow_sampling        = 0.5
14     metadata              = "INCLUDE_ALL_METADATA"

```

Utklipp som viser compute-vpc.tf

```

Strelizia > 🐦 firewall.tf > ...
1 resource "google_compute_firewall" "allow_http" {
2   name      = "allow-http"
3   network   = google_compute_network.vpc_http_servers.name
4
5   allow {
6     protocol = "tcp"
7     ports    = ["80"]
8   }
9
10  source_ranges = ["0.0.0.0/0"]
11  target_tags   = ["http"]
12 }
13

```

Utklipp som viser firewall.tf.

```

Strelizia > 🐦 load-balancer.tf > 📄 resource "google_compute_url_map" "http_url_map"
1 resource "google_compute_url_map" "http_url_map" {
2   name          = "http-url-map"
3   default_service = google_compute_backend_service.http_backend.id
4 }
5
6 resource "google_compute_target_http_proxy" "http_proxy" {
7   name      = "http-proxy"
8   url_map   = google_compute_url_map.http_url_map.id
9 }
10
11 resource "google_compute_global_forwarding_rule" "http_forwarding_rule" {
12   name      = "http-forwarding-rule"
13   target     = google_compute_target_http_proxy.http_proxy.id
14   port_range = "80"
15 }
16

```

Utklipp som viser load-balancer.tf

```

Strelizia / 🐦 provider.tf / ...
1 provider "google" {
2   project = "strelizia-443612"
3   region  = "europe-north1"
4 }
5

```

Utklipp som viser provider.tf

```

Strelizia > security.tf > ...
1 resource "google_compute_security_policy" "sql_xss_protection" {
2   name      = "sql-xss-protection-policy"
3   description = "Protect against SQL injection and XSS attacks"
4
5   rule {
6     priority = 1000
7     action   = "deny(403)"
8     match {
9       expr {
10        expression = "evaluatePreconfiguredWaf('sqli-stable')"
11      }
12    }
13    description = "Block SQL injection attempts"
14  }
15
16  rule {
17    priority = 2000
18    action   = "deny(403)"
19    match {
20      expr {
21        expression = "evaluatePreconfiguredWaf('xss-stable')"
22      }
23    }
24    description = "Block XSS attacks"
25  }
26
27  rule {
28    priority = 2147483647
29    action   = "allow"
30    match {
31      versioned_expr = "SRC_IPS_V1"
32      config {
33        src_ip_ranges = ["*"]
34      }
35    }
36    description = "Default rule to allow traffic"
37  }
38 }

```

Utklipp som viser security.tf

Etter at jeg hadde fullført koden, kjørte jeg en ny test med tfsec for å evaluere sikkerheten og identifisere potensielle svakheter. Dette sikret at koden var i tråd med anbefalte sikkerhetsprinsipper og at eventuelle gjenværende problemer kunne adresseres før implementering.

```

Result #1 CRITICAL Firewall rule allows ingress traffic from multiple addresses on the public internet.

firewall.tf:10

1 resource "google_compute_firewall" "allow_http" {
2   ...
10 [ source_ranges = ["0.0.0.0/0"]
11 ...
12 }

II google-compute-no-public-ingress
Impact The port is exposed for ingress from the internet
Resolution Set a more restrictive cidr range

