NTNU
Norwegian University of
Science and Technology

Automatisert utplassering av containere i en dynamisk infrastruktur

Forfattere
Jostein Bergslien
Brage Veiseth Kleven
Daniel Monsen

Bachelor i IT-drift og informasjonssikkerhet
20 ECTS
Institutt for informasjonssikkerhet og kommunikasjonsteknologi
Norges teknisk-naturvitenskapelige universitet,

20.05.2019

Veileder Erik Hjelmås

Sammendrag av Bacheloroppgaven

Tittel:	**Automatisert utplassering av containere i en dynamisk infrastruktur**
Oppgave no.	28
Dato:	20.05.2019
Deltakere:	Jostein Bergslien Brage Veiseth Kleven Daniel Monsen
Veiledere:	Erik Hjelmås
Oppdragsgiver:	Escio
Kontaktperson:	Terje Krogstad, terje.krogstad@escio.no, 95972382
Nøkkelord:	Automatisering, Infrastruktur, Serverløs, AWS, CI/CD, DevOps
Antall sider:	57
Antall vedlegg:	6
Tilgjengelighet:	Åpen

Sammendrag:

Escio er en lokal bedrift i Gjøvik som har fått behov for å strømlinjeforme sine rutiner for utplassering av container-applikasjoner i Amazon Web Services. Til dette ønsker de å få utviklet en prototype bestående av tjenester eller verktøy som støtter automatisering og orkestrering. Prosjektgruppens problemstilling ble derfor å finne det de mente var den beste løsningen basert på Escio sine behov og krav. Det ble gjort en undersøkelse av aktuelle kandidater av verktøy og tjenester innenfor sine respektive bruksområder. Undersøkelsen resulterte i en selvstendig rapport med en anbefaling av egnede tjenester. Basert på tjenestene anbefalt, ble det utviklet en prototype for automatisert utplassering og en støttet orkestrert infrastruktur i henhold til problemstillingen. Prototypen består av tjenestene Bitbucket, CloudFormation, Elastic Container Service i sammen med den serverløse tjenesten Fargate. Prosjektgruppen konkluderte at å abstraktere bort serverdrift til Amazon Web Services vil kunne gi oppdragsgiver fordeler.

Summary of Graduate Project

Title:	**Automated deployment of containers in a dynamic infrastructure**
Project no.	28
Date:	20.05.2019
Authors:	Jostein Bergslien Brage Veiseth Kleven Daniel Monsen
Supervisor:	Erik Hjelmås
Employer:	Escio
Contact Person:	Terje Krogstad, terje.krogstad@escio.no, 95972382
Keywords:	Automation, Infrastructure, Serverless, AWS, CI/CD, DevOps
Pages:	57
Attachments:	6
Availability:	Open

Abstract:

Escio is a local company in Gjøvik that has a need to streamline its routines for deploying their container applications in Amazon Web Services. They want a prototype developed consisting of services or tools that support automation and orchestration. The project group's research issue was to find what they thought was the best solution based on Escio's needs and requirements. A survey was made including relevant candidates for tools and services within their respective areas of use. The survey resulted in an independent report with a recommendation for suitable services. Based on the services recommended, a prototype was developed for automated deployment and a supported orchestrated infrastructure according to the problem. The prototype consists of the services of Bitbucket, CloudFormation and Elastic Container Service in conjunction with the serverless service Fargate. The project team concluded that abstracting server operations to Amazon Web Services could bring benefits to Escio.

Forord

Vi ønsker å takke oppdragsgiver Escio for oppgaven. En spesiell takk rettes til kontaktperson Terje Krogstad for godt samarbeid. Vi ønsker også å takke Escio ansatt Aleksander Steen for god hjelp. Til slutt ønsker vi å takke vår veileder førsteamanuensis Erik Hjelmås ved NTNU, Gjøvik, for gode råd og tilbakemeldinger.

Innhold

Figurer

Tabeller

Listings

1 Innledning

1.1 Bakgrunn

Informasjonsteknologi hjelper organisasjoner og dens interessenter å nå sine mål. Når systemene fungerer etter hensikt produserer de verdi, enten direkte eller indirekte. Med dette knyttes IT-drift tett sammen med organisasjonens verdiskapning. Escio er et lokalt selskap i Gjøvik-regionen og tilbyr sine kunder utvikling av innovative digitale løsninger. Enten i form av tjenestedesign og rådgivning, utvikling av applikasjoner og tjenester, eller utleie av utviklerteam. Selskapet har tatt i bruk skytjenester for å sette sine applikasjoner og tjenester ut i drift for kundene. I denne sammenheng har selskapet fått behov for å automatisere sin praksis vedrørende utrulling av ny kode ut i driftsmiljøet dem disponerer i skytjenesten Amazon Web Services.

1.2 Oppgaven

1.2.1 Problemområde

Skytjenestenes fremmars på slutten av 2000-tallet endret arbeidsmetodikken fra konvensjonell serverdrift, og ledet frem til at det oppsto problemer mellom programvareutviklere og driftsavdelinger. Med tiden førte dette til at partene satte seg ned for å finne løsninger på problemene som oppsto, og samarbeidet resulterte i “The DevOps movement“. DevOps [1] kan beskrives som et sett av praksiser som automatiserer prosessene mellom utviklere og driftavdelinger i den hensikt at tjenester kan bygges, testes og settes ut i produksjonsmiljøet raskere og mer pålitelig. Målet med med DevOps-samarbeidet er å forkorte systemutviklingslivssyklusen samtidig som man leverer nye funksjoner, utfører vedlikehold og oppdaterer tjenesten i tråd med organisasjonens mål.

De fleste organisasjoner i dag har tatt i bruk IT-automatiseringsverktøy og dynamiske infrastrukturplattformer i form av private eller offentlige skytjenester. Som følge av dette har programmerbare infrastrukturer fått en nøkkelrolle som beste praksis i utførelsen av DevOps [2]. Automatisering tar sikte på å redusere feil sammenlignet med manuelle oppsett, og tillater at programvare kan settes ut i produksjon med større fleksibilitet, mindre nedetid og til en lavere kostnad for organisasjoner enn tidligere.

1.2.2 Oppgavebeskrivelse

Oppdragsgiver har sett et behov for å strømlinjeforme sin arbeidsmetodikk for utrulling av container-baserte applikasjoner ut i en orkestrert infrastruktur i AWS. Prosjektgruppen har fått i oppgave å utføre følgende oppgaver:

1. Lage en selvstendig rapport med beskrivelse av systemer og tjenester som bygger opp om en fullautomatisert flyt.
2. Utvikle en prototype basert på en av Escio sine applikasjonskodebaser som viser en fullautomatisk flyt fra kode til en skalerbar applikasjon i AWS.

Sammendraget av selvstendig rapport er gjengitt i kapittel 3, og den fulle besvarelsen av punkt 1 ligger i sin helhet i vedlegg F. Følgende rapport besvarer punkt to i punktlisten. Rapporten innholder en beskrivelse av implementerte tjenester i prototypen og hvordan disse har blitt konfigurert. Det har blitt gjort modifiseringer i problemstillingen siden prosjektplanen ble opprettet, følgende problemstilling vil bli besvart i konklusjonen av denne rapporten:

Hvilken løsning kan tilfredsstille oppdragsgivers krav og behov for en automatisert utplassering av container-applikasjoner i en orkestrert infrastruktur i AWS?

1.2.3 Oppdragsgivers krav

Oppdragsgivers generelle krav til prototype er følgende:

- Container-applikasjoner skal kunne utplasseres automatisk i en dynamisk infrastruktur i AWS.
- Prototypeinfrastruktur skal være orkestrerert, samt skalere og lastbalansere container-applikasjoner basert på trafikk.

Prototypens funksjonelle, ikke-funksjonelle og sikkerhetsmessige krav blir omtalt ytterligere i kapittel 4 Kravspesifikasjon.

1.2.4 Avgrensning

For å kunne fullføre oppgaven innenfor tidsfristen har det blitt gjort noen avgrensinger.

- Tjenestene Bitbucket og AWS var forhåndsbestemt, det ble derfor ikke drøftet andre løsninger innenfor sine respektive områder.
- Kun oppdragsgivers vedlagte stateless-applikasjoner ble benyttet for å teste prototypen.
- Det ble kun foretatt testing om applikasjonen klarte å rulle ut automatisk til infrastrukturen i AWS.
- Det ble ikke laget tester for applikasjonenes funksjonalitet i forbindelse med CI/CD-pipeline.
- Det har ikke blitt foretatt kostnadsberegninger for prototype.

1.3 Formål

Rapporten skal forhåpentligvis gi Escio en mulig prototypeløsning for CI/CD, samt en tilpasset infrastruktur til drift av container-applikasjoner i AWS. Målet med rapporten er å forbedre dagens praksis relatert til utrulling av de applikasjoner og tjenester selskapet utvikler.

1.4 Målgruppe

Oppdragsgiver er rapportens hovedsakelige målgruppe. Rapporten kan også være til hjelp for andre bedrifter eller personer som er i en lignende situasjon, eller som ønsker lære mer om ulike tjenester i AWS, og hvordan de kan implementeres.

1.5 Prosjektgruppens bakgrunn

Gruppemedlemene og forfattere av denne rapport går det samme studieprogrammet IT-drift og informasjonssikkerhet, ved NTNU i Gjøvik. Alle i gruppen har hatt de samme forutsetningene med tanke på tidligere fag og studieretning.

1.6 Roller

Oppdragsgiver for prosjektet er bedriften Escio, med Terje Krogstad som kontaktperson. Prosjektgruppens veileder har vært Erik Hjelmås, førsteamanuesis for Norges teknisk-naturvitenskapelige universitet i Gjøvik. Gruppeleder for prosjektet har vært Jostein Bergslien. Prosjektet har brukt utviklingsmodellen SCRUM, hvor Brage V. Kleven har hatt rollen som Scrum-Master.

1.7 Kapittelinndeling

1 **Innledning** - Gir en innføring i oppgavens omfang.
2 **Teori** - Beskriver teorien som er styrende for rapportens sammenheng.
3 **Sammendrag i selvstendig rapport** - Sammendraget i selvstendig levert rapport gjengis.
4 **Kravspesifikasjon** - Tar for seg funksjonelle-, kvalitetsmessige- og sikkerhetsmessige krav og tilhørende use- og misuse cases.
5 **Dagens flyt og beste praksis** - Tar for seg bedriftens praksis og knytter den opp mot fagområdets beste praksis.
6 **Prototype** - En generell beskrivelse av prototypen og dens funksjoner.
7 **Implementasjon av prototype** - Kapittelet forklarer hvordan og hvorfor de forskjellige tjenestene er impementert.
8 **Sikkerhet** - Dette kapittelet beskriver Identity and access management.
9 **Avslutning** - Avslutningsvis har det blitt drøftet forskjellige aspekter ved prototypen, videre arbeid og en evaluering av arbeidet gjennomført. Rapporten avsluttes med konklusjon.

I **Vedlegg** finnes det vedlagte dokumenter relatert til prosjektet. Blant annet ordliste A, statistikk B og selvstendig rapport F.

2 Teori

2.1 Automatisering

Ifølge det Store Norske Leksikon, defineres automatisering på følgende vis:

"*Automatisering er teknikken å få systemer til å fungere uten, eller med liten grad av menneskelig medvirkning [3]*".

Denne definisjonen er universell og var viktig i forhold til oppgavebeskrivelsen. Formålet med å automatisere en CI/CD-pipeline for Escio er å få de forskjellige systemene til å kjøre uten eller med liten grad av menneskelig medvirkining, med tanke på å teste, bygge og kjøre ut applikasjonen til produksjon.

2.2 Programmerbar infrastruktur

Programmerbar infrastruktur har som formål med å forenkle IT operasjoner. For utvikling av systemer og tjenester brukes det en del prinsipper definert i boken *Infrastructure as code* av *Kief Morris* [2]. Boken blir hyppig referert i fagmiljøet, blant annet av AWS [4].

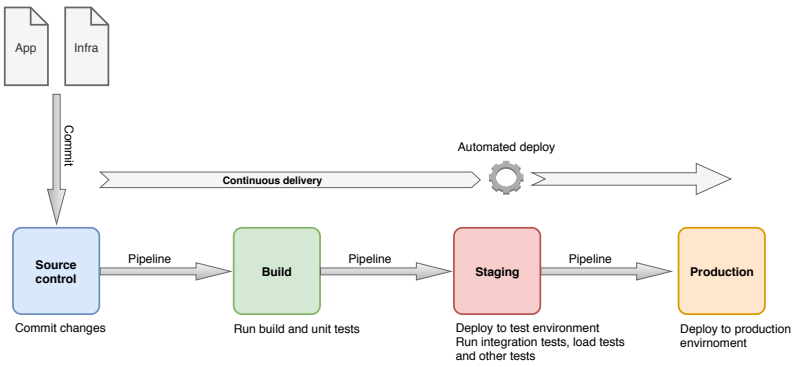
Programmerbar infrastruktur er prinsipper inspirert av programvareutvikling for å gjøre endringer i infrastrukturen. Endringer i infrastrukturen blir definert i konigurasjonfiler, som vil hjelpe med å holde alle systemer konsistente. Ved å holde elementer i infrastrukturen konsistente vil man ifølge Kief Morris effektivisere tiden på operasjoner som provisjonering, konfigurering, oppdatering og administrering av tjenester og systemer, samt oppdage og løse problemer raskere.

Målet med å innføre konseptene i programmerbar infrastruktur i bedrifter eller organisasjoner er for å gjøre kontinuerlig endringer og forbedringer på infrastrukturen til en rutine. “Treat your servers like cattle, not pets“, er et populært uttrykk innen DevOps miljøet. Betydningen av uttrykket er å behandle alle systemer likt for å holde de konsistente, dette for å kunne reprodusere deler av infrastrukturer på en enkel og effektiv måte.

2.3 CI/CD

Continuous integration(CI) eller *kontinuerlig integrasjon* på norsk, er praksisen ved å reglemessig integrere og teste alle endringer gjort i et system ettersom de blir utviklet [2]. Målet med CI er å raskt avdekke feil, forbedre kvaliteten av programvare og redusere den tiden det tar å validere og utgi den.
Continuous delivery(CD) eller *kontinuerlig utrulling* på norsk, er praksisen hvor endringer i programvaren automatisk bygges, testes og klargjøres for produksjon [4]. Kontinuerlig leveranse ekspanderer omfanget av CI, ved at alle endringer i programvaren distribueres til et testmiljø, produksjonsmiljø eller begge.

En misforståelse når det kommer til CD er at hver endring som blir gjort blir rullet ut til produksjon automatisk. Dette heter *continuous deployment* eller *kontinuerlig utplassering* på norsk, og er ikke det samme som continuous delivery. Det er opp til bedriften om de har lyst til å innføre continuous deployment.
En *CI/CD-pipeline* er bygget opp av forskjellige steg, som alle er med på å kvalitetsikre applikasjonen. Figur 1 illustrerer en CI/CD-pipeline, hvor stegene er forklart under.