More Information
- https://aquasecurity.github.io/tfsec/latest/checks/google/compute/no-public-ingress/
- https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_firewall#source_ranges
- https://www.terraform.io/docs/providers/google/r/compute_firewall.html

```

Utklipp fra Tfsec.

Tfsec identifiserer en kritisk feil i Terraform-koden, hvor webserveren tillater tilgang fra alle IP-adresser (0.0.0.0/0). Dette gjelder imidlertid kun for port 80, som håndterer HTTP-trafikk. I et privat eller regionspesifikt miljø ville det vært hensiktsmessig å begrense tilgangen til spesifikke IP-adresser eller regioner. Ettersom denne løsningen er ment å være globalt

tilgjengelig, er det nødvendig å holde port 80 åpen for alle IP-adresser. Dette er et bevisst valg som samsvarer med løsningens krav om global tilgjengelighet.

Videre anbefales det å implementere HTTPS for å sikre dataoverføringer, spesielt dersom produktet er en webserver. Det er også lurt å vurdere mekanismer for å blokkere IP-adresser som sender et uvanlig høyt antall forespørsler, for å beskytte mot potensielle angrep som denial-of-service (DoS).

Etter dette ble terraform apply kjørt og alle ressurser ble opprettet uten noen problemer.

Utklipp som viser at ressursene er blitt opprettet:

Filter

Enter property name or value

| <div></div> | Status | Name <div>↑</div> | Zone | Recommendations | In use by | Internal IP | External IP | Connect |
|-------------|--------------|--------------------------------|-----------------|-----------------|-----------|------------------|---------------------|-------------------------------|
| <div></div> | <div>✓</div> | strelizia-cg1z | europa-north1-a | | | 10.10.0.2 (nic0) | 34.88.118.59 (nic0) | SSH <div>▾</div> <div>⋮</div> |

Utklipp som viser at en vm er blitt opprettet.

| Filter Enter property name or value | | | | | | |
|-------------------------------------|------------------------------|-----------------------|-------------|-----------|--------|---|
| <input type="checkbox"/> | Name | Load balancer type | Access type | Protocols | Region | Backends |
| <input type="checkbox"/> | http-url-map | Application (Classic) | External | HTTP | | ✓ 1 backend service (1 instance group, 0 network endpoint groups) ⋮ |

Utklipp som viser at last balansøren er blitt opprettet.

| <input type="checkbox"/> | Name ↑ | Region | Stack Type | Primary IPv4 range |
|--------------------------|---|---------------|---------------------|--------------------|
| <input type="checkbox"/> | vpc-http-servers-subnet | europa-north1 | IPv4 (single-stack) | 10.10.0.0/24 |

Utklipp som viser at VPC er opprettet.

| <input type="checkbox"/> | Status | Name ↑ | Region | Storage Location | Schedule frequency (UTC) |
|--------------------------|--------|-------------------------------------|---------------|---------------------|---|
| <input type="checkbox"/> | ✓ | daily-backup-policy | europa-north1 | eu (European Union) | Every day, starts between 3:00 AM 4:00 AM |


Utklipp som viser at back up er opprettet.

| <input type="checkbox"/> | Status | Name ↑ | Instances | Template | Group type | C |
|--------------------------|--------|-------------------------------|-----------|------------------------------------|------------|---|
| <input type="checkbox"/> | ✓ | igm-strelizia | 1 | strelizia-template | Managed | D |

Utklipp som viser at instanse gruppen er blitt opprettet.



http-url-map

Classic Application Load Balancer

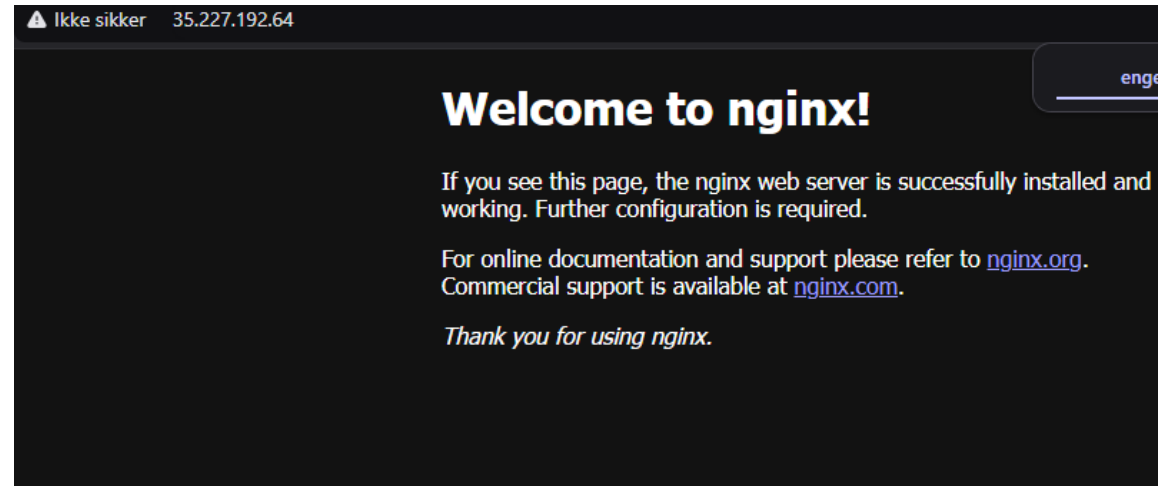
 Faster web performance and improved web protection with Cloud CDN and Cloud Armor. [Learn more](#)

- DETAILS
- MONITORING
- CACHING
- MIGRATION

Frontend

| Protocol  | IP:Port | Certificate | Certificate Map | SSL Policy | Network Tier  |
|--|------------------|-------------|-----------------|------------|--|
| HTTP | 35.227.192.64:80 | - | | | Premium |



Utklipp som viser den eksterne ipen til last balansøren.



Utklipp som viser at det er mulig og nå vmen ved bruk av last balansøren sin eksterne ip.

http-backend-service

General properties

| | |
|---|---|
| Description | Backend service for HTTP traf |
| Load balancer type | Classic Application Load Bala |
| Endpoint protocol | HTTP |
| In use by | http-url-map |
| Timeout  | 30 seconds |
| IP address selection policy  | Only IPv4 |
| Health check | http-health-check VIEW HEAL |
| Backend security policy | sql-xss-protection-policy |

Utklipp som viser at backend, security og health check er blitt opprettet.

B.

Hvordan kan man analysere logger og integrere dem i et SIEM-system?

I GCP finnes det en innebygd Log Explorer som samler logger fra blant annet Audit, VPC, Compute, og andre ressurser i plattformen.

Her har jeg filtrert loggene etter en spesifikk IP-adresse og kan dermed se alle tilkoblinger denne IP-adressen har gjort mot subnettet eller lastbalanseren.

The screenshot shows the Google Cloud Logs Explorer interface. At the top, there's a search bar with the query `src_ip 85.122.203.173`. Below the search bar, there are filters for "All resources", "All log names", "All severities", and "Correlate by". The search results are displayed in a table with columns for "SEVERITY", "TIME", and "SUMMARY". The first result is a log entry from 2024-12-04 15:10:20.537, showing a connection from the source IP 85.122.203.173 to the destination IP 10.10.0.2 on port 80 using protocol 6. The log entry is expanded, showing the full JSON payload.

Log fields

Search fields and values

RESOURCE TYPE

Subnetwork 8

SEVERITY

Default 8

Timeline

Nov 28, 1:00 PM

8 results

Showing logs for last 7 days from 11/28/24, 1:18 PM to 12/5/24, 1:18 PM. [Extend time by: 1 day](#) [Edit time](#)

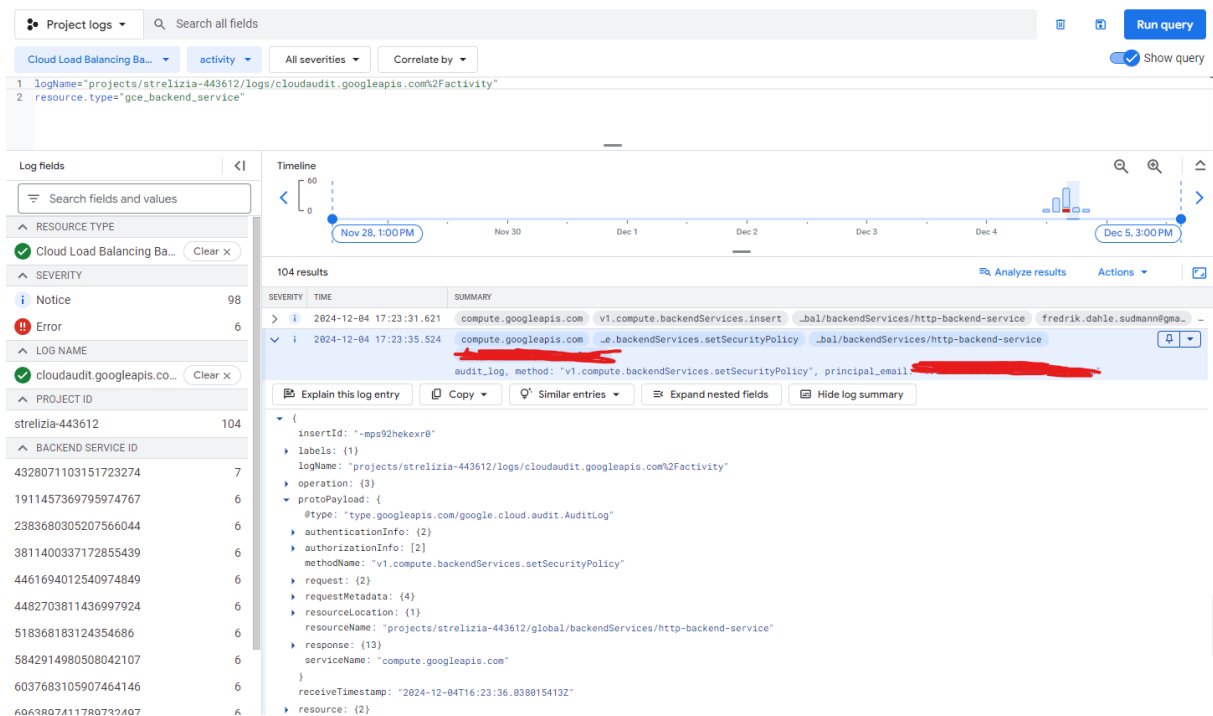
| SEVERITY | TIME | SUMMARY |
|----------|-------------------------|--|
| Info | 2024-12-04 15:10:20.537 | {"bytes_sent": "0", "connection": {}, "dest_instance": {}, "dest_vpc": {}, "end_time": "2024-12-04T14:03:44.821485884Z", "start_time": "2024-12-04T14:03:44.821485884Z"} |

[Explain this log entry](#) [Copy](#) [Similar entries](#) [Expand nested fields](#) [Hide log summary](#)