Figur 1: CI/CD-pipeline

- **Source control:** Utviklere pusher kode ut til et *repository*, der koden blir *merget* sammen.
- **Build:** Applikasjonen blir bygget. Enhetstester og statisk kodeanalyse blir utført.
- **Staging:** Applikasjonen blir distribuert til et testmiljø, hvor det blir utført integrasjonstester, ytelsestester og komponent-tester. Dette for å passe på at applikasjonen er klar til å bli rullet ut til produksjon.
- **Production:** Applikasjonen blir rullet ut til produksjonsmiljøet.

2.4 Provisjonering

Her benyttes Kief Morris sin tolkning av provisjonering:

"Med provisjonering menes det å gjøre et infrastrukturelement som en server eller en nettverksenhet klar til bruk [2]."

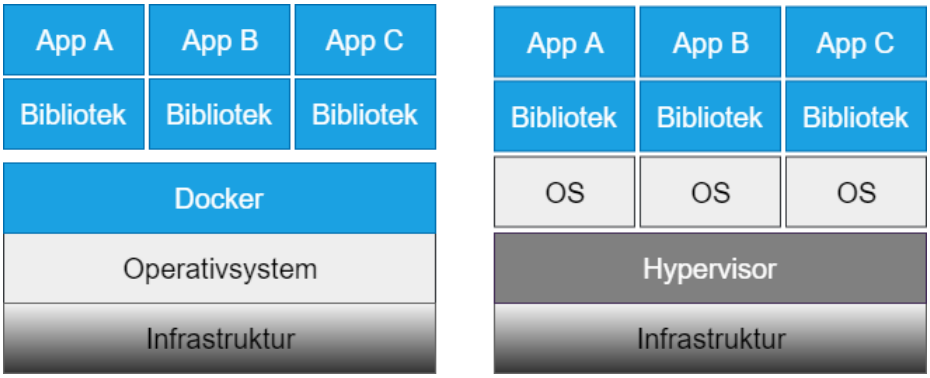
Avhengig av hvilket infrastrukturelement som er gitt, kan dette innebære:

- Tilordne ressurser til elementet.
- Opprette elementet.
- Installere programvare på elementet.
- Konfigurere elementet.
- Registrere elementet med infrastrukturtjenester.

Formålet med å automatisere provisjonering er å etablere en repeterbar, konsistent, selv-dokumentert og gjennomsiktig provisjoneringsprosess.

2.5 Container

En container er en eksekverbar fil pakket inn med alt av avhengigheter for å kunne kjøre en applikasjon. Container er bygget for å optimalisere ressursene ved at alle containere kun benytter de ressursene den har behov for [5]. For å lage, distribuere og kjøre containerbasere applikasjoner er det verktøyet Docker som har ansvaret for. På figur 2, kan man se arkitekturen bak containere og virtuelle maskiner.



Figur 2: Container og virtuell maskin

Forskjellen mellom en virtuell maskin og en container er at en container virtualiserer operativsystemet, mens en virtuell maskin virtualiserer maskinvaren. En container kjører på en fysisk server og dens operativsystem. En container får tildelt de ressursene fra operativsystemet som er nødvendig for å kunne kjøre. I motsettning til en virtuell maskin som har et helt eget operativsystem. En hypervisor virtualiserer maskinvaren og lager et operativsystem til hvert enkelt virtuelle maskin.

2.6 AWS

AWS står for Amazon Web Services, som er en skytjenesteleverandør som leverer datakapasitet, databaselagring og andre funksjonaliteter for å hjelpe bedrifter og organisasjoner til å skalere og bidra til en økonomisk vekst [6]. AWS tilbyr et bredt utvalg av tjenester som gruppen har benyttet for å sette opp prototypen. Disse tjenestene er kort beskrevet under.

2.6.1 CloudFormation

CloudFormation er en tjeneste som hjelper brukerne å administrere AWS ressurser [7]. Tjenesten provisjonerer og konfigurerer de spesifiserte ressursene ved hjelp av en eller flere konfigurasjonsfiler og støtter JSON og YAML format.

2.6.2 Elastic Container Service (ECS)

ECS er en rask og lett skalerbar container-orkestreringstjeneste som gjør det enkelt å kjøre, stoppe og administrere containere i en klynge. ECS kan brukes sammen med den serverløse tjenesten Fargate, eller med virtuelle maskiner i EC2 (Elastic Compute Cloud) [8]. Tjenesten brukes til å planlegge hvor containerne skal plasseres i klyngene basert på ressurskrav, isolasjons-policy og tilgjengelighetskrav. Tjenesten eliminerer behovet for at

brukeren selv må operere egen klyngestyring, konfigurasjonsstyringsystem eller skalering av styringsinfrastruktur.

2.6.3 Virtual Private Cloud (VPC)

Amazon Virtual Private Cloud er en tjeneste som deler opp nettverket inn i et virtuelt nettverk og er isolert fra andre virtuelle nettverk i AWS [9]. VPC ligner på et tradisjonelt nettverk hvor du opererer med ditt eget datasenter. Formålet med å sette opp en VPC er å dele opp nettverket inn i forskjellige subnet, dette er for å gjøre applikasjonen og infrastrukturen sikrere. I hvert subnet er det mulig å sette opp forskjellige sikkerhetsmekanismer.

2.6.4 Elastic Load Balancing (ELB)

Amazon Elastic Load Balancing er en tjeneste som har i oppgave å distribuere innkommende applikasjon- eller nettverks-trafikk på tvers av ressursene [10]. ELB kan eksempelvis fordele trafikken på flere EC2-instanser eller containere. Ved å benytte seg av lastbalansering vil applikasjonene oppnå bedre feiltoleranse og tilgjengelig, uavhengig av trafikk.

2.6.5 Elastic Compute Cloud (EC2)

Elastic Compute Cloud er en tjeneste i AWS som leverer dataressurser, kalt server-instanser som i hovedsak er virtuelle maskiner [11]. AWS gir brukeren muligheten til å skalere ressurser som minne, CPU og lagringsplass etter behov, samt fjerne eller kjøre opp nye virtuelle maskiner. Å benytte seg av EC2 vil brukeren fortsatt ha muligheten til å utføre oppgaver som å konfigurere sikkerhet, nettverk og lagringskapasitet. Ved bruk av EC2 vil behovet for å kjøpe hardware og ha fysisk kontroll over ressursene elimineres.

2.6.6 Fargate

Fargate er en *compute engine* for Amazon ECS [12]. Her kan brukere kjøre containere uten å måtte forholde seg til servere. Dette vil si at provisjonering, konfigurering og skalering av virtuelle maskiner ikke lengre er nødvendig for å kjøre containere. Med Fargate elimineres all interaksjon med servere, og gjør at det virtuelle miljøet kan sømløst skalere.

2.6.7 CloudWatch

Amazon CloudWatch er en monitoreringstjeneste bygget for at utviklere, systemdriftere og annet IT-personell skal kunne opprettholde tjenestenes tilgjengelighet [13]. Dette innebærer logger, metrisk data og hendelser for alle AWS-ressurser, applikasjoner og tjenester. Tjenesten kan sette alarmer, visualisere logger, sammenligne metrisk data, automatisere oppgaver reaktivt og feilsøke problemer.

2.6.8 Command Line Interface (CLI)

Et viktig prinsipp i fagfeltet programmerbar infrastruktur handler om å kunne håndtere infrastruktur som programvare [14]. Dette betyr at at dynamiske infrastrukturer må ha noen viktige egenskaper for å kunne tilfredstille fagfeltets prinsipper. Disse egenskapene er definert i boken *Infrastructure as Code* av *Kief Morris* [2]. En dynamisk infrastrukturplattform skal være programmerbar, kunne benyttes ved behov og være selvbetjente. AWS tilfredsstiller alle disse egenskapene og tilbyr brukere et åpent-kildekodeverktøy for

å kommunisere med skytjenesten ved hjelp av terminalen. Verktøyet tillater brukeren eller tjenester til å kommunisere med AWS-tjenestene ved hjelp av kommandoer i et shell. Med disse kommandoene kan man ta i bruk all funksjonalitet i terminalen på lik linje som det grafiske brukergrensesnittet AWS Management Console tilbyr. AWS-CLI kan benyttes i ulike terminaler:

- **Linux shells** - Vanlige shell programmer som *bash*, *zsh* og *tsch* kan benyttes til å kjøre kommandoer i Linux, macOS eller Unix.
- **Windows command line** - På Windows kan kommandoer kjøres i PowerShell eller i CMD.
- **Remotely** - Det kan også kjøres kommandoer til instanser gjennom en remote terminal som PuTTY eller SSH, eller ved bruk av AWS System Manager.

I løpet av dette prosjektet har det blitt brukt AWS CLI kommandoer til å verifisere *templates*, endre kjørende ressurser, og rulle ut de forskjellige infrastrukturelementene i tjenesten CloudFormation.

2.6.9 AWS SDK

Amazon har en rekke Software Development Kits (SDK) for flere forskjellige programspråk [15]. Slike *kits* spiller en sentral rolle i utviklingen av automatisert programmerbar infrastruktur, ettersom at deler av CI/CD prosessen må eksekvere korrekte funksjoner uten menneskelig tilsyn [2].

2.7 Serverløs arkitektur

Serverløs er et begrep som ofte blir misforstått, det betyr ikke at ingen servere blir brukt. Tanken bak serverløse tjenester er at utviklere gir slipp på operasjonelle oppgaver relatert til drifting av servere. Driftsoppgaver som provisjonering, monitorering, vedlikehold og skalering overføres dermed til tjenesteleverandøren. Serverløs databehandling, eller ordet serverløs er en tilnærming for å kjøre ut tjenester på skyen hvor tjenesteleverandøren har full kontroll over serverne applikasjonene kjører på. Serverløs er et relativt nytt konsept og har vært i stor vekst de siste årene. Dette er fordi bedrifter og organisasjoner har begynt å fokusere mer på container-baserte applikasjoner og miktrotjenester. På figur 3 kan det sees populæriteten på søketermen “Serverless“ de siste årene. I 2015 var det året serverløse tjenester skiftet fokus til å handle mer om hendelsesdrevet programmering [16]. Dette betyr at et program ikke kjører før den blir trigget av en handling. *FaaS* [17] tjenesten AWS Lambda var den første plattformen som tilbydde en serverløs løsning og ble lansert i 2014 [18]. Lambda var en av de viktigste bidragsyterne for å fremme tilnærmingen serverløs [19] og senere fulgte andre leverandører som Google og Microsoft med på trenden. De lanserte henholdsvis Google Cloud Functions[1] og Microsoft Azure Functions[2].

Figur 3: Populæriteten på søketermen serverless (Google Trends)

[1] https://azure.microsoft.com/nb-no/services/functions/
[2] https://azure.microsoft.com/nb-no/services/functions/

3 Sammendrag av levert rapport

Rapporten har beskrevet skytjenestemodeller og relaterte AWS-tjenester. Det er i tillegg beskrevet og drøftet enkelte utvalgte verktøy fra andre produsenter i henhold til sine respektive bruksområder. Fellenevneren for disse er at de kan være ledd i å bygge oppe en automatisert flyt. Rapporten gir blant annet leseren en oversikt over aktuelle AWS-tjenester og hvor i kjeden disse kan benyttes i en CI/CD-pipeline. Verktøy for infrastrukturorkestrering, serverkonfigurasjon og containerorkestrering har blitt omtalt i hvert sitt kapittel.
Siden verktøyene omtalt i rapporten måtte oppfylle sine respektive krav for å bli beskrevet, resulterte dette også i at verktøyene i all hovedsak egner seg til formålene og bruksområdet. Det ble derfor vanskelige og kvantifisere disse på en tilfredstillende måte med den tid som sto til rådighet. På dette grunnlag ble det besluttet at verktøy og tjenesters kompatibilitet mot AWS, god dokumentasjon og størrelse på brukermiljø, samt enklest mulig operasjon for driftspersonell skulle vektes høyest.
Slutningene trukket for skytjenestene omtalt i rapporten er at AWS tilbyr godt egnede kandidater til å forenkle og automatisere driftsoppgaver, og kan med dette forbedre dagens flyt. Fordelen med å benytte AWS-tjenester er at bedriften i dag kjører majoriteten av sine applikasjoner i AWS. Tjenestene som tilbys er svært godt dokumentert og har bred støtte i AWS-miljøet. Det tilbys også omfattende kundeservice, dersom bedriften får behov for dette. Ulempen ved bruk av AWS-tjenestene er at bedriften knytter seg sterkere til én leverandør.
For orkestrering av infrastrukturer ble to verktøy beskrevet. HashiCorp Terraform, og AWS CloudFormation. Fordelen med CloudFormation er at dette verktøyet er optimalisert for AWS, og har best støtte blant relaterte tjenester innad i miljøet. Terraform er i motsetning et bedre alternativ dersom Escio på sikt ønsker å orkestrere ressurser på tvers av skytjenesteleverandører.
Escio benytter i dag Ansible til serverkonfigurasjon. Det ble sett på to alternativer, Chef og Puppet. Alle disse verktøyene er godt egnet til formålet, og oppfyller alle krav. Tjenesten AWS Fargate har blitt anbefalt, derfor er ikke et server-konfigurasjonsvertkøy nødvendig.
For container-orkestrerngsverktøy ble to verktøy beskrevet, Docker Swarm og Kubernetes. Av de to førstnevnte kandidatene, anbefales Docker Swarm. Dette fordi det er et enklere verktøy og betjene, har god dokumentasjon og et stort brukermiljø. For orkestrering av containere har likevel ECS blitt anbefalt, dette fordi tjenesten støtter Fargate og enkel å bruke.
Under konklusjon i denne rapport er det beskrevet en mulig løsning til en prototype for CI/CD. Denne prototypen baserer seg på bruk av Bitbucket pipelines, CloudFormation, ECS og Fargate.

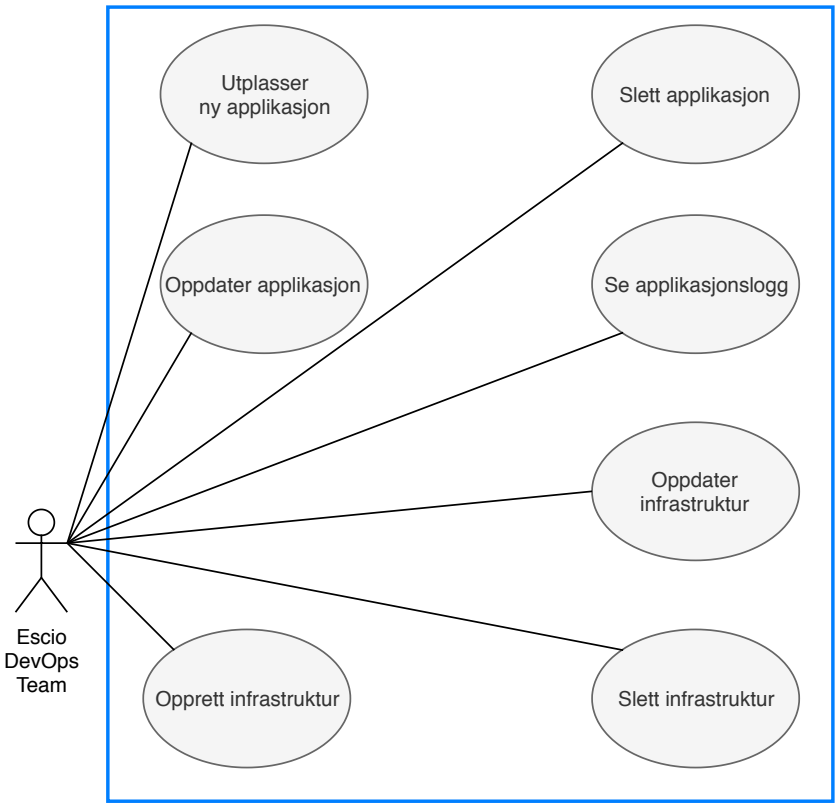
4 Kravspesifikasjon

Dette kapittelet tar for seg funksjonelle-, kvalitetsmessige-, og sikkerhetsmessige krav og tilhørende use cases og misuse cases.