```
{  "insertId": "mhmv2wffui3b3"  "jsonPayload": {    "bytes_sent": "0"    "connection": {      "dest_ip": "10.10.0.2"      "dest_port": 80      "protocol": 6      "src_ip": "85.122.203.173"    }  }}
```

Utklipp som viser at noen har koblet til last balansøren.

Det er også mulig å hente ut aktivitetslogger (activity logs) som viser hva som opprettes, endres eller slettes i skyen, samt når dette skjer. Under er et eksempel:



Utlipp som viser at en bruker har oppdatert security policyen til http-backend-service.

Hvordan implementere en Siem løsning?

Ved hjelp av Google Clouds integrerte loggfunksjoner kan logger enkelt eksporteres til Splunk for videre analyse og overvåking. Dette muliggjør en robust, skalerbar og feiltolerant løsning som støtter både sikkerhets- og IT-operasjoner.

Denne løsningen samler logger fra organisasjons-, mappe- og prosjektnivåer i en aggregert logg-sink som sender dem videre via en pipeline til Splunk. Arkitekturen inneholder følgende komponenter: Cloud Logging samler logger og sender dem til en aggregert logg-sink som bruker Pub/Sub som destinasjon. Pub/Sub oppretter et tema og abonnement som videresender loggene til Dataflow-pipelines. Dataflow består av en primær pipeline som leverer logger til Splunks HTTP Event Collector (HEC) via Pub/Sub, og en sekundær pipeline som behandler feilede leveranser ved å flytte dem til et separat emne for feilhåndtering. Splunk mottar loggene via HEC for analyse og sikkerhetsovervåking.

Fordelene med denne integrasjonen er mange. Dataflow håndterer ressursene for dataeksport som en administrert tjeneste, noe som reduserer administrasjonsbyrden. Dataflow skalerer automatisk for å håndtere variasjoner i loggvolum og distribuerer arbeidsmengder over flere arbeidere for å unngå "bottle-necks". Feilede leveranser blir automatisk forsøkt sendt på nytt, og uløste meldinger blir sendt til en "dead-letter topic" for inspeksjon. Bruken av Cloud NAT og private IP-adresser beskytter Dataflow-pipeline-VMer, mens HEC-token lagres sikkert i Secret Manager for å minimere risikoen for eksponering. Videre maksimeres datagjennomstrømningen ved bruk av batching og parallell behandling, samtidig som belastningen på Splunk HEC-endepunktet reduseres.

Loggene fra Google Cloud kan brukes til en rekke formål. Sikkerhetsmessig gir de mulighet til å identifisere mistenkelig aktivitet og beskytte mot cyberangrep. For IT-operasjoner kan de forbedre ressursovervåking og drift. Innen samsvar hjelper de med å overholde regulatoriske krav som GDPR, mens de for forretningsanalyse kan generere innsikt fra sanntidsdata.

Med Splunk som SIEM-verktøy oppnår du sanntids trussel deteksjon som forhindrer sikkerhetsbrudd med kontinuerlig overvåking. Splunk gir kontekstualisering ved å korrelere loggdata for et helhetlig bilde av systemet, samtidig som det effektiviserer hendelseshåndtering med automatiserte varsler og rask respons. Kostnadsoptimalisering er også mulig ved å identifisere ineffektiv ressursbruk.

For å implementere løsningen, oppretter du en logg-sink i Cloud Logging som sender logger til et Pub/Sub-emne. Dataflow-templates brukes til å sette opp pipelines som streamer logger til Splunk. Splunk HEC må konfigureres med riktig sikkerhet, inkludert bruk av Secret Manager for tokenhåndtering. Overvåk og analyser loggstrømmer for å sikre optimal ytelse og pålitelighet.

Ved å eksportere logger fra Google Cloud til Splunk oppnår du en robust og skalerbar løsning for sikkerhetsovervåking og operasjonell innsikt. Med riktig arkitektur og optimalisering reduseres risiko, driftseffektiviteten forbedres, og kravene til en moderne, global IT-infrastruktur oppfylles.

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

Oppgave B:

1.

Cloud-teknologi har revolusjonert hvordan vi bruker og administrerer IT-ressurser. Denne innovasjonen er mulig takket være virtualisering, som gjør det mulig for store datasentre med fysisk maskinvare å kjøre mange virtuelle maskiner. Virtualisering tillater skyleverandører å tilby ressurser til langt flere brukere enn det som ville vært mulig kun med fysisk maskinvare (CloudFlare, 2024).

Ved å utnytte ressursene mer effektivt bidrar teknologien til kostnadseffektivitet og reduserer behovet for investering i ekstra maskinvare. For eksempel kan en fysisk server kjøre flere virtuelle maskiner, noe som maksimerer bruken av eksisterende ressurser (Cloudflare, 2024).

Skyleverandører tilbyr et bredt spekter av tjenester, som infrastruktur, applikasjoner, utviklingsverktøy og datalagring. Disse organiseres i tre hovedmodeller:

- **Software-as-a-Service (SaaS):** Programvare levert som en tjeneste (CloudFlare, 2024).
- **Platform-as-a-Service (PaaS):** Utviklingsplattformer som gir verktøy og infrastruktur for å bygge og distribuere applikasjoner (CloudFlare, 2024).
- **Infrastructure-as-a-Service (IaaS):** Grunnleggende infrastruktur som servere, nettverk og lagring, levert som en tjeneste (CloudFlare, 2024).

Fra et økonomisk perspektiv gir skytjenester bedrifter muligheten til å redusere kostnader ved å unngå investeringer i egen infrastruktur (CloudFlare, 2024). I stedet betaler de kun for kapasiteten de faktisk bruker, en modell kjent som "pay-as-you-go." Denne fleksibiliteten gjør det enklere for bedrifter å skalere opp eller ned avhengig av behov. For eksempel tilbyr Google Cloud en slik betalingsmodell, som hjelper bedrifter å håndtere IT-budsjetter på en mer forutsigbar måte (Google, 2024).

Skytjenester reduserer også kostnader knyttet til vedlikehold og oppdateringer, da dette håndteres av leverandørene. Dette frigjør ressurser og tid, slik at bedrifter kan fokusere på sin kjernevirksomhet (CloudFlare, 2024).

2.

Tekniske fordeler med et cloud-oppsett inkluderer muligheten for automatisk skalering av ressurser basert på belastning (IBM, 2024). Brukere kan velge mellom offentlige, private eller hybride skyløsninger, avhengig av behov knyttet til datalagring, sikkerhet og andre krav. For eksempel gir tjenester for datalagring og katastrofegjenoppretting (BDR) bedrifter mulighet til å kopiere og lagre data på en ekstern server. Dette beskytter mot strømbrydd, skade eller tap av data. BDR-tjenester gjør det enklere å gjenopprette data etter korrupsjon, skadelig programvare og andre uforutsette hendelser, noe som sikrer kontinuitet i virksomheten (IBM, 2024).