4.1 Funksjonelle krav

I oppgavebeskrivelsen 1.2.2 beskrives prosjektoppgavens fokus. Det blir i dette kapittelet forsøkt å belyse de viktigste funksjonelle kravene og brukertilfellene for prototypen.

4.1.1 Use case



Figur 4: Use Case Diagram

Use Case	**Utplasser ny applikasjon**
Aktør	DevOps
Hensikt	Utplassering av ny applikasjon ut i produksjonsmiljøet i AWS.
Beskrivelse	Utvikler har en ny applikasjon som skal ut i infrastrukturen oppdragsgiver disponerer i AWS.
Forutsetninger	Applikasjonen må være ferdig utviklet, testet og godkjent i henhold til bedriftens interne rutiner. Det må opprettes en Dockerfile i rot mappen til gjeldene prosjekt. Tilhørende konfigurasjonsfil (Task definition) for applikasjonens ressurskrav må spesifiseres, i tillegg til pipeline-miljøvariabler i Bitbucket.
Ettervirkninger	Ny applikasjon er utplassert i produksjon for kunde. Kostnad for benyttede AWS-ressurser begynner å løpe.
Spesielle krav	Tilgang til aktuelt Bitbucket repository.
Normal hendelsesflyt	1. Utvikler merger/pusher til master. 2. CI/CD-pipeline eksekverer filen “bitbucket-pipeline.yaml“ automatisk.
Ulike feilsituasjoner	Feil i sikkerhet-token fører til at ikke pipeline kan kommunisere med AWS. Feil i Dockerfile fører til at applikasjon ikke blir bygget eller at containeren ikke klarer å kjøre i Fargate. Feil parametere i Task definition fører til automatisk rollback i AWS CloudFormation.

Tabell 1: Utplasser ny applikasjon

Use Case	**Oppdater applikasjon**
Aktør	DevOps
Hensikt	Oppdatering av en kjørende applikasjon i produksjonsmiljøet i AWS.
Beskrivelse	Utvikler har en ny versjon av en applikasjonen og ønsker å utplassere denne.
Forutsetninger	Applikasjonen som skal oppdateres må være ferdig utviklet, testet og godkjent i henhold til bedriftens interne rutiner. Tilhørende konfigurasjonsfil og Dockerfil justeres ved behov.
Ettervirkninger	Oppdatert applikasjon er utplassert i produksjon for kunde.
Spesielle krav	Tilgang til aktuell Bitbucket repository. Tidligere versjon av applikasjonen må eksistere i produksjon, ellers trigges Use Case “Utplasser ny applikasjon“ 1.
Normal hendelsesflyt	1. Utvikler merger/pusher til master. 2. CI/CD-pipeline sjekker om applikasjonen eksisterer i produkjonsmiljøet, og eksekverer filen “bitbucket-pipeline.yaml“ automatisk.
Ulike feilsituasjoner	Se feilsituasjon i tabell 1.

Tabell 2: Oppdater applikasjon

Use Case	**Se applikasjonslogg**
Aktør	DevOps
Hensikt	Se ønsket loggdata for applikasjon.
Beskrivelse	Utvikler ønsker å se en eller flere registrerte loggdata for en eller flere applikasjoner.
Forutsetninger	Spesifisert logging i dockerfilen.
Ettervirkninger	Utvikler kan se applikasjonens loggdata.
Spesielle krav	AWS-bruker med CloudWatch IAM rettigheter. En eksisterende 'LogGroup' må opprettes og denne gruppen spesifisert i Task Definition.
Normal hendelsesflyt	Utviklerer spesifiserer logging i Dcckerfile, og kan lese av loggen i tjeneten CloudWatch ved bruk av CLI-kommandoer eller ved hjelp av AWS sitt GUI.
Ulike feilsituasjoner	Spesifikasjonsfeil i Dockerfile.

Tabell 3: Se applikasjonslogg

Use Case	**Slett applikasjon**
Aktør	DevOps
Hensikt	Slette en applikasjon i produksjon i AWS.
Beskrivelse	Utvikler ønsker å slette en applikasjon fra produksjon.
Forutsetninger	Applikasjonen må eksistere. Kunde/bruker/system bør ikke bruke/være i avhengighetsforhold til applikasjonen som slettes.
Ettervirkninger	Applikasjonen kan ikke lenger brukes. Kostnader relatert til benyttede AWS-ressurser slutter å løpe.
Spesielle krav	AWS-bruker tilstrekkelig IAM rettigheter.
Normal hendelsesflyt	Utvikler bruker AWS-CLI eller GUI til å slette aktuell applikasjon.
Ulike feilsituasjoner	Dersom sikkerhetsfunksjonen "Termination protection"7.1.3 i tjenesten CloudFormation er aktivert, vil ikke applikasjonen slettes, og systemet gir tilbakemelding om dette.

Tabell 4: Slett applikasjon

Use Case	**Opprett infrastruktur**
Aktør	DevOps
Hensikt	Opprette en infrastruktur eller sett av AWS-ressurser som kan brukes til å drifte applikasjoner, eller støtte denne funksjonen.
Beskrivelse	Utvikler konfigurerer ønskede AWS-ressurser med det formål å opprette en infrastruktur som tilfredsstiller bedriftens mål.
Forutsetninger	Spesifisert konfigurasjonsfil eller filer.
Ettervirkninger	En ny infrastruktur i samsvar med spesifikasjon.
Spesielle krav	En AWS-bruker med administrerende IAM tilganger.
Normal hendelsesflyt	1. Validering av konfigurasjonsfiler via AWS sitt CLI/GUI. 2. Kjøre opp konfigurasjonsfiler til CloudFormation via AWS sitt CLI/GUI.
Ulike feilsituasjoner	Om det har blitt brukt ugyldige variabler, vil det trigge en automatisk tilbakerulling.

Tabell 5: Opprett infrastruktur

Use Case	**Oppdater infrastruktur**
Aktør	DevOps
Hensikt	Oppdatere infrastrukturen etter behov.
Beskrivelse	Oppdatering av infrastrukturen har til formål til å forbedre eller endre infrastrukturoppsettet i den hensikt å sikre ytelse, sikkerhet eller andre viktige egenskaper basert på bedriftens behov og målsetning.
Forutsetninger	Infrastruktur eksisterer. Utføre ønskede endringer i aktuell konfigurasjonsfil eller filer. Validere, teste og godkjenne oppdateringen før oppdateringen settes i drift.
Ettervirkninger	Oppdatert infrastruktur i samsvar med spesifikasjon.
Spesielle krav	En AWS-bruker med administrerende IAM tilganger.
Normal hendelsesflyt	1. Opprett Change Sets 7.1.1 for ressurser som skal oppdateres ved bruk av AWS-CLI / GUI i CloudFormation. 2. Oppdater konfigurasjonsfil eller filer. 3. Valider, test og godkjenn oppdateringen. 4. Gjennomfør oppdatering.
Ulike feilsituasjoner	Dersom ressurser er tilknyttet *stack level policies*, og/eller *resource level policies* kan dette medføre at de aktuelle ressurser ikke oppdateres.

Tabell 6: Oppdater infrastruktur

Use Case	**Slett infrastruktur**
Aktør	DevOps
Hensikt	Slette infrastruktur
Beskrivelse	Utvikler ønsker å slette infrastrukturen eller elementer i denne.
Forutsetninger	Det finnes en infrastruktur.
Ettervirkninger	Infrastrukturen er fjernet, og kostnader relatert ressursbruk slutter å løpe.
Spesielle krav	En AWS-bruker med administrerende IAM tilganger.
Normal hendelsesflyt	Utvikler bruker AWS CLI/GUI til å slette aktuell infrastruktur.
Ulike feilsituasjoner	Dersom sikkerhetsfunksjonen “Termination protection“ 7.1.3 i tjenesten CloudFormation er aktivert, vil ikke infrastrukturen slettes, og systemet gir tilbakemelding om dette.

Tabell 7: Slett infrastruktur

4.2 Kvalitetsmessige krav

I denne seksjonen blir det omtalt noen kvalitetsmessige krav for infrastrukturen generelt, og som ikke direkte omfatter de funksjonelle kravene.

4.2.1 Ikke-funksjonelle krav

Oppetid

Oppdragsgiver har ikke angitt krav til oppetid for skytjenesteleverandøren i sin oppgavebeskrivelse. Derfor blir det her henvist til AWS sin *Service Level Agreement (SLA)* [20]. For aktuelle tjenester benyttet i dette prosjektet gjelder en oppetid på 99.99%, basert på det AWS beskriver som kommersielt forsvarlig innsats fra deres side. Dersom oppetiden til benyttede tjenester er lavere enn dette, kan oppdragsgiver være kvalifisert til å mot-

ta *service credits* etter angitte satser som ikke gjengis her. For å motta *service credits* må bedriften åpne en sak i AWS Support Center.

Versjonskontroll og backup for konfigurasjonsfiler og relaterte data

Oppdragsgiver ønsker å beholde Bitbucket som versjonskontrollsystem, og er et krav som har blitt fulgt gjennom arbeidet med prosjektet. Det har derimot ikke blitt angitt noe system for backup av konfigurasjonsfiler eller andre relaterte data. Til backup anbefales AWS Simple Storage Service (S3), som tilbyr kryptering. Alternativt kan backup gjøres lokalt, etter bedriftens sikkerhetsrutiner.

Gjenopprettingstid for infrastruktur ved feil

Det har ikke blitt spesifisert noen spesielle krav relatert til tidsbegrensning for gjenoppretting av infrastrukturen som følge av system- eller brukerfeil. Basert på prosjektgruppens erfaringer ved opprettelse eller oppdatering av prototype-infrastruktur, har tidsbruken medgått til dette ligget mellom 5-10 minutter. Tid til gjenoppretting av applikasjoner driftet av prototype-infrastrukturen kommer i tillegg og vil variere etter kompleksitet og relasjoner til andre systemer. Det oppfordres på dette grunnlag å bruke ECS testmiljøklyngen *Development* 7.2.2 til dette formålet som ble opprettet i prototypen.

4.2.2 Krav til brukerstøtte og dokumentasjon

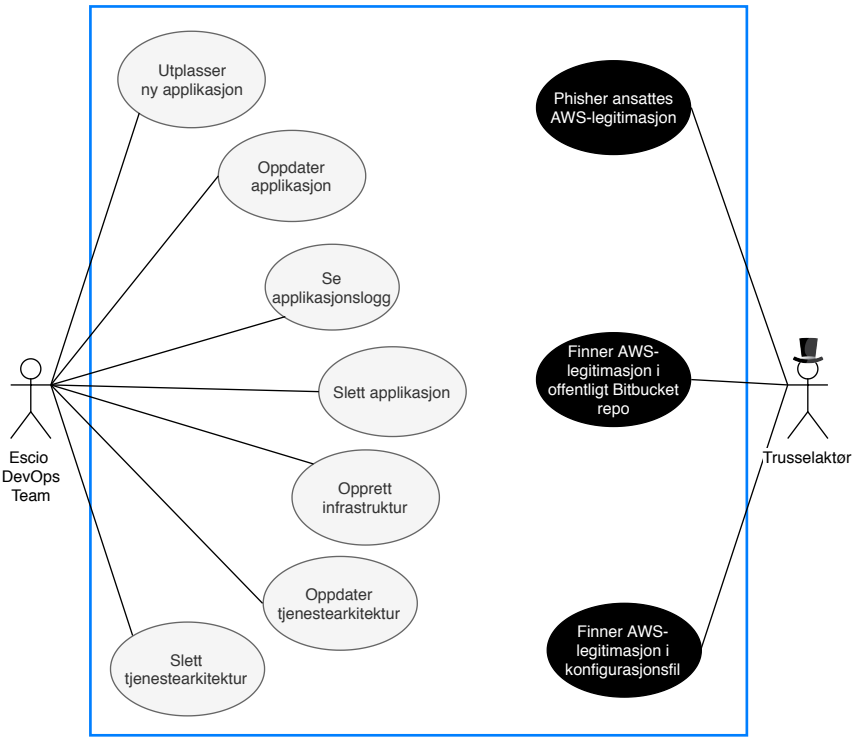
Oppdragsgiver har gitt uttrykk for at de veier brukerstøtte og dokumentasjon høyt. Skytjenesteleverandøren AWS tilbyr dette i forskjellige former hvor de viktigste blir gjengitt her.

- **Dokumentasjon:** Består i hovedsak av dokumenter, whitepapers og beste-praksis-guider [21].
- **Treningsprogram og sertifisering:** AWS tilbyr forskjellige typer treningsprogram og sertifiseringer [22].
- **Brukestøtte:** AWS tilbyr forskjellige nivåer; basic, developer, business og enterprise [23]. Kostnader ved bruk av disse er angitt fra $29 til $15000 USD per måned. Andre satser kan forekomme basert på bruk [24].
- **Konferanser:** AWS annonserer konferanser ved jevne mellomrom i forskjellige deler av verden, hvor mange av disse blir gjort opptak av og publisert blant annet på YouTube.

4.3 Sikkerhetsmessige krav

I dette avsnittet blir det prosjektgruppen mener er de viktigste sikkerhetsmessige aspektene ved prototypen belyst ved hjelp av et misuse case diagram og tilhørende misuse cases. Sikkerhet relatert til applikasjoner er ikke en del av denne vurderingen.

4.3.1 Misuse case



Figur 5: Misuse Case Diagram

Misuse case	**Phisher ansattes AWS-legitimasjon**
Aktør	Ekstern trussel
Hensikt	Utføre sabotasje på infrastruktur for egen vinning.
Beskrivelse	En trusselaktør sender en phishng-mail til en ansatt i bedriften og lurer den ansatte til å gi fra seg AWS-legitimasjon.
Konsekvens	Trusselaktøren har tilgang til infrastrukturen og kan utføre ondsinnede handlinger som kan påvirke bedriftens økonomi, pålitelighet og omdømme.
Tiltak	Opplæring av de ansatte om phishing angrep.

Tabell 8: Phisher ansattes AWS-legitimasjon

Misuse case	**Finner AWS-legitimasjon i offentlig Bitbucket repo**
Aktør	Ekstern trussel
Hensikt	Utføre sabotasje på infrastruktur for egen vinning.
Beskrivelse	Trusselaktør finner et offentlig repository der Bitbucket-pipelinen benytter AWS-legitimasjon direkte i klartekst.
Konsekvens	Trusselatør har tilgang til infrastrukturen og kan utføre ondsinnede handlinger som kan påvirke bedriftens økonomi, pålitelighet og omdømme.
Tiltak	Hold alle repositories skjulte. Ta i bruk Bitbucket sine miljøvariabler og benytt krypterings-funksjonen for sensitive variabler.

Tabell 9: Finner AWS-legitimasjon i Bitbucket

Misuse case	**Finner AWS-legitimasjon i konfigurasjonsfil**
Aktør	Ekstern trussel
Hensikt	Utføre sabotasje på infrastruktur for egen vinning.
Beskrivelse	En aktør anskaffer seg en konfigurasjonsfil som inneholder AWS-legitimasjon i klartekst og bruker dette til å sabotere infrastrukturen.
Konsekvens	Trusselaktør har tilgang til infrastrukturen og kan utføre ondsinnede handlinger som kan påvirke bedriftens økonomi, pålitelighet og omdømme.
Tiltak	Bruke parametere aktivt der legitimasjon benyttes. Unngå å bruke legitimasjon i konfigurasjonsfilene.

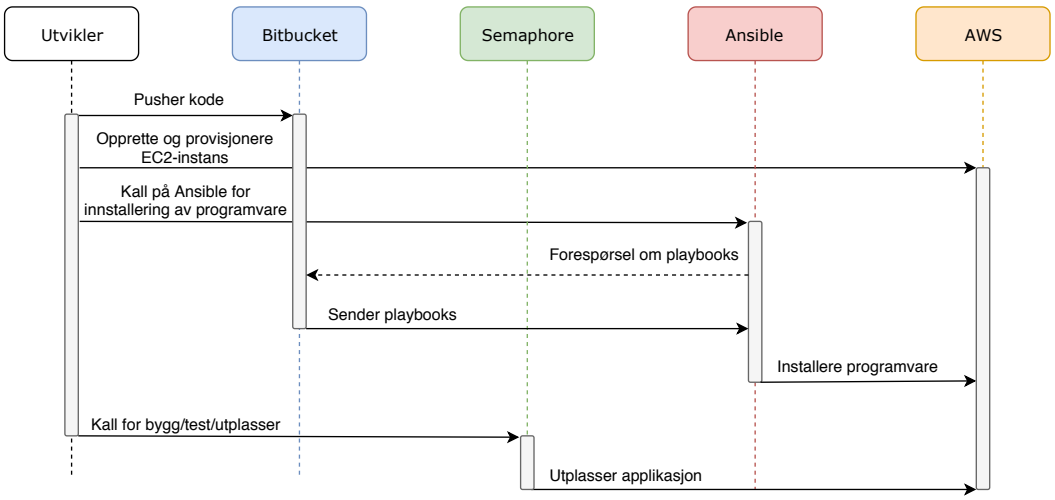
Tabell 10: Finner AWS-legitimasjon i konfigurasjonsfil

5 Oppdragsgivers praksis

5.1 Dagens praksis

For at leseren av rapporten skal få et bedre innblikk i hvordan oppdragsgiver utplasserer sine applikasjoner i produksjonsmiljøet og hvilke ressurser i AWS som drifter disse vil følgende kapittel forsøke å oppsummere dette.

Sekvensdiagrammet i figur 6 viser hvordan en nyutviklet applikasjon utplasseres til AWS. Utvikler sender ferdig utviklet kode til versjonskontrollsystemet Bitbucket. Bitbucket brukes til å oppbevare applikasjonskode og server-konfigurasjonsfiler. Når bedriften skal utplassere en applikasjon må en utvikler logge seg inn i AWS, for deretter å manuelt opprette og provisjonere en EC2-instans med nødvendige ressurser. Etter at serveren har blitt opprettet sender utvikler et kall til server-konfigurasjonsverktøyet Ansible. Denne henter en generisk konfigurasjonsfil fra Bitbucket som brukes til å installere nødvendig programvare på EC2-instansen, samt sette opp rettigheter til brukere. Etter at EC2-instansen er ferdig konfigurert og klar til bruk, benytter utvikler tjenesten Semaphore. Ved bruk av Semaphore sitt grafiske brukergrensesnitt bygges, testes og utplasseres applikasjonen ved hjelp av manuell handling. Escio har ikke satt opp noen form for autoskalering av ressursene dem disponerer. Dette betyr at dersom Escio har behov for å skalere opp eller ned tjenesten, må en utvikler gå inn å endre på dette manuelt.



Figur 6: Sekvensdiagram for oppdragsgivers utplasseringsprosess i AWS

5.2 Dagens praksis i forhold til beste praksis

For å sammenligne dagens praksis mot beste praksis, har boken *Infrastructure as Code: Managing servers in the cloud* av *Kief Morris* [2], og forfatter og programvareutvikler Martin Fowler brukt som faglitteratur [25].

Kief Morris skriver at versjonskontrollsystemer er et av de viktigste elementene i et programmerbart infrastruktur regime. Et versjonskontrollsystem burde innholde alt som kan være med å bygge og gjenoppbygge elementer i infrastrukturen. Versjonskontrollsystemet bør innholde konfigurasjonsfiler, testkode, kildekode og dokumentasjon. Fordelen med å versjonskontrollere elementer i infrastrukturen er flere. Det brukes til å holde orden på all historikk av hvilke endringer som har blitt gjort, og hvem som har utført disse. Dette øker sporbarheten under feilsøking og gir situasjonsforståelse for de ansatte. Om feilen ikke blir funnet er det mulig å gjøre tilbakerulling til tidligere fungerende versjon. Escio benytter seg av versjonkontrollsystemet Bitbucket som er i tråd med alle disse egenskapene [26].

Til å bygge, teste og utplassere applikasjoner bruker bedriften tjenesten Semaphore. Dette er et egnet CI/CD verktøy [27], blant annet fordi det har et kommandolinjegrensesnitt og støtte for eksekvering uten menneskelig tilsyn. Problemer oppstår da det ikke ligger til rette en infrastruktur som lar seg utnytte dette verktøyets egenskaper. Oppdragsgiver har en metodikk for integrasjon og leverase, men er ikke utført kontinuerlig. Martin Fowler forklarer [28] at å innføre CI i bedriften kan være med på å forbedre utviklingen av applikasjoner og kan være med å forebygge feil. Ved å benytte praksissen CD [29] vil det redusere sannsynlighetene for feilproduksjon av applikasjon.

Et annen viktig verktøy *Morris* beskriver, er bruk av et infrastruktur-definisjonsverktøy, ofte kalt orkestreringsverktøy. Dette er et verktøy som blir brukt til å definere, implementere og oppdatere infrastrukturarkitekturen og bruker konfigurasjon-definisjonsfiler til å allokere ressurser, modifisere, legge til, eller fjerne elementer, slik at den samsvarer med spesifikasjonene som er deklarert. Oppdragsgiver har ikke tatt i bruk et slikt verktøy for å holde orden på infrastrukturen. De har ikke definert tjenestene og ressursene infrastrukturen skal bruke i konfigurasjonsfiler. Dette skaper en uoversiktlig infrastruktur som kan være vanskelig å håndtere med tanke på konfigurasjonsdrift, skalering og monitorering.

Ansible er et server-konfigurasjonsverktøy og kan brukes til å allokere maskinvare, konfigurere nettverk, og installere programvare etter hva som er spesifisert i konfigurasjonsfilene. En av hovedoppgavene til verktøyet er å holde alle servere i en infrastruktur konsistent. Escio bruker i dag Ansible først etter en EC2-instans er provisjonert manuelt. Ved hjelp av en generisk konfigurasjonsfil, installeres nødvendig programvare for applikasjonene som skal driftes og rettigheter for brukere opprettes. Ifølge *Morris* skal Ansible automatisk opprette og provisjonere de definerte ressursene, samt unngå konfigurasjonsdrift. Ved å holde serverne konsistente vil det være enkelt å reprodusere flere servere uten menneskelig interaksjon.

6 Generelt og prototypen

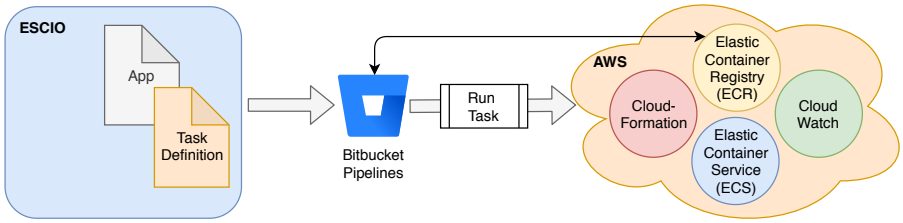
Prototypen som fremmes her er et forslag til løsning basert på bedriftens behov og krav, hvor fagfeltet programmerbar-infrastruktur sine råd om beste praksis har blitt ivaretatt. Råd om beste praksis ved bruk av AWS sine tjenester har også blitt brukt for å kvalitetssikre at benyttede ressurser konfigureres riktig. Prototypen legger vekt på at utviklerne i bedriften skal få en automatisert utrulling av sine applikasjoner, samt at infrastrukturen disse driftes på opererer mest mulig effektiv og selvstendig. Det er også vektlagt at utviklerne skal ha god oversikt over hele infrastrukturen, inkludert deres applikasjoner, og få tilbakemeldinger dersom feil oppstår. Det har også vært viktig å sette opp en infrastruktur som kan utvides og videreutvikles, slik at den kan dekke fremtidige behov for bedriften. Forslaget til utplassering av applikasjon og støttet infrastruktur er illustrert i figur 7 og 8, og er hovedelementene i prototype.

Den foreslåtte infrastrukturen består av flere forskjellige elementer. Sammen gjør disse elementene det enklere, tryggere og mer oversiktlig for oppdragsgiver å rulle ut og drifte sine applikasjoner. Det har blitt lagt vekt på at infrastrukturen skal være robust, både med tanke på tilgjengelighet og sikkerhet. Infrastrukturarkitekturen kan versjonskontrolleres og lagres i et repo, på samme måte som for applikasjoner. Nye revideringer av infrastrukturen kan enkelt utplasseres og testes i et testmiljø, før den blir tatt i bruk. Samtidig gjør dette at infrastrukturen enkelt kan flyttes mellom ulike AWS-regioner verden over, dersom en uforutsett hendelse skulle inntreffe regionen infrastrukturen er opprettet på. Det er også muligheter for å kunne orkestrere infrastrukturen på tvers av regioner, dersom bedriften finner dette nyttig. Dette kan gi oppdragsgiver fleksibilitet og ytterligere redundans.

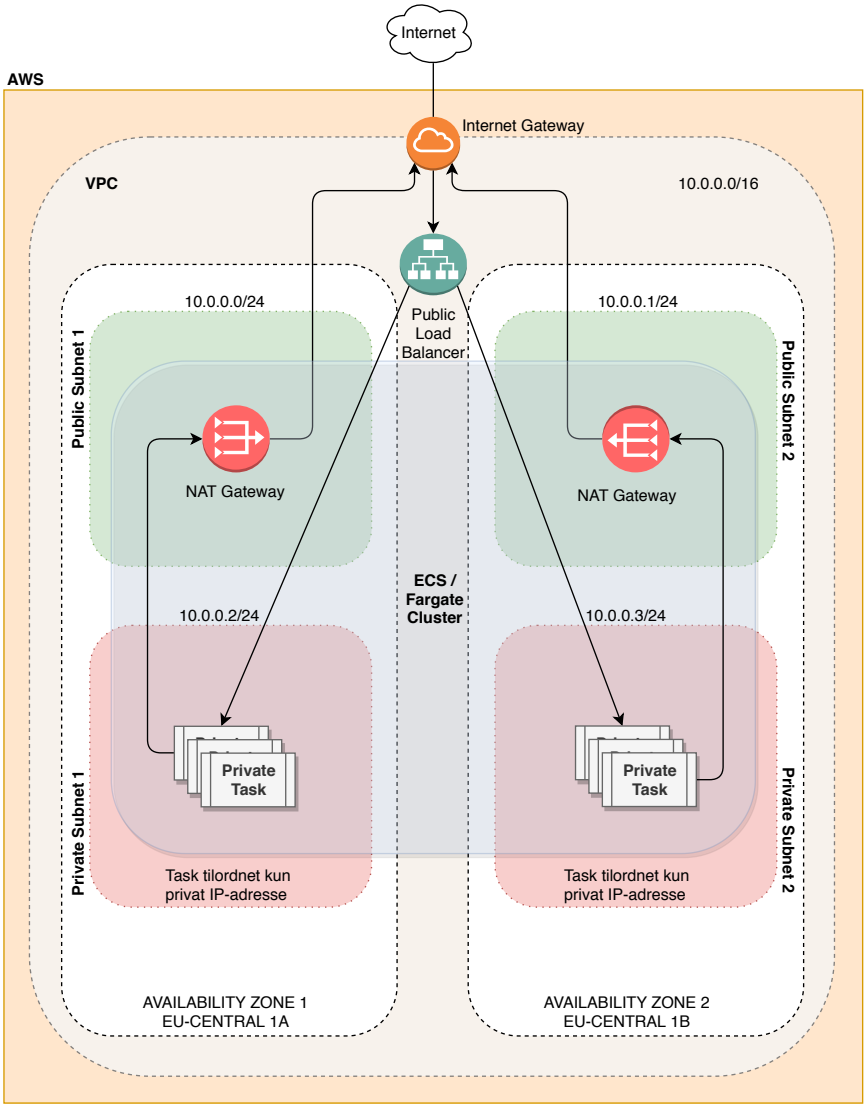
Infrastrukturen er tilpasset til containere i tråd med oppgavebeskivelsen. Det ble vektlagt at forskjellige applikasjoner kan ha ulike krav til maskinressurser. Løsningen støtter spesifisering av ressurstildeling på to nivåer. På oppgave-nivå og container-nivå. I tillegg kan det spesifiseres hvor mange container-instanser det skal kjøres av en type applikasjon i produksjon, og om dette antallet skal ha lov til å skalere automatisk ytterligere. En viktig del av oppgavebeskrivelsen var at løsningen skulle kunne lastbalansere og skalere automatisk. Lastbalansereren distribuerer trafikk mellom container-instansene slik at feiltoleransen forbedres, samtidig som den styrker tilgjengeligheten for kundene. For å sørge for at applikasjonenes ytelser ute i produksjonmiljøet til enhver tid er optimale, monitoreres containernes helse kontinuerlig.

Prototypen inkluderer implementasjon av et VPC. Nettverket er delt opp i to offentlige og to private subnett. Subnettene er satt opp over to tilgjengelighetsoner i AWS-regionen infrastrukturen opprettes. Hver tilgjengelighetsone i regionen er uavhengig av hverandre, og dersom feil oppstår i en av disse, vil forsatt applikasjonen være tilgjengelig for kundene. Dette er med forbehold om at det kjøres minimum to instanser av den aktuelle

applikasjonen. Lastbalansereren er plassert i et offentlig subnett og distribuerer aktuell trafikk mellom de to private subnettene plassert i de to tilgjengelighetsonene. Fordelen med dette er redundans, tilgjengelighet og sikkerhet. Utgående kommunikasjon skjer via en NAT-gateway i de to offentlige subnettene.