I dag tilbyr skyløsninger avanserte sikkerhetsfunksjoner, som spesialisert programvare for å beskytte data, applikasjoner og infrastruktur. Disse inkluderer blant annet virtuelle private skyer (VPC), API-nøkler, sikkerhetsinformasjon og hendelseshåndtering (SIEM) og mer (IBM, 2024). Et globalt nettverk av datasentre gjør skybaserte applikasjoner og data tilgjengelige fra praktisk talt alle enheter med internettilgang. Dette gjør det mulig for bedrifter å plassere arbeidsbelastninger nær kundene sine, noe som gir raskere tilkobling og lavere ventetid (IBM, 2024).

Skyplattformer gir også umiddelbar tilgang til infrastruktur, noe som gjør det mulig å raskt utvikle og lansere applikasjoner uten å måtte administrere fysisk maskinvare. DevOps- og IT-team kan raskt teste, forbedre og lansere nye produkter og tjenester, slik at organisasjoner kan få applikasjoner raskere ut på markedet (IBM, 2024). Skyleverandører tar seg av underliggende infrastruktur, noe som frigjør tid og ressurser for organisasjoner til å fokusere på utvikling av applikasjoner og andre prioriteringer. Automatiseringsverktøy som sanntidsdashboards og AI-drevet dataanalyse reduserer manuelle oppgaver og administrative byrder (IBM, 2024).

3.

IaaS (Infrastructure-as-a-Service)

Hva det er:

IaaS gir deg tilgang til IT-infrastruktur som virtuelle servere, lagring og nettverk. Dette er den mest fleksible skytjenestemodellen og gir deg full kontroll over IT-miljøet ditt, samtidig som du slipper å vedlikeholde fysisk maskinvare (IBM, 2024).

Ansvarsfordeling:

- Du har ansvar for:
 - Applikasjoner: Installere og vedlikeholde de programmene du trenger.
 - Operativsystemer: Konfigurere og oppdatere OS som kjører på serverne.
 - Runtime-miljøer: Sørge for at applikasjoner kjører som de skal, inkludert avhengigheter og konfigurasjoner.
 - Data: Sikre og administrere dataene dine, inkludert backup og gjenoppretting.
 - Nettverkskonfigurasjon: Konfigurere brannmur, virtuelle nettverk og tilgangsregler.
- Leverandøren har ansvar for:
 - Maskinvare: Sørge for at serverne, lagringen og nettverket fungerer.
 - Virtualisering: Kjøre og administrere virtuelle maskiner eller containere.

- Datasenter: Vedlikeholde fysiske datasentre, inkludert strøm, kjøling og fysisk sikkerhet.

Fordeler:

- Maksimal fleksibilitet og skalerbarhet.
- Ideell for organisasjoner som trenger kontroll over sine IT-miljøer, men ikke vil vedlikeholde maskinvare.
- Perfekt for applikasjoner med uforutsigbar eller raskt voksende ressursbruk.

Eksempler på bruk:

- Test- og utviklingsmiljøer.
- Hosting av nettsider eller applikasjoner som trenger skalerbarhet.
- Sikkerhetskopiering og katastrofegjenoppretting.

PaaS (Platform-as-a-Service)

Hva det er:

PaaS gir deg en ferdig plattform for å utvikle, teste og distribuere applikasjoner. Plattformen inkluderer alt du trenger, fra operativsystem til utviklingsverktøy, slik at du kan fokusere på koding og innovasjon (IBM, 2024).

Ansvarsfordeling:

- Du har ansvar for:
 - Applikasjoner: Utvikle, teste og administrere applikasjonene dine.
 - Data: Sikre og håndtere dataene dine i applikasjonen.
- Leverandøren har ansvar for:
 - Infrastruktur: Administrere servere, lagring og nettverk.
 - Operativsystemer og runtime-miljøer: Oppdatere og vedlikeholde disse.
 - Utviklingsverktøy: Tilby rammeverk, biblioteker og API-er for enklere utvikling.
 - Sikkerhet: Beskytte plattformen og dens komponenter.

Fordeler:

- Reduserer tiden det tar å utvikle og lansere applikasjoner.
- Krever mindre teknisk vedlikehold av infrastruktur.
- Støtter samarbeid mellom utviklingsteam gjennom delte miljøer.

Eksempler på bruk:

- Utvikling av mobil- og webapplikasjoner.
- API-utvikling og integrasjoner.
- Applikasjoner som krever hurtig lansering eller eksperimentering.

SaaS (Software-as-a-Service)

Hva det er:

SaaS er en ferdig applikasjon levert over internett. Brukerne trenger bare å logge inn for å få tilgang – alt annet håndteres av leverandøren (IBM, 2024).

Ansvarsfordeling:

- Du har ansvar for:
 - Brukere: Administrere brukertilgang og roller.
 - Data: Sikre egne data i applikasjonen, inkludert samsvar med personvernregler som GDPR.
- Leverandøren har ansvar for:
 - Applikasjonen: Vedlikeholde og oppdatere programvaren.
 - Infrastruktur og sikkerhet: Holde servere, lagring og nettverk i drift, og beskytte dem mot angrep.
 - Tilgjengelighet: Sørge for at applikasjonen er tilgjengelig i henhold til avtalt SLA (Service Level Agreement).

Fordeler:

- Krever ingen teknisk administrasjon.
- Rask oppstart – logg inn og start arbeidet.
- Skalerbarhet – legg til flere brukere eller mer lagring etter behov.

Eksempler på bruk:

- E-posttjenester som Gmail eller Outlook.
- Samarbeidsverktøy som Slack og Microsoft Teams.
- CRM-verktøy som Salesforce og HubSpot.

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

4.

Skysikkerhet refererer til et sett av cybersikkerhet policies, beste praksiser, kontroller og teknologier som brukes for å beskytte applikasjoner, data og infrastruktur i skybaserte miljøer. Dette inkluderer tiltak for å sikre lagring og nettverksbeskyttelse mot interne og eksterne trusler, tilgangshåndtering, datastyring, samsvar, og katastrofegjenoppretting. Sikkerheten i skytjenester handler om å implementere retningslinjer, prosesser og teknologier for å beskytte data, støtte regelverkssamsvar, og gi kontroll over personvern, tilgang og autentisering for brukere og enheter.

De fleste skyleverandører følger en delt ansvarsmodell hvor leverandøren sikrer infrastrukturen som understøtter skyen, mens kunden har ansvar for å sikre applikasjoner, data og nettverk som kjører "i" skyen. Denne modellen varierer basert på tjenestemodellen—Infrastructure as a Service (IaaS), Platform as a Service (PaaS), eller Software as a Service (SaaS).