Figur 7: Prototypen for utplassering av applikasjoner med benyttede tjenester



Figur 8: Infrastrukturarkitekturen i prototypen og dens komponenter

7 Implementasjon av prototype

7.1 CloudFormation

CloudFormation er en kostnadsfri tjeneste i AWS som hjelper brukere til å provisjonere og administrere ressurser i en infrastruktur ved hjelp av konfigurasjonsfiler. Tjenesten gjør det enkelt å kontrollere og spore endringer, samt replikere ressurser ved behov. I tillegg til disse egenskapene tilbyr tjenesten flere sikkerhetsmekanismer som kan benyttes for å beskytte kritiske ressurser i infrastrukturen. Tjenesten kan betjenes ved bruk av terminalen eller gjennom et GUI. Disse egenskapene dannet grunnlaget for at CloudFormation ble implementert i prototypen. HashiCorp Terraform[1] ble beskrevet som en alternativ kandidat i rapporten som ble levert oppdragsgiver i løpet av prosjektet. Dette verktøyet er på lik linje med CloudFormation et meget godt egnet verktøy til formålet, og har noen fordeler som ikke CloudFormation besitter. Det samme kan sies om CloudFormation. Gjennom dette kapittelet blir de viktigste konseptene i tjenesten beskrevet. Det blir også nevnt noen sikkerhetsmekanismer som ble tatt i brukt i løpet av prosjektet, og noen som det anbefales oppdragsgiver å ta i bruk dersom dem finner dette gunstig.

7.1.1 Konsepter i CloudFormation

De tre grunnleggende konsepter i CloudFormation er Template, Stacks og Change Sets. Disse konseptene blir brukt gjennom hele rapporten, og blir derfor forklart nærmere.

Template

Et template er en mal i enten JSON- eller YAML-formatert tekst som beskriver infrastrukturen. CloudFormation bruker denne malen til å provisjonere de spesifiserte ressursene fra malen ut i AWS. Malen er oppdelt i ni seksjoner som er forklart i tabell 11.

Seksjoner	**Forklaring**
Format Version	Identifiserer kompatibiliteten til malen. Den siste versjonen av “format version“ er 2010-09-09. (ÅÅÅÅ-MM-DD). Verdien må spesifiseres som en *streng*. Dersom en mal ikke blir gitt en verdi vil CloudFormation anta siste versjon. Seksjonen er dermed valgfri.
Description	Feltet kan benyttes til å gi brukere informasjon om malen, teksten spesifiseres som en *streng*. Seksjonen er valgfri dersom ikke “Format version“ spesifiseres.
Metadata	Er valgfri, men kan benyttes til å inkludere informasjon om vilkårlige JSON eller YAML objekter spesifisert i malen. Benyttes ofte for å gi implementasjoninformasjon om angitte ressurser. Enkelte CloudFormation funksjonaliteter kan motta innstillinger og konfigurasjoninformasjon som spesifiseres i *Metadata*. For å kunne utnytte disse funksjonene må en “*Metadata Key*“ benyttes. Et eksempel på en slik nøkkel er "*AWS::CloudFormation::Init*". En slik nøkkel kjører et script som eksempelsvis installerer programvare på en gitt instans angitt i Metadata ved utrulling av malen.

[1] https://www.terraform.io/

Parameters	Er definerte verdier som sendes til en mal under *runtime*, enten ved oppdatering eller oppretting av en *stack*. Man kan referere til parametre fra *Resource* og *Outputs* seksjonen av malen.
Mappings	Denne seksjonen kan brukes til kartlegging av nøkler med tilhørende verdier, som kan benyttes til å angi betingede parameterverdier, lik en oppslagstabell. Bruken av *mappings* er valgfri.
Conditions	Kan brukes til å definere under hvilke omstendeligheter en entitet skal opprettes eller konfigureres. Det kan eksempelvis lages en *condition* som assosieres med en ressurs, slik at CloudFormation kun oppretter ressursen dersom betingelsen er oppfylt. Bruk av *Conditions* er valgfri.
Transform	Kan benyttes til å spesifisere en eller flere makroer som CloudFormation bruker til å prosessere malen. Makroene eksekveres i den rekkefølge som er spesifisert. Bruk av *transform* er valgfritt.
Resources	Spesifiserer *stack* ressurser og deres egenskaper. Dette er den eneste seksjonen i en mal som kreves. En deklarert ressurs får sin egen identifikator kalt Amazon Resource Namespace (ARN). ARN kan benyttes for å referere til en spesifikk ressurs.
Output	Denne seksjonen kan benyttes til å deklarere output-verdier. Disse verdiene kan importeres av andre *stacks*, returnes i svar etter kall, eller å vises i CloudFormation konsollen og/eller terminalen. Bruk av *Output*-seksjonen er valgfri.

Tabell 11: CloudFormation Template

Stack

En stack er en samling av AWS-ressurser som kan administreres som en enkelt enhet [30]. Alle ressurser i en stack er definert av et Cloudformation-template med samme struktur som ble beskrevet i avsnittet 7.1.1. Dette gir en fleksibel metode for å opprette, oppdatere og slette en samling av relaterte ressurser på en enkel måte. Å behandle ressurser i stacks gir også andre fordeler. Det gjør det mulig sette opp infrastrukturer som baserer seg på vanlige rammeverk som *multitier-architecture*[2], *service-oriented architecture*[3], eller å lage skreddersydde løsninger for tilpassede behov.

Infrastrukturen som ble utviklet for oppdragsgiver består av mange forskjellige ressurser, der de viktigste er beskrevet gjennom denne rapport. Disse ressursene er alle systematisert og gruppert i stacks, og er igjen kategorisert etter hvilke funksjon de har i infrastrukturen. Dette gjør det enklere for utviklere å gjøre endringer på ressurser i en aktuell funksjon i infrastrukturen. Felles for alle ressursene benyttet er at de danner grunnlaget for en fungerende infrastruktur, der hovedoppgaven til denne er å drifte de container-baserte applikasjonene for bedriften. I prinsippet kunne alle ressursene det var behov for bli definert sammen i en og samme stack. Dette er derimot ikke anbefalt praksis [31]. For kritiske ressurser tilbyr CloudFormation forskjellige sikkerhetsmekanismer som er omtalt avsnitt 7.1.3.

[2]https://en.wikipedia.org/wiki/Multitier_architecture
[3]https://en.wikipedia.org/wiki/Service-oriented_architecture

I tabell 12, er en oversikt over de forskjellige stacks prototypemiljøet er bygget opp av for oppdragsgiver. Kildekoden er levert sammen med rapporten.

Stacks i prototypen	Forklaring
cluster.yaml	Denne konfigurasjonsfilen definerer ressurser relatert til ECS/-Fargate klyngen, som hånterer og orkestrere livssyklusen til containere som kjøres i produkjonsmiljøet. ECS og Fargate er forklart nærmere i kapittel 7.2.
vpc.yaml	Denne konfigurasjonsfilen definerer den virtuelle private skyen, det private og offentlige subnettet, antall disponible IP adresser i nettverket og flere andre ressurser. VPC er er omtalt nærmere i kapittel 7.3
repository.yaml	Denne konfigurasjonsfilen definerer et Elastic Container Registry (ECR) til bruk for lagring av *docker images*. ECR er omtalt nærmere i kapittel 7.5
IAM.yaml	Identity and Access Management (IAM) er en service som håndterer sikker aksesskontroll til AWS-ressurser. IAM ble benyttet i prosjektet for å kontrollere autentiserte brukere og hvilke av disse som er autorisert til å bruke de forskjellige ressursene. IAM er omtalt nærmere i kapittel 8.1.

Tabell 12: Stacks

Change Sets

Change Sets gjør det mulig å se hvordan en eller flere endringer i en stack påvirker kjørende ressurser i en infrastruktur før disse endringene eventuelt implementeres [32]. Bruken av Change Sets i CloudFormation gjør det med andre ord mulig å oppdatere kjørende ressurser i en stack, uten å måtte ta disse ut av drift først. Dette fungerer ved at man først lager et change set for den originale stack som kjører. Etter endringene er utført i den nye konfigurasjonsmalen for den aktuelle stack, sendes denne til AWS. Dette kan gjøres fra terminalen eller via konsollen. Den originale kjørende konfigurasjonsmalen sammenlignes mot den nye, og det vises hvilke endringer som vil bli utført dersom den nye endrede konfigurasjonen blir implementert. Bruken av denne funksjonen kan være nyttig for oppdragsgiver dersom dem ønsker å se hvordan nye endringer vil påvirke infrastrukturen. I løpet av prosjektet har Change Sets blitt benyttet under konstruksjonen av infrastrukturen, og kan dermed anbefales for oppdragsgiver.

7.1.2 Validering av konfigurasjonsmaler

For validering av konfigurasjonsmaler kan det benyttes programvare. *Cfn-lint* er et program som eksaminerer kode i en fil for å se etter problemer med syntaks eller feil i koden som kan føre til problemer ved eksekvering. Programmet kan brukes selvstendig eller integreres i CI/CD-pipeline [33]. I prototype er cfn-lint integrert i pipeline 7.6.

7.1.3 Sikkerhetsmekanismer

CloudFormation tilbyr sikkerhetsmekanismer som kan beskytte stacks og ressurser på flere nivåer. Den underliggende fundamentale sikkerhetsmekanismen i AWS-systemet er IAM, og er derfor omtalt mer omfattende i kapittelet relatert sikkerhet 8.1.

Termination Protection

Den enkleste formen for å beskytte en stack er å benytte seg av funksjonen "Termination Protection [34]". Funksjonen kan være nyttig å implementere for kritiske deler av infrastrukturen. Aktivering av *termination protection* for en stack hindrer at denne slettes ved en feil. Funksjonen kan skrus på eller av for en eller flere stacks ved bruk av terminalen eller det grafiske brukergrensesnittet.

Stack policies

For å beskytte ressurser i en stack mot oppdateringer, kan stack policies tilknyttes en stack. Dette er et dokument i JSON eller YAML format som utformes i tillegg til en konfigurasjonsmal for en stack. Formålet med en slik stack policy, er at man kan skjerme ressurser innad i en stack mot oppdateringer [35]. Dette kan være spesielt nyttig for ressurser som holder data.

```
Statement:
- Effect: "Deny"
  Action: "Update:*"
  Principal: "*"
  Resource: "*"
  Condition:
    StringEquals:
      ResourceType:
      - AWS::RDS::DBInstance
```

Listing 1: Kodeutlippet viser en stack policy

Resource policies

Er en policy som tillegges en ressurs i en stack, og spesifiserer hvem som har tilgang til ressursen og hvilke handlinger de kan utføre på denne [36].

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  myVolume:
    Type: "AWS::EC2::Volume"
    DeletionPolicy: "Snapshot"
    Properties:
      AvailabilityZone: "us-east-1a"
      Size: "200"
```

Listing 2: Kodeutklippet viser en resource policy

7.1.4 Monitorering og revisjonskontroll

Dynamiske infrastrukturer gjør det trivielt å ta i bruk nye ressurser. Og ettersom ressursbruken øker, kan administreringen av disse med tiden bli en tidkrevende og vanskelig oppgave for bedrifter å håndtere uten god versjonskontroll og endringspolitikk. Dette kan på sikt føre til det som kalles konfigurasjonsdrift og få negative konsekvenser som

infrastrukturskjørhet og produksjonsangst. I boken *Infrastucture as Code* av *Kief Morris* [2] beskriver forfatteren flere utfordringer relatert til konfigurasjonsdrift og viktigheten av versjonskontroll. Flere av disse utfordringene er aktuelle for oppgaven, og tjenesten CloudFormation har enkelte innebygde funksjoner som kan benyttes for å mitigere disse.

Detektering av konfigurasjonsdrift

Chuck Meyer, Sr. Developer Advocate for AWS CloudFormation definerer konfigurasjonsdrift på følgende måte (sitatet er oversatt fra engelsk):

> *"En hver endring utført utenfor CloudFormation til en eller flere ressurser i en stack som modifiserer de forventede konfigurasjonsverdiene til ressursene vil resultere i konfigurasjonsdrift i en stack [37]."*

Detektering av drift fungerer ved at siste versjon av en konfigurasjonsmal under operasjon i CloudFormation får status som forventet-verdi. Denne forventede verdien sammenlignes kontinuerlig mot aktuelle verdier, altså konfigurasjonsverdiene til ressursser provisjonert. Drift er definert som endringene mellom forventede- og aktuelle verdier.

Endringer som fører til driftdetektering kan være en av følgende situasjoner:

- Modifisering av en stack-ressurs sin verdi.
- Modifisering av en normalverdi tilknyttet en ressurs i en stack.
- Sletting av en eller flere ressurser i en stack.

Konsekvensene av konfigurasjonsdrift i AWS kan medføre at automatiserte oppdateringsprosesser feiler, slik at disse blir forsinket. Ofte kan dette kreve at systemadministratorer må bruke tid til å gjennomføre feilsøking. Andre problemer kan relateres revisjonskontroll, ved at konfigurasjoner divergerer fra bedriftens godkjente arkitektur, og at disse endringene ikke har blitt dokumentert, testet og godkjent.

Dynamisk monitorering

En nyttig funksjon i CloudFormation er automatisk tilbakerulling. Dette kan utføres ved at tjenesten monitorerer ressursoppdateringens eller ressursopprettelsens tilstand. Dersom eksempelsvis ressursoppdateringen fører til at ressursens tilstand destabiliseres, som følge av redusert ytelse eller kommunikasjonsproblemer, vil CloudFormation rulle tilbake til forrige fungerende versjon automatisk. Tidsomfanget til automatisk tilbakerulling kan ytterligere ekspanderes ved å spesifisere *rollback triggers*. En rollback trigger muliggjør integrasjon av applikasjons- og ressursnivå alarmer fra tjenesten CloudWatch. Disse baseres på spesifiserte metrikker og lar CloudFormation monitorere ressursene lenger enn bare oppdateringsperioden. Dersom en endring utført mot en ressurs i en oppdateringsprosess fører til at en eller flere registrert alarmer utløses, vil CloudFormation stoppe oppdateringen og rulle tilbake i løpet av den spesifiserte tidsperioden. Dette tidsrommet gir også oppdateringen mulighet til å stabilisere seg. Skulle en alarm likevel utløses i løpet av perioden, ligger fortsatt forrige versjon tilgjengelig, slik at den raskt kan gjenopprettes. Når den spesifiserte tiden er utløpt, vil CloudFormation fjerne den gamle versjonen. Tjenesten CloudWatch er omtalt mer omfattende i kapittel 7.4. Fremgangsmåten for å benytte seg av rollback triggers er beskrevet under:

1. Definer en rollback trigger i JSON eller YAML, og spesifiser hvilke metrikker eller alarmer CloudFormation skal monitorere, og hvor lenge den/disse skal monitoreres under en oppdatering eller opprettelse av en stack.
2. Legg til den definerte rollback trigger-filen til den aktuelle stack som skal oppdateres i API-kallet for oppdatering i terminalen.

Registrering av endringer

Å registrere endringer utført i mot stack-ressurser er viktig, spesielt dersom noe galt skjer. Dette fungerer også som dokumentering. CloudFormation lagrer og monitorerer alle konfigurasjoner fra nåværende tidspunkt og bakover i tid ved hjelp av tjenesten AWS Config. Tjenesten tilbyr å definere regler for viktige ressurser, som kan brukes til å utløse varsler til systemadministratorer i tilfeller der endringer utføres mot disse. Denne funksjonen benytter tjenesten Amazon Simple Notification Service (SNS).

7.1.5 Oppsummering

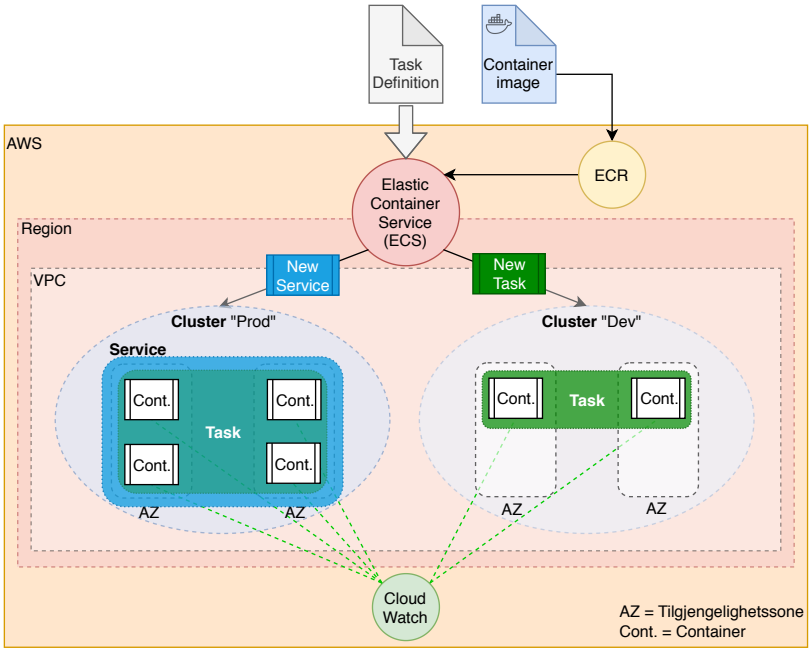
Dette avsnittet har tatt for seg tjenesten CloudFormation som gir brukere et kraftig verktøy for orkestrering av infrastrukturer på en oversiktlig og sikker måte. Tjenesten er i tråd med fagområdets krav, og oppdragsgiver oppfordres på dette grunnlaget å ta det i bruk. I avsnittet har kun de viktigste konsepter blitt omtalt, i tillegg til viktige sikkerhetsmekanismer og monitoreringstjener som tilbys. Brukerveiledningen [38] til tjenesten er i overkant av 3000 sider, og det ble på grunn av dette ikke mulighet til å se på alle egenskapene. I det neste avsnittet blir det tatt for seg tjenesten Elastic Container Service (ECS) som orkestrerer og styrer livssyklusen til container-applikasjoner.

7.2 Elastic Container Service (ECS)

Tjenesten ECS ble tatt i bruk i prototypen fordi oppdragsgiver ønsket et verktøy til orkestrering av sine container-applikasjoner i AWS. Grunnlaget for valget av ECS er flere. Først og fremst har tjenesten bred støtte blant andre tjenester innad i AWS-miljøet. Tjenesten er enkel å bruke, godt dokumentert og oppfyller bedriftens krav. Samtidig slipper driftspersonell å versjonskontrollere tjenesten. I bruk, brukes alltid siste versjon tilgjengelig, og oppdateringer skjer i bakgrunnen og administreres av AWS. På grunn av oppdragsgivers relativt begrensede driftsbemanning ble det i prosjektet forsøkt å finne gode løsninger som er enkle å administrere. Den største fordelen med ECS er at det kan benyttes sammen med utplasseringstjenesten Fargate, som er en serverløs tjeneste. Dette betyr at driftspersonell kun trenger å håndtere kontrollplanet. Oppgaver relatert til serverkonfigurasjon, opprettholdelse av konsistes og andre relaterte arbeidsoppgaver vil med dette kunne sløyfes, og tiden kan brukes til andre oppgaver som produserer verdi for bedriften. Ved å sammenligne ECS/Fargate mot Kubernetes, hvor det finnes ett kontrollplan og et dataplan, må brukerne administrere begge deler [39]. Og hver gang en ny versjon av Kubernetes blir tilgjengelig, må dette tas stilling til.

7.2.1 Komponenter i ECS

Gjennom dette avsnittet blir de viktigste komponentene i ECS og den serverløse utplasseringstjenesten Fargate beskrevet, i tillegg til viktige funksjoner og hvordan disse har blitt implementert i prototypen. Skalering og monitorering blir også omtalt. Figur 9, forsøker å illustrere ECS og hvordan en Task Definition sammen med et container-image danner en Task eller Service, samt hvordan disse utplasseres i en klynge 7.2.2 og orkestreres av ECS. Illustrasjonen viser også relaterte tjenester.



Figur 9: Komponenter i ECS

7.2.2 Cluster

En klynge er en logisk gruppering av *Tasks* eller *Services*. For en region må det opprettes minst en klynge. Alle opprettede klynger er såkalte hybrid-klynger. Med dette menes det at containere kan utplasseres ved bruk av EC2, Fargate eller i en kombinasjon av disse, noe som gir oppdragsgiver fleksibilitet. I prototypen brukes Fargate til utplassering av containere. Det blir opprettet to klynger, et for produksjon og et for utvikling. I produksjonsklyngen vil applikasjoner kunne driftes for kunde. I utviklingsklyngen kan utviklere jobbe i et isolert miljø, hvor de kan utføre tester og prøve ut ny funksjonalitet. En opprettet klynge kan betjene flere tilgjengelighetssoner i en region, slik at en Task eller Service får forbedret redundans. For mer informasjon om tilgjengelighetssoner og regioner se kapittel 7.3.1. I listing 3 vises det hvordan det opprettes en stack med to klynger i CloudFormation. De to klyngene har forskjellige navn. Det tildelte navnet er identiteten til klyngene, som andre ressurser i AWS kan referere til.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Container Cluster on ECS

# ECS klynge for AWS Fargate applikasjoner.
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: 'Production'

  TestCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: 'Development'
```

Listing 3: Kodeutklipp for opprettelse av to klynger

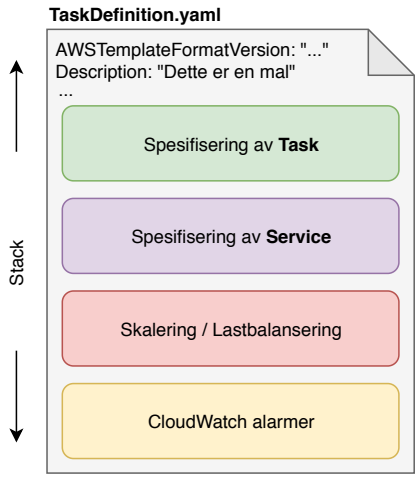
7.2.3 Task Definitions

En Task Definition kreves for å kunne kjøre containere i ECS. En Task Definition er en konfigurasjonsfil som provisjonerer alle ressursene en applikasjon trenger for å kjøre. Filen kan skrives i JSON eller YAML og brukes av tjenesten Bitbucket Pipeline i utplasseringsprosessen av applikasjonen. Dette er omtalt ytterligere i kapittel 7.6. Etter utplassering blir konfigurasjonsfilen synlig i CloudFormation i form av en stack, samt at applikasjonen vises i kontrollpanelet til ECS. Konfigurasjonsfilen er inndelt i forskjellige deler[40], og må spesifiseres ved hjelp av parametre. Tabell 13 gir en oversikt over de viktigste parametrene benyttet i prototypen. Konfigurasjonsfilens viktigste deler for prosjektet er vist i figur 10 og er beskrevet mer omfattende i sine respektive avsnitt.

Parametre	**Forklaring**
Family	Brukes til å navngi en Task med et versjonsnummer. Parameteret kreves.
Task Role	En rolle for at containerene skal kunne gjøre AWS API kall.

Task Execution Role	En rolle for å hente container images og skrive logger.
Network Mode	Definerer hvilke nettverksenhet som skal brukes. Kan defineres som none, bridge, awsvpc eller host. Fargate er avhengig av awsvpc ressurser.
Container Definitions	Definerer ressurser og konfigurasjoner som trengs for å kjøre en container i en Docker Daemon. Parameteret kreves.
Volumes	Brukes for å legge til ekstra lagringsplass containerene kan ta utnytte av.
Task Placement Contstraints	Definerer betingleser i henhold til Task. Det kan være tilgjengelighetssone, instansetype eller andre egendefinerte egenskaper.
Launch Types	Hvilken del av klyngen en Task skal kjøre i. Kan defineres som FARGATE, EC2 eller i kombinasjon.
Task Size	Allokerer en mengde minne og cpu containerene kan bruke.

Tabell 13: Parametere for Task Defintion



Figur 10: Datastruktur på Task Definition

Task

Denne delen av Task Definition spesifiserer alle egenskapene som må til for å utplassere en task i ECS. Det spesifiseres eksempelvis hvor mye CPU- og minneressurser som skal allokeres en Task. Ressursfordelingen kan spesifiseres på to ulike nivåer. På Task-nivået spesifiseres hvor mye ressurser en Task skal ha tilgjengelig totalt, mens på container-nivå spesifiseres det hvor mye ressurser hver enkelt container kan bruke. Skillet mellom disse er viktig, ettersom at det kan kjøres flere containere i en og samme Task. Valgmulighetene for CPU og minne er gjengitt i tabellen 7.2.3. Hver opprettet Task i Fargate får tildelt 10 GB med “Docker layer storage“ [41]. Denne lagringskapasiteten er ephemeral, og slettes dersom containeren stoppes. Lagringskapasiteten deles mellom alle containerne definert i Task. Dersom det trengs varig lagring, kan parameteret *volumes* spesifiseres.

CPU (value)	Memory value (MiB)
256 (0.25 vCPU)	512 (0.5GB), 1024 (1GB), 2048 (2GB)
512 (0.5 vCPU)	1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)
1024 (1 vCPU)	2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)
2048 (2 vCPU)	Mellom 4096 (4GB) og 16384 (16GB) inkrementelt med 1024 (1GB)
4096 (4 vCPU)	Mellom 8192 (8GB) og 30720 (30GB) inkrementelt med 1024 (1GB)

Tabell 14: Muligheter for valg av vCPU og minne

I prototypens Task Definition vist i listing 4 er det spesifisert sju komponenter i Task. Hvorav tre av disse komponentene leses fra parametere ettersom de varierer mellom applikasjonene. Parameteret *Family* leses inn for å navngi en Task, slik at brukeren kan lettere skille mellom hvilket arbeid som utføres. CPU og minne defineres også via parametere, fordi ressurskrav kan variere. Task er satt til å være avhengig av at Fargate er tilgjengelig. Den kan også eventuelt settes opp til å bruke EC2. Prototypens Task har blitt tildelt rollen *ECSTaskExecutionRole* definert i IAM 8.1 for å kunne autentiseres i ECR 7.5 og logføre applikasjonsaktivitet 7.2.5. Samt ble nettverksmodus satt til VPC 7.3 og ECS compiler til *FARGATE* for at den skal begrenses til den serverløse infrastrukturen.

```
Task:
  Type: AWS::ECS::TaskDefinition
    Properties:
    Cpu: !Ref TaskCpuAmount
    ExecutionRoleArn: !Importvalue 'ECSTaskExecutionRole'
    Family: !Ref TaskDefinition
    Memory: !Ref TaskMemoryAmount
    NetworkMode: "awsvpc"
    RequiresCompatibilities:
      - "FARGATE"
    ContainerDefinitions:
      - Container Definition
```

Listing 4: Konfigurasjonskode for en Task

Parameteret *Container Definitions* definerer en eller flere containere og ressursene knyttet til disse. I listing 5 blir alle containere tilknyttet et navn. Loggføring og kontroll blir dermed enklere og mer oversiktlig for alle de kjørende containere. Containerene tilknyttes et ImageURL fra et parameter. Den angitte URL-adressen skal lede direkte til et Docker image i et tilgjengelig register. I denne delen ble også selve containerens CPU- og minne-bruk spesifisert. Dette er viktig dersom flere containere skal kjøre under samme Task og fordelingen av ressurser er kritisk for containeres funksjonalitet.

```
ContainerDefinitions:
- Name: !Ref ContainerName
  Image: !Ref ImageURL
  Cpu: !Ref ContainerCpuAmount
  Memory: !Ref ContainerMemoryAmount
  PortMappings:
    - ContainerPort: !Ref ContainerPort
      Protocol: !Ref PortProtocol
  LogConfiguration:
    LogDriver: awslogs
    Options:
      awslogs-group: !ImportValue 'cluster:CloudWatchLog'
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: !Ref 'ServiceName'
```

Listing 5: Konfigurasjonskode for Container

Service

ECS gjør det mulig å kjøre og vedlikeholde et spesifisert antall instanser av en Tasks samtidig i en klynge. Dette kalles en Service [42]. ECS tilbyr to typer Scheduling-strategier av Service. I prosjektet ble strategien *Replica* [43] implementert fordi utplasseringstjenesten Fargate kun støtter denne. Replica plasserer og opprettholder et ønsket antall av Tasks i klyngen, og distribuerer Tasks på tvers av tilgjengelighetssonene. I tilfeller der en Task stopper eller feiler, uavhengig av årsak, vil det bli opprettet en ny. Dersom en Task ikke klarer å oppnå status *running* ved en gjennoppretting, vil Scheduler redusere gjennopprettingsforsøkene over tid. Dette hindrer unødvendig bruk av ressurser på en ikke-funksjonell Task. Brukeren vil bli varslet om problemet og kan bruke CloudWatch-logger for å løse det.

Kode gjengitt i listing 6 knytter Task til en Service. Dette gir støtte for skaleringen av antall Task, samt forbedrer applikasjonenes tilgjengelighet i produksjonsmiljøet. En Service knyttes opp til den spesifiserte klyngen og med Fargate som lanseringstype. I *DesiredCount* defineres det et antall Tasks ECS skal forholde seg til, og kan variere på grunn av autoskaleringen. I *DeploymentConfiguration* deklareres to tall. Disse tallene representerer nedre og øvrig grense for helsestatusen til en Service. Helsestatus definerer hvor mye en Service kan skalere opp eller ned antall Tasks under oppdatering eller utplassering av applikasjon [44]. I prototypen er intervallet satt til å være 50 og 200. Maksimal helse er satt til å være 200, slik at den kan opprette nye versjoner av Task før den fjerner gamle. Minimal helse er satt til 50 for kunne oppdatere Service uten å bruke ekstra kapasitet i klyngen. Når helautomatiske oppdateringer benyttes, er dette en god sikkerhetmekanisme for å opprettholde tilgjengeligheten og eliminere nedetid.