Kilde brukt: <https://cloud.google.com/learn/what-is-cloud-security?hl=nb>

Hva skjedde?

I juli 2019 annonserte Capital One en stor sikkerhetsbrist hvor en utenforstående person fikk

uautorisert tilgang til personopplysninger om ca. 100 millioner kunder i USA og 6 millioner i Canada. Angriperen utnyttet en feilkonfigurert AWS S3-lagringsbøtte som var en del av Capital Ones skyløsning.

Konsekvenser:

Den eksponerte informasjonen inkluderte navn, adresser, telefonnumre, e-postadresser, fødselsdatoer og selvrapportert inntekt fra søknader om kredittkort mellom 2005 og 2019. I tillegg ble følgende sensitive data kompromittert:

- 140,000 Social Security-numre (USA).
- 80,000 tilknyttede bankkontonumre.
- 1 million Social Insurance-numre (Canada).

Heldigvis ble ingen kredittkortnumre eller innloggingsinformasjon eksponert, og FBI fanget gjerningspersonen, som hadde bakgrunn som tidligere Amazon-ansatt.

Hvordan håndterte Capital One det?

Etter hendelsen:

- Feilen ble umiddelbart rettet, og føderale myndigheter ble involvert.
- Gjerningspersonen ble arrestert og dataene gjenopprettet.
- Capital One iverksatte tiltak for å varsle berørte kunder og tilby gratis kredittovervåkning og identitetsbeskyttelse.

Lærdom fra hendelsen:

Denne hendelsen illustrerer hvor viktig det er å sikre skytjenester mot feilkonfigurasjoner. De viktigste lærdommene inkluderer:

- Streng tilgangskontroller: Sørg for at sensitive data ikke er offentlig tilgjengelige gjennom feilkonfigurering.
- Overvåkning og revisjon: Bruk automatiserte verktøy for å overvåke og identifisere uautoriserte endringer eller feil i konfigurasjonen.
- Opplæring og bevissthet: Sikre at ansatte med tilgang til skyløsninger har tilstrekkelig opplæring i beste praksis.

Skyens delt ansvar:

Denne hendelsen understreker behovet for en robust forståelse av delt ansvar i skysikkerhet. Mens AWS sikret den underliggende infrastrukturen, lå ansvaret for riktig konfigurering hos Capital One.

Ved å ta lærdom fra hendelser som Capital One kan organisasjoner styrke sine skysikkerhetsstrategier og forhindre fremtidige sikkerhetsbrister.

Kilder brukt: <https://www.capitalone.com/digital/facts2019/>

5.

Cloud-native applikasjoner representerer en ny tilnærming til hvordan programvare bygges og leveres, og utnytter de unike mulighetene som skyen tilbyr. I motsetning til tradisjonelle monolittiske applikasjoner, som ofte blir komplekse og vanskelig å vedlikeholde, brytes cloud-native applikasjoner ned i små, løst koblede mikrotjenester. Dette gjør det mulig for team å jobbe mer effektivt, oppdatere funksjonalitet raskere, og skalere deler av systemet uavhengig av hverandre.

En av de viktigste grunnpilarene i cloud-native arkitektur er bruk av containere og orkestreringsplattformer som Kubernetes. Containere muliggjør portabilitet og fleksibilitet ved å pakke inn applikasjoner med alle nødvendige avhengigheter, slik at de kan kjøre i ethvert miljø. Kubernetes gir oversikt og kontroll over disse containerne, sørger for lastbalansering, og reparerer feil automatisk, noe som forbedrer påliteligheten og tilgjengeligheten til applikasjonen.

DevOps-metodologi og CI/CD-pipelines spiller også en sentral rolle i cloud-native utvikling. Ved å automatisere bygging, testing og distribusjon av kode, kan team levere endringer raskt og pålitelig, uten behov for manuelle prosesser. Dette bidrar til hyppigere oppdateringer og raskere innovasjon.

Cloud-native tjenester inkluderer også API-er som muliggjør kommunikasjon mellom mikrotjenester, samt skybaserte løsninger som IaaS, PaaS og SaaS. Disse tjenestene gir et fundament for å bygge og kjøre applikasjoner som er skalerbare, robuste, og kostnadseffektive.

En av de største fordelene med cloud-native applikasjoner er skalerbarhet. Dynamisk ressursallokering lar applikasjoner tilpasse seg endringer i etterspørsel, noe som sikrer optimal ytelse til enhver tid. Dette, kombinert med muligheten til å kjøre applikasjoner nesten hvor som helst, gjør cloud-native til et ideelt valg for moderne virksomheter.

Til tross for fordelene kommer cloud-native med utfordringer, som behovet for å håndtere distribuerte systemer og mange bevegelige deler. Organisasjoner må også overkomme motstand mot kulturelle endringer og bygge opp den teknologiske kompetansen som kreves for å implementere komplekse teknologistakker. Likevel, med riktig strategi og ekspertise, kan organisasjoner utnytte det fulle potensialet av cloud-native for å oppnå økt innovasjon og pålitelighet.

Kilder brukt: <https://cloud.google.com/learn/what-is-cloud-native?hl=nb>

6.

Migreringsstrategier for skyen beskriver ulike tilnærminger for å flytte arbeidsmengder til skyen. Det finnes syv hovedstrategier, ofte referert til som de "7 R-ene": Retire, Retain, Rehost, Relocate, Repurchase, Replatform, og Refactor. Disse strategiene tilpasses ulike behov, avhengig av applikasjonens kompleksitet, krav og forretningsmål.

Retire innebærer å avvikle applikasjoner som ikke lenger har verdi for virksomheten. Dette kan inkludere applikasjoner med lav ressursbruk, eller som ikke har blitt brukt over en lengre periode. Ved å pensjonere slike applikasjoner kan man redusere kostnader, sikkerhetsrisiko og administrasjonsbyrden knyttet til vedlikehold av utdatert teknologi.

Retain brukes når applikasjoner må forbli i den eksisterende infrastrukturen. Dette kan være nødvendig på grunn av sikkerhets- og samsvarsregler, avhengigheter til andre applikasjoner, eller fordi applikasjonen nylig har gjennomgått oppgraderinger som gjør migrering unødvendig på kort sikt. Noen ganger kan det også være hensiktsmessig å beholde applikasjoner inntil en SaaS-versjon blir tilgjengelig fra leverandøren.

Rehost, også kjent som "lift and shift", innebærer å flytte applikasjoner til skyen uten å gjøre noen endringer. Denne strategien er enkel og rask og brukes ofte i store migreringsprosjekter. Ved å flytte applikasjonene direkte til skyen, minimeres nedetid og forstyrrelser i driften. Når applikasjonen er i skyen, blir det enklere å optimalisere og modernisere i etterkant.

Relocate handler om å flytte applikasjoner til en skybasert versjon av deres nåværende plattform, eller til et annet nettverk, region eller konto i skyen. Dette er en rask metode for å migrere, da applikasjonens arkitektur forblir uendret. Relokering krever ikke ny maskinvare eller endringer i applikasjonen.

Repurchase, ofte kalt "drop and shop", innebærer å erstatte eksisterende applikasjoner med skybaserte løsninger, som for eksempel SaaS. Denne strategien reduserer kostnader knyttet til vedlikehold, infrastruktur og lisensiering, samtidig som den gir tilgang til nye funksjoner og større fleksibilitet. For eksempel kan en spesialutviklet applikasjon byttes ut med en standard skybasert løsning.

Replatform, også kjent som "lift, tinker, and shift", innebærer å flytte applikasjoner til skyen med mindre justeringer for å dra nytte av skytjenester, redusere kostnader eller forbedre driftseffektiviteten. Dette kan inkludere å flytte databaser til administrerte tjenester eller modernisere applikasjoner ved å konvertere dem til containere.

Refactor innebærer å redesigne applikasjoner for å utnytte skybaserte funksjoner fullt ut. Dette er en kompleks strategi som ofte brukes når applikasjoner har betydelige begrensninger, eller når det er behov for økt skalerbarhet og fleksibilitet. Refaktorisering kan inkludere oppdeling av monolittiske applikasjoner til mikrotjenester eller omorganisering av databaser for å møte spesifikke krav til samsvar og sikkerhet.

Disse strategiene gir organisasjoner fleksibilitet til å velge tilnærmingen som best passer deres behov, samtidig som de oppnår fordelene med migrering til skyen, som økt skalerbarhet, sikkerhet og kostnadseffektivitet.

7.

En Web Application Firewall (WAF) er en sikkerhetsløsning som beskytter webapplikasjoner mot ulike typer angrep ved å overvåke, filtrere og blokkere HTTP- og HTTPS-trafikk mellom en applikasjon og internett. Hovedmålet med en WAF er å beskytte webapplikasjoner mot kjente sårbarheter, som for eksempel SQL-injeksjoner, Cross-Site Scripting (XSS), og andre angrep som retter seg mot applikasjonens funksjonalitet eller data.

Ved å analysere trafikken på applikasjonsnivå (Layer 7) kan en WAF identifisere mistenkelig atferd og beskytte applikasjonen mot målrettede angrep. Dette gjøres ved bruk av forhåndsdefinerte regler for å oppdage spesifikke angrepsmønstre, eller ved hjelp av tilpassede regler som er konfigurert for å møte en organisasjons spesifikke behov.

En WAF brukes ofte for å sikre tilgjengeligheten og sikkerheten til kritiske applikasjoner. Den kan hindre overbelastning av serverressurser ved å filtrere ut ondsinnede forespørsler, redusere risikoen for datalekkasjer, og sikre samsvar med sikkerhetsstandarter. Spesielt i distribuerte miljøer, som hybrid- eller multi-skyarkitekturer, gir en WAF en enhetlig tilnærming til sikkerhet, selv når applikasjoner er distribuert på tvers av ulike infrastrukturer eller offentlige nettverk.

Ved å bruke en løsning som Google Cloud Armor, som inkluderer WAF-funksjoner og DDoS-beskyttelse, kan virksomheter dra nytte av avansert trafikkfiltrering og skalering på tvers av Googles globale nettverksinfrastruktur. Dette gir en kraftig beskyttelse mot både kjente sårbarheter og avanserte angrep, samtidig som det opprettholder applikasjonens ytelse og tilgjengelighet.

Kilder brukt: <https://cloud.google.com/blog/products/identity-security/new-waf-capabilities-in-cloud-armor>

8.

DevSecOps, som står for utvikling, sikkerhet og drift, er en tilnærming som integrerer sikkerhet i alle faser av programvareutviklingssyklusen. Målet er å redusere risikoen for å slippe kode med sårbarheter ved å gjøre sikkerhet til et felles ansvar, fremfor å behandle det som en ettertanke. Dette oppnås gjennom samarbeid, automatisering og tydelige prosesser, der sikkerhet diskuteres og testes allerede fra planleggingsfasen. I motsetning til tradisjonelle metoder, hvor utvikling, testing og sikkerhet er separate steg og ofte utføres sekvensielt, fokuserer DevSecOps på å identifisere og løse sikkerhetsproblemer kontinuerlig. Dette gir raskere leveranser av programvare med høyere kvalitet og færre risikoer. Tradisjonelle tilnærminger har ofte lange utviklingssykluser der sikkerhet legges til helt på slutten, noe som kan føre til kostbare og komplekse reparasjoner. DevSecOps, derimot, bygger sikkerhet inn i utviklingsprosessen fra starten av og benytter automatiserte verktøy for kontinuerlig testing, integrasjon og levering, noe som gjør det lettere å oppdage og rette feil tidlig. For skybaserte applikasjoner, som ofte er bygget for skalerbarhet og motstandsdyktighet, er DevSecOps spesielt godt egnet fordi det kombinerer prinsippene for sikkerhet og rask utvikling i et miljø som ofte er dynamisk og komplekst.

Kilder brukt: <https://www.microsoft.com/en-gb/security/business/security-101/what-is-devsecops#:~:text=DevSecOps%2C%20which%20stands%20for%20development,releasing%20code%20with%20security%20vulnerabilities>.

9.

For å bruke skytjenester som Azure, AWS eller GCP på en sikker måte, er det essensielt å forstå flere nøkkelområder relatert til sikkerhet og arkitektur. En grunnleggende forståelse av den delte ansvarsmodellen er avgjørende. Denne modellen fastsetter at skyleverandøren er ansvarlig for å sikre infrastrukturen, mens kunden har ansvar for å beskytte ressursene som er plassert i skyen, som applikasjoner, data og konfigurasjoner. Dette rammeverket krever en klar forståelse av hvor leverandørens ansvar slutter og kundens begynner.

Identitets- og tilgangsstyring (IAM) er en sentral del av sikkerheten i skyen. Dette innebærer å administrere brukere, roller og tillatelser med prinsippet om minst privilegium, hvor man kun gir de nødvendige tilgangene. Multifaktorautentisering (MFA) bør aktiveres for alle brukere, spesielt administratorer, for å sikre at tilgangsrettigheter ikke blir kompromittert. Nettverkssikkerhet er også kritisk, og dette oppnås ved å bruke virtuelle private nettverk (VPC-er), subnett og brannmurregler for å segmentere nettverket og isolere følsomme arbeidsbelastninger.