Service har blitt koblet opp til en VPC, og brukes til å definere hvor applikasjonen skal utplasseres. Av sikkerhetsmessige årsaker har Service blitt knyttet opp mot to private subnet, med en tilhørende sikkerhetsgruppe som restrikterer trafikk til containere. Dette er henholdsvis forklart i kapittel 7.3 og 8.2. Til slutt blir Service knyttet til en applikasjonslastbalanserer, forklart i kapittel 7.3.3.

```
Service:
Type: 'AWS::ECS::Service'
DependsOn: ListenerRule
Properties:
  ServiceName: !Ref 'ServiceName'
  TaskDefinition: !Ref Task
  Cluster: !ImportValue 'cluster:ClusterName'
  LaunchType: FARGATE
  DesiredCount: !Ref TaskAmount
  DeploymentConfiguration:
    MaximumPercent: !Ref MaxHealth
    MinimumHealthyPercent: !Ref MinHealth
  NetworkConfiguration:
    AwsvpcConfiguration:
      Subnets:
        - !ImportValue 'vpc:PrivateSubnetOne'
        - !ImportValue 'vpc:PrivateSubnetTwo'
      SecurityGroups:
        - !ImportValue 'vpc:FargateContainerSecurityGroup'
  LoadBalancers:
    - ContainerName: !Ref ContainerName
      ContainerPort: !Ref ContainerPort
      TargetGroupArn: !Ref TargetGroup
```

Listing 6: Konfigurasjonskode for Service

7.2.4 Skalering

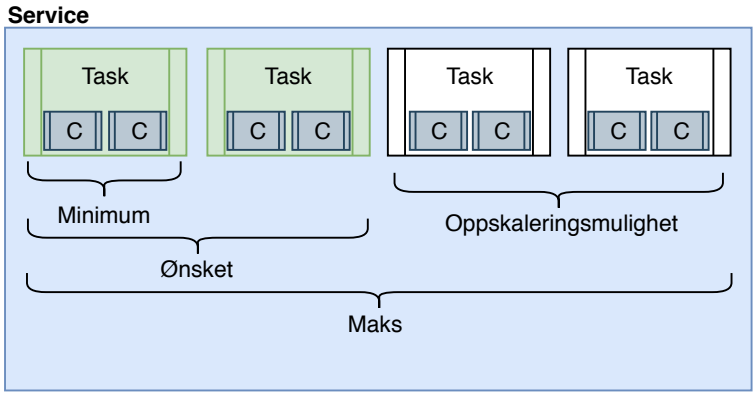
I prototypen har CloudWatch blitt brukt til å monitorere applikasjonene i drift. ECS publiserer metrisk data til CloudWatch angående CPU- og minnebruk 7.4. I tilfeller der en Task får kritisk høyt forbruk av CPU, vil CloudWatch sende et kall til ECS om at en Service må skalere opp antallet Tasks. Arbeidsmengden blir dermed automatisk fordelt blandt nye Tasks. Hvis CPU forbruket blir lavt, vil den skalere ned igjen til den minimum antall Tasks. Denne skaleringsprosessen styres av tjenesten Application Auto Scaling, og blir kalt på av CloudWatch sine alarm policies. Prossessen er automatisk og jobber selvstendig i infrastrukturen. Det er satt begrensinger på hvor mye en Service kan skalere, dette er for å hindre at den skalerer ukontrollert. I prototypen har dette blitt satt til å være minimum 1 og maksimum 10.

I listing 7 er det vist hvordan skaleringsmålet er *DesiredCount* [45], den kontrollerer antallet ønskede Task i en Service. Den angitte Service ble implementert fra parametere inn i *ResourceID*. En autoskaleringsrolle ble knyttet opp mot denne Service med tilganger til å gjøre endringer på vegne av ECS, mens skalerings-begrensinger blir satt av brukerens parameter. Dette gjør at applikasjonen er selvstendig og kan skalere uten menneskelig tilsyn.

```
ScalableTarget:
  Type: AWS::ApplicationAutoScaling::ScalableTarget
  DependsOn: Service
  Properties:
    ServiceNamespace: 'ecs'
    ScalableDimension: 'ecs:service:DesiredCount'
    ResourceId:
      Fn::Join:
        - '/'
        - - service
          - !Ref 'ClusterName'
          - !Ref 'ServiceName'
    MinCapacity: !Ref 'MinScale'
    MaxCapacity: !Ref 'MaxScale'
    RoleARN: !ImportValue AutoscalingRole
```

Listing 7: Konfigurasjonskode for skaleringsterskler

I statistikken for en applikasjon tilsendt fra Escio (se vedlegg B) kan det sees at ressursbruken for applikasjonen i drift er for det meste lav. Periodevis kommer en større mengde arbeid for CPU. Dermed ble det viktig for prototypen å sette en skaleringspolicy reaktivt i henhold til CPU-bruken. Skalering horisontalt med denne metoden heter Step-Scaling og er den skaleringsmetoden anbefalt ved bruk av ECS Service [46]. For å benytte seg av Step-scaling må det spesifiseres to terskler, en for høy og en for lav CPU-bruk. Step-Scaling fungerer slik at en Task skaleres opp eller ned, basert på de spesifiserte tersklene. Ved oppskalering kan Task skalere flere steg av gangen, mens ved nedskalering skaleres kun et steg av gangen. Det er forskjellige metoder Step-Scaling kan utnytte, men *ChangeInCapacity* passet best for å øke arbeidskapasiteten for en Service i et serverløst miljø. I figur 11 illustreres et eksempel på hvordan en Task med to containere i en Service kan skaleres med et *DesiredCount* på to, *minCapacity* på en og *maxCapacity* på fire.



Figur 11: Illustrasjon av skaleringsmulighet

7.2.5 Monitorering og logging

For å loggføre all aktivitet innad i containeren, har det blitt implementert en CloudWatch logg-gruppe. Logg-gruppen må etableres på forhånd, før den kan refereres til i *ContainerDefinition* 5. Det ble spesifisert en logg-gruppe i prototypen for test-applikasjonen, men det er anbefalt å lage en gruppe per kjørende applikasjon. Denne loggen består av contineren sine logger, samt det som blir skrevet ut i containeren sitt shell [47]. I prototypen defineres logg-gruppene sammen med klyngene. I listing 8 vises hvordan det ble opprettet en logg-gruppe for test-applikasjonen, kalt *FargateLog*. Den ble satt til å lagre logger i 30 dager, men dette kan endres etter behov.

```
  # Logger alt som skjer i klyngen, nyttig for feilsøking.
  CloudWatchLogsGroup:
   Type: AWS::Logs::LogGroup
   Properties:
     LogGroupName: 'FargateLog'
     RetentionInDays: 30
...
```

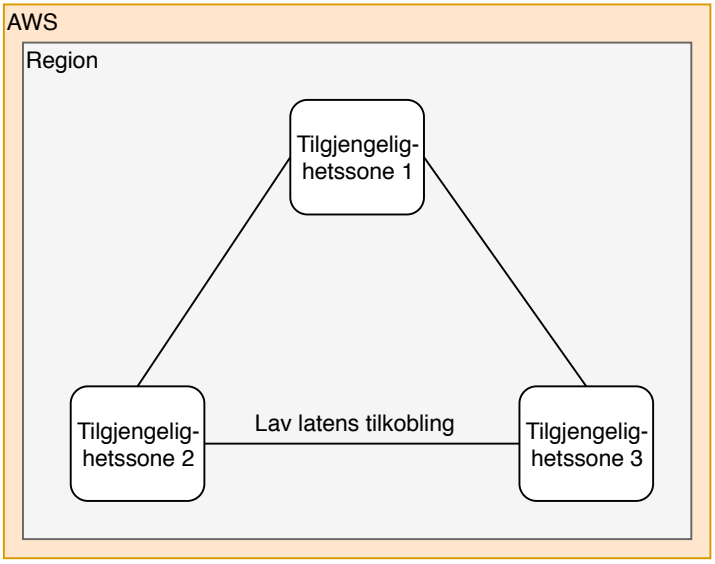
Listing 8: Konfigurasjonskode for monitoreringsgruppe

7.3 Virtual Private Cloud

I sammenheng med bruk av tjenesten Fargate ble det besluttet å sette opp en Virtual Private Cloud (VPC). Dette ble gjort for å dra nytte av flere av AWS sine tjenester, i tillegg til for å forbedre sikkerhet, skalerbarhet og redundans. En VPC tilbyr brukeren et egetdefinert isolert nettverk. Dette kan kombineres med avanserte sikkerhetsfunksjoner som security groups og ACL. Nettverket kan videre deles opp i private og offentlige subnets, og distribueres over flere tilgjengelighetsoner i en eller flere regioner.

7.3.1 Region og tilgjengelighetssoner

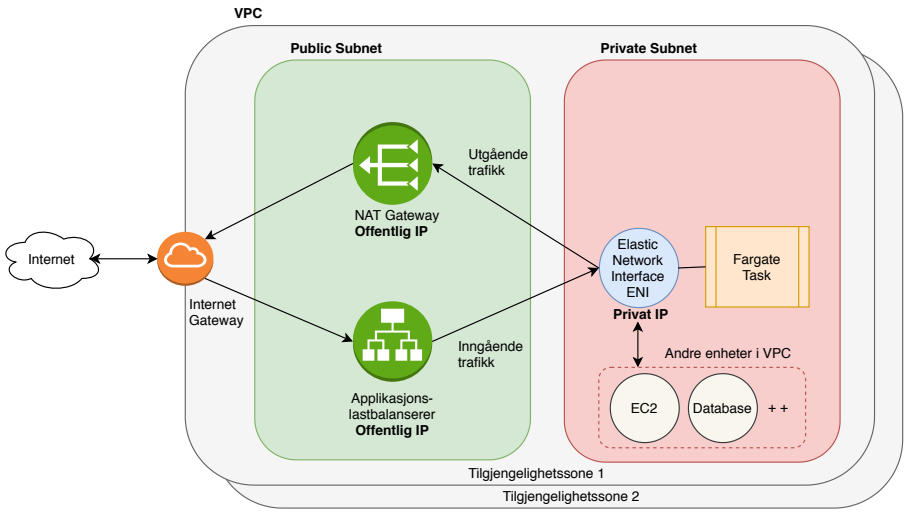
En region er et separert geografisk område som er helt selvstendig fra andre regioner. Det betyr at ressursene som blir provisjonert i en region ikke vil være tilgjengelige i andre regioner. Som vist på figur 12 har en region flere tilgjengelighetssoner. Disse arbeider isolert, men har en lav latens tilkobling med hverandre. På grunn av denne forbindelsen kan trafikken automatisk bli omdirigert i tilfeller der den ene tilgjengelighetssonen går ut av drift. I prosjektet har regionen Frankfurt blitt benyttet. I denne regionen har det blitt satt opp to tilgjengelighetssoner, slik at applikasjonene til oppdragsgiver får forbedret redundans og høyere feiltoleranse.



Figur 12: Illustrasjon av region og tilgjengelighetssoner

7.3.2 Komponenter i VPC

På figur 13 er det illustrert komponenter og hvordan trafikken strømmer gjennom prototypens nettverk. Nettverket er delt opp i private og offentlige subnet i to tilgjengelighetssoner. Ved å dele opp nettverket vil det være enklere for oppdragsgiver å kontrollere innkommende trafikk, og vil være med på å beskytte infrastrukturen mot trusler. Komponentene blir forklart etter hvor dem er plassert i nettverket. På grunn av oppdragsgivers krav om lastbalansering, har denne komponenten i nettverket fått sin egen seksjon.



Figur 13: Illustrasjon av VPC, private og offentlige subnets med deres komponenter

Privat subnet

I prototypen plasseres en Task i to private subnet. Hver opprettede Task i et privat subnet blir tildelt en privat IP-adresse gjennom et Elastic Network Inteface (ENI) [48]. ENI fungerer som et virtuelt nettverkskort og er bindeleddet mellom lastbalansereren og en Task. Lastbalansereren får ved hjelp av ENI vite hvilken container den skal sende trafikken til. Alle containere definert i en Task deler samme nettverkskonfigurasjon. I tilfeller der det finnes flere containere i en Task kan disse skilles ved hjelp av statiske port-nummere. Dersom en Fargate Task skal kommunisere med andre ressurser som databaser eller EC2 instanser, kommuniserer disse også via ENI.

Offentlig subnet

I det offentlige nettverket har det blitt plassert en lastbalanserer og en NAT-gateway. Lastbalansereren har som oppgave å være tilknytningspunktet til brukerne. Den mottar, filtrerer og videresender trafikken til det private nettverket. Siden alle Tasks som blir opprettet er private har det blitt implementert en NAT-gateway. Denne blir brukt når trafikken er ferdig prosessert og skal sendes tilbake til brukeren.

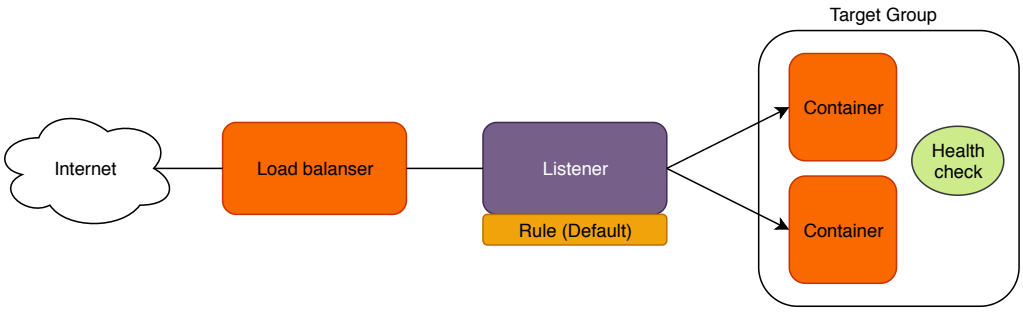
NAT

Hensikten med implementasjonen av NAT-gateway er å endre på sender/mottakeradressen på IP-pakker. Ved å endre på IP-pakker kan flere enheter benytte seg av samme IP-adresse. Ved hjelp av denne metoden kan alle enheter i et privat nettverk sende ut trafikk bak en offentlig IP-adresse, samt begrense bruken av IPV4-adresser da dette er begrenset.

7.3.3 Lastbalanserer

Escio bruker i dag rundt 40 EC2-instanser, dette uten en lastbalanserer. Ved å implementere en lastbalanserer i dagens praksis kunne Escio ha kjørt flere applikasjoner fordelt på instansene og dermed oppnådd redundans. En lastbalanserer kan spare bedriften for unødvendig bruk av ressurser, samt øke stabiliteten og feiltoleransen til applikasjonene. Lastbalansereren er et tilknyttingspunkt for brukere og har som oppgave å fordele trafikk mellom subnets i tilgjengelighetssonene.

AWS Elastic Load Balancer tilbyr tre typer lastbalanserere: Application Load Balancer (ALB), Network Load Balancer (NLB) og Classic Load Balancer (CLB). For valg av type lastbalanserer ble den sistnevne ikke vurdert da den ikke støtter Fargate [49]. Forskjellen mellom ALB og NLB er på hvilket nivå i OSI-modellen de opererer på. ALB opererer på nivå 7 (applikasjon) og baserer seg på HTTP/HTTPS trafikk. Lastbalansereren har avanserte rutingprotokoller og egner seg bra sammen med containerbaserte applikasjoner [50]. NLB opererer på nivå 4 (nettverk), denne baserer seg på TCP og TLS trafikk, og er optimalisert for å behandle flere millioner forespørsler i sekundet [51]. I prosjektet ble det besluttet at ALB er den beste lastbalansereren for oppdragsgivers web-applikasjoner grunnet AWS sine anbefalinger.



Figur 14: Illustrasjon av ALB komponenter

I figur 14 er arkitekturen bak Application load balancer illustrert, og består av følgende komponenter:

Komponenter	**Beskrivelse**
Listener	En listener er en port med protokoller, den har som oppgave å lytte etter forespørsler fra brukere og videresende forespørselen til en *Target Group*. For å bestemme hvilke forespørsler som skal sendes hvor, er det satt opp regler. Det må være minst en regel per listener som sier hvor trafikken skal bli sendt. En regel må spesifisere en Target Group, med minst en betingelse og hvilken prioritet forespørselen har.
Target Group	Er en homogen gruppe, det vil si at inne i hver enkelt Target Group kan det bare finnes en type Target. I en Target group er det også mulig å implementere health checks. Dette er beskrevet i seksjon 7.3.3
Target	En Target er enten en EC2 instans, container eller en IP-adresse.

Tabell 15: Komponenter innad i lastbalanser

Under implementasjonen av lastbalansereren ble det lagt vekt på sikkerhet. I prototypen har lastbalansereren og Fargate Tasks blitt delt inn i henholdsvis offentlig og private nettverk. Ved å legge de containerbaserte applikasjonene i et privat nettverk, vil det med riktig konfigurering redusere muligheten for at trussler kan påvirke bedriften. Fargate Tasks ligger i et isolert privat nettverk hvor de kun kan motta trafikk gjennom lastbalansereren. Lastbalansereren ligger i et offentlig nettverk, det vil si at hvem som helst

utenifra kan sende trafikk til lastbalansereren. ALB vil filtrere trafikken og sende den til riktig Target Group.

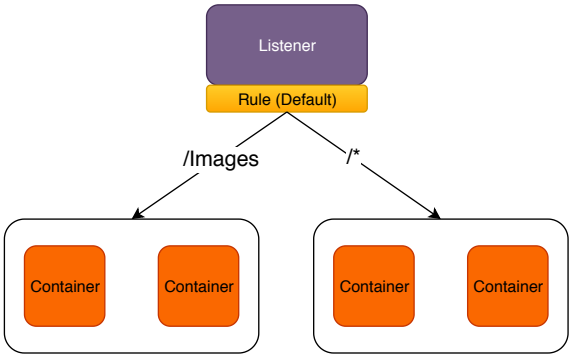
Listener

I listing 9 er det beskrevet regler til komponenten listener. I prototypen ble det opprettet en Target Group, fordi det var ikke nødvendig med flere da det kun var en testapplikasjon testet under prosjektet. Det ble i tillegg opprettet en listener, grunnet at det kun finnes en Target Group. Det er enkelt å opprette flere listeners og Target Groups hvis det blir nødvendig. Koden viser at listener spesifiserer en Target Group, en betingelse og hvilken prioritet forespørselen har. Siden det kun finnes en Target Group vil all trafikk bli distribuert til den. På linje 6 kan man se at *priority* har blitt satt til 2, hver enkelt regel har en prioriteringsliste hvor den med høyest prioritet vil bli behandlet først. I dette tilfellet har ikke prioriteten noen påvirking siden det kun finnes en regel, men hvis det blir definert flere regler er det viktig å klassifisere dem i riktig prioriteringsliste. På linje 7-10 blir betingelsen for trafikken definert, den blir satt til å være *path-pattern* og verdien på trafikken er satt til å være */**. Path-pattern dirigerer trafikken basert på URL-pathen og sender forespørselen til den Target Group som passer til den beskrivelsen. I koden er det konfigurert at alle forespørsler til applikasjonen blir sendt til den definerte Target Group. I denne løsningen ble det kun opprettet en generell regel på lastbalansereren, men det er mulig å definere mer spesifikke regler.

```
1  ListenerRule:
2      Type: AWS::ElasticLoadBalancingV2::ListenerRule
3      DependsOn: TargetGroup
4      Properties:
5          ListenerArn: !ImportValue 'vpc:PublicListener'
6          Priority: 2
7          Conditions:
8              - Field: path-pattern
9                Values:
10                 - /*
11         Actions:
12             - TargetGroupArn: !Ref TargetGroup
13               Type: forward
```

Listing 9: Konfigurasjonskode for applikasjonens lytteregel

Et eksempel for å forklare hvordan path-pattern kan utnyttes er illustrert i figur 15. På figuren er det satt opp to regler, disse har forskjellig prioritet. Verdien og prioriteten settes til å være henholdsvis */images(2)* og */*(1)*. Når en bruker prøver å kontakte tjenesten vil lastbalansereren prioritere regelen /images fordi den har høyest prioritet. Hvis regelen ikke passer til forespørselen, vil lastbalansereren benytte seg av den andre regelen */**. Trafikken vil dermed bli dirigert til den spesifiserte Target Group. Ved å dele opp trafikken på denne måten vil man enklere kunne analysere trafikken og gjøre justeringer for å optimalisere ressursene i infrastrukturen.



Figur 15: Eksempel på listener

Health checks

Health checks er en nyttig funksjon for å opprettholde stabilitet. I Target Group blir en health check definert, denne brukes til å monitorere helsen til containere. Lastbalanse-reren prøver kun å sende trafikk mellom healthy Targets, hvis en Target blir unhealthy vil trafikken automatisk bli omdirigert. Health checks kan ikke foreta seg operasjoner som å terminere eller kjøre opp nye Targets, dette blir gjort av ECS. I ECS sin Task De-finition har det blitt konfigurert at det alltid skal finnes minst to Tasks(Targets), samt definert minimum og maksimumsgrensen til en Target. Hvis en Target kommer under minimumsgrensen på 50% helse vil ECS terminere gjeldene Task og kjøre opp en ny automatisk.

```
TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
        VpcId: !ImportValue 'vpc:VPCId'
        Port: !Ref LoadBalancerPort
        Protocol: HTTP
        Name: !Join ['-',[!Ref 'ClusterName', !Ref 'ServiceName']]
        Matcher:
            HttpCode: 200-299
        HealthCheckIntervalSeconds: 10
        HealthCheckProtocol: HTTP
        HealthCheckTimeoutSeconds: 5
        HealthyThresholdCount: 5
        TargetType: ip
```

Listing 10: Konfigurasjonskode for lastbalanserer sin målgruppe

I koden 10 er det vist hvordan health checks er konfigurert. Det er anbefalt å ikke overkomplisere, derfor er de anbefalte konfigurasjonene til AWS blitt fulgt [52]. Som vist i koden er *HttpCode* satt til å være 200-299. *HttpCode* brukes til å sjekke om re-sponsen fra en Target er vellykket, hvis responsen er mellom intervallet vil det bety at responsen er i orden. Parameteren *HealthCheckIntervalSeconds* er satt til å være 10, den bestemmer hvor ofte hver enkelt Target skal bli sjekket. Her kan man sette alt mellom 5-300 sekunder. Det finnes ikke mange Targets i prototypen, det vil derfor være lurt å

sjekke Targets regelmessig. På linje 11 er *HealthCheckProtocol* satt til å være HTTP. Dette er protokollen parameteret bruker for å teste helsen til et Target. Her er det mulig å bruke HTTPS, TCP og TLS. HTTP ble brukt da det var anbefalt ved bruk av ALB.

For å teste om en Target er unhealthy blir parameteren *HealthCheckTimeoutSeconds* benyttet. Denne er satt til å følge anbefalingen fra AWS på 5 sekunder, hvis ikke det kommer en response innen denne tid kan det være en indikator på at en Target er unhealthy. *HealthyThresholdCount* paramteren forteller hvor mange ganger en Target må svare før den blir sett på som healthy. Her er anbefalingen blitt fulgt og har blitt satt til å være 5.

Andre anbefalte egenskaper

ALB er utstyrt med flere egenskaper, men har ikke blitt implementert i dette prosjektet. I listener finnes det flere regler som kan være nyttig for Escio å ta i bruk.

Regelen *Fixed response* går ut på å ha et forhåndsbestemt svar, lastbalansereren har et sett med fixed responses for å svare på spesifikke forespørsler. Typisk bruk av fixed response kan være ved vedlikehold av web-tjenester. Hvis en bruker prøver å kontakte tjenesten vil lastbalansereren kunne gi et svar uten å måtte sende forespørselen til backend.

Regelen *Redirects* har som oppgave å omdirigere en URL til en annen. Den har muligheten til å omdirigere en HTTP forespørsel til en HTTPS, dette vil være med på å styrke sikkerheten. Det er også mulighet til å omdirigere en bruker fra en nettside til en annen, denne fungerer utmerket hvis en bruker prøver å kontakte en utdatert nettside.

Sikkerhet

En sikkerhetsmekanisme i AWS er å implementere *Web Application Firewall (WAF)*. Denne tjenesten skal beskytte bedriften for kjente sårbarheter i form av SQL-injection eller cross-site scripting (XSS) [53]. I WAF kan det bli satt opp Access control lists (ACL), regler og betingelser for å tilltate eller blokkere innkommende HTTP/HTTPS forespørsler til ALB. Denne tjenesten medfører ekstra kostnader.

7.4 CloudWatch

Amazon CloudWatch er en monitoreringstjeneste som overvåker ressurser og applikasjoner i AWS. Monitorering har som mål å vise riktig informasjon til den personen som har behov for det og består av følgende elementer [13]:

Innhente og analysere data

Ved å innhente og analysere data kan CloudWatch automatisk fremstille metrikker for tjenestene brukt, samt lage sitt eget tilpassede dashbord. Ved å analysere data kan oppdragsgiver finne løsninger for å optimalisere ressursene ved hjelp av statistikken generert.

Hendelser

Amazon CloudWatch Events leverer en strøm av systemhendelser i sanntid som beskriver endringer i infrastrukturen [54]. Ved å konfigurere regler, kan hendelser sendes til en eller flere strømmer. CloudWatch kan håndtere hendelsene reaktivt ved å bruke korrigerende tiltak etter behov.

Logging

CloudWatch Logs er en sentralisert lagringsplass for all logg-data i infrastrukturen [55]. Servere, nettverksenheter og applikasjoner kan bruke egne logger. Det er da anbefalt å sentralisere loggen til samme lagringsenhet. CloudWatch har en funksjon som sender logger til en lagringstjeneste kalt S3. Ved å samle data på samme sted vil det være enklere å analysere loggene og se hva som er galt. Det er også enklere å ta backup av loggen.

7.5 Elastic Container Registry

Elastic Container Registry (ECR) er en administrert Docker container-register tjeneste. Den lagrer, administrerer og distribuerer Docker container-images. Amazon ECR er direkte integrert med Amazon ECS, som gjør produksjonsflyten enklere å håndtere [56]. Å bruke denne tjenesten har flere fordeler. For å kunne bruke registeret, trenger brukeren en autoriseringstoken. Ved å bruke AWS CLI sin get-login får Docker autentisering til å bruke ECR. I tillegg er brukeren avhengig av å ha de riktige tilgangene i AWS IAM. Disse funksjonene gjør ECR til et sikkert sted å lagre alle container-applikasjoner. I prototypen ble ECR deklarert som en stack i AWS Cloudformation.

Det er flere tjenester som tilbyr registre for docker images. Dockerhub og JFrog Artifactory er eksempler på andre registre, men ble ikke implementert i denne prototypen. Dockerhub er enkel og gratis å implementere, men mangler flere av de gode sikkerhetsmekanismene som AWS ECR tilbyr [57]. JFrog er stort og tilbyr høy tilgjengelighet, men er mer komplekst å håndtere og implementere [57]. AWS ECR kan erstattes av en av disse, men alle containerene skal kjøre i AWS ECS.

7.6 Bitbucket Pipelines

I prototypen ble Python brukt i pipeline for å kommunisere med AWS sitt API ved hjelp av en klient kalt Boto3. Boto3 gjør integrasjonen av Python applikasjoner, biblioteker og scripts lettere både inne i infrastrukturen og fra utsiden [15]. Den har to API nivåer; Client/lav-nivå og Resource/høynivå. Client-nivået gir en-til-en kartlegging til de underlagte HTTP API operasjonene, mens resource gjemmer spesifikke nettverks kall og gir ressursobjekter tilgang til attributter og funksjoner.

Bitbucket er bedriftens kildekodeverktøy og har en innebygget pipeline. Escio hadde et ønske om å utnytte dette verktøyet for å bygge opp den kontinuerlige utplasseringen av container-applikasjonene. Bitbucket pipeline er i prototypen bygget opp av forskjellige steg. Disse stegene gjenspeiles av hvert ledd i figuren CI/CD 1. Hvert steg i pipelinen er en egen eksekverende container. Denne kjører et definert image den puller ned fra DockerHub. Dette effektiviserer tiden det tar å kjøre en pipeline, ettersom at den slipper å installere komponenter som trengs for å kjøre de forskjellige stegene under hver eksekvering. For å definere hva en pipeline skal gjøre i Bitbucket, må brukeren skrive en *bitbucket-pipelines.yml*. I denne prototypen består denne av fire steg; test, build, stage og deploy.

Test

I denne delen er det utviklerens jobb å implementere tester. Dette er for verifisering av koden slik at en applikasjon ikke bygges og stages hvis den ikke fungerer som den skal.

Build

I denne delen brukes et Docker image med AWS-CLI installert. Steget starter med å logge seg inn på den AWS kontoen i regionen som er implementert i miljøvariablene til Bitbucket. Den bygger dermed applikasjonen fra en Dockerfile som skal ligge i root mappa til repositoriet. Etter containerens oppbygging, blir imaget pushet opp til Elastic Container Registry.

Stage

I denne delen verifiserer pipeline om koden i taskdefinition.yaml inneholder korrekte ressurser, unødvendig kode eller feil. Her brukes et *image* som inneholder cfn-lint og AWS-CLI. Cfn-lint er et åpent-kildekode program laget i samarbeid med AWS for å validere en konfigurasjonsmal opp mot CloudFormation sine verdier og egenskaper [33]. AWS-CLI brukes deretter for å validere at CloudFormation kan lese Task Definition.

Deploy

For å kunne effektivt og feilfritt kommunisere med AWS sitt API under utplassering av applikasjonen, ble AWS SDK i python scriptene benyttet. Dette steget kaller på scriptet *checkExsistingStacks.py* med Image URL som parameter. Her sjekkes det om det allerede finnes en Stack i CloudFormation med samme navn og velger enten *createStack.py* eller *updateStack