Data i skyen må beskyttes gjennom kryptering både i hviletilstand og under transport, og dette bør støttes av en pålitelig nøkkelhåndteringstjeneste. I tillegg bør backupstrategier og katastrofegjenopprettingsplaner være på plass for å beskytte data mot tap. Overvåking og logging er essensielt for å oppdage mistenkelig aktivitet. Tjenester som Azure Monitor, AWS CloudTrail og GCP Cloud Logging gir innsikt i hvem som har tilgang til ressurser og hva som skjer i miljøet.

Konfigurasjonsstyring spiller en viktig rolle i å sikre at ressursene er riktig konfigurert. Verktøy som AWS Config, Azure Security Center og GCP Security Command Center kan brukes til å oppdage og rette opp feilkonfigurasjoner. Automatisering gjennom Infrastructure as Code (IaC) som Terraform og CloudFormation gir konsistente distribusjoner og gjør det enklere å opprettholde sikkerhetspolicyer. I tillegg er det viktig å ha en tydelig plan for hendelseshåndtering, inkludert bruk av skybaserte verktøy for analyse og risikobegrensning ved sikkerhetshendelser.

Best praksis inkluderer å bruke private endepunkter og VPN-er for sikre tilkoblinger, implementere brannmurer for webapplikasjoner (WAF) for å beskytte mot vanlige angrep, og overvåke kostnader for å redusere angrepsflater. Et nulltillitsprinsipp bør også innføres, hvor

all trafikk behandles som potensiell risiko inntil det er verifisert. Videre kan skyens fleksibilitet utnyttes gjennom autoskalering og kontinuerlig etterlevelse av sikkerhetsstandarder som CIS Benchmarks eller NIST-rammeverket.

Skyleverandørene tilbyr omfattende arkitekturguider for å bygge sikre og robuste løsninger. For eksempel har Microsoft Azure sin Well-Architected Framework, AWS tilbyr sin Well-Architected Framework, og GCP har sitt Cloud Architecture Framework. Disse ressursene gir konkrete anbefalinger og eksempler på hvordan man kan utnytte skytjenester på en sikker og effektiv måte. Gjennom kontinuerlig læring og implementering av disse prinsippene kan man sikre skybaserte løsninger som både er pålitelige og motstandsdyktige.

10.

Til tross for skyens rolle som ryggraden i moderne teknologi som kunstig intelligens, maskinlæring og IoT, bringer den også med seg betydelige utfordringer som kan påvirke organisasjoner negativt. Nedenfor utforsker vi noen av de mest fremtredende ulempene ved skytjenester og deres potensielle konsekvenser.

1. Driftsstans og avhengighet av internett

En av de største utfordringene med skytjenester er avhengigheten av en stabil internettforbindelse. Nedenfor kan føre til store forstyrrelser og økonomiske tap, spesielt for virksomheter som er sterkt avhengige av kontinuerlig tilgang til sine data og systemer. Uten internett er skyressurser utilgjengelige, noe som kan redusere produktivitet og operasjonell effektivitet.

2. Sikkerhet og personvern

Med 82 % av alle datainnbrudd som involverer data lagret i skyen, står sikkerhet som en stor bekymring. Sensitiv informasjon i skyen kan være et attraktivt mål for cyberangrep. Selv med tiltak som multifaktorautentisering og tilgangskontroll, forblir risikoen for uautorisert tilgang betydelig. Dette stiller høyere krav til både tjenesteleverandører og brukere når det gjelder datasikring.

3. Begrenset kontroll og fleksibilitet

Brukere har mindre kontroll over infrastrukturen når de benytter skytjenester. Leverandører setter ofte teknologiske begrensninger som kan gjøre det vanskelig å tilpasse tjenester til spesifikke behov. Dette kan påvirke hvordan organisasjoner tilpasser og skalerer applikasjonene sine.

4. Vendor lock-in

Skyleverandører benytter ofte proprietære teknologier som kan gjøre det utfordrende å bytte leverandør. Å migrere data og systemer fra én plattform til en annen kan innebære betydelige kostnader og tekniske hindringer.

5. Kostnadsutfordringer og båndbreddeproblemer

Selv om skyen kan virke kostnadseffektiv, kan økt bruk føre til uforutsette kostnader. Høy båndbreddeforbruk kan resultere i tilleggskostnader og redusert ytelse. For organisasjoner med omfattende dataflyt kan dette bli en økonomisk byrde.

6. Varierende ytelse og tekniske problemer

Skyens ytelse kan variere basert på arbeidsmengden og tjenesteleverandørens kapasitet. I tillegg kan tekniske problemer i skyen være utfordrende å diagnostisere og løse, noe som kan forårsake forsinkelser i operasjonene.

7. Økt angrepsflate og datasikkerhetsrisiko

Skyens tilgjengelighet via internett gjør den mer utsatt for cyberangrep. Kombinert med mangelfull støtte fra noen leverandører, kan det være vanskelig å respondere raskt på trusler eller problemer.

Fordeler ved å unngå skytjenester

Det er tilfeller der det kan være fordelaktig å unngå skytjenester. Lokale systemer gir større kontroll over data og infrastruktur, reduserer avhengigheten av internett, og eliminerer risikoen for vendor lock-in. For spesifikke bruksområder med høye krav til sikkerhet og ytelse, kan lokal infrastruktur være et bedre valg.

Selv om skytjenester gir fleksibilitet og skalerbarhet, må organisasjoner nøye veie fordelene mot disse ulempene og vurdere sine spesifikke behov før de implementerer skybaserte løsninger.

Kilder brukt: <https://www.cloudwards.net/disadvantages-of-cloud-computing/>

Bibliografi

Cloudflare. (2024). *How does the cloud work?* Cloudflare Learning Center.
<https://www.cloudflare.com/learning/what-is-cloudflare/>

Google. (2024). Google cloud Pricing.
<https://cloud.google.com/pricing?hl=nb>

Google. (2024). PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different?
<https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=nb>

IBM. (2024). What are the benefits of cloud computing?
<https://www.ibm.com/topics/cloud-computing-benefits>

IBM. (2024). What are IaaS, PaaS and SaaS?
<https://www.ibm.com/topics/iaas-paas-saas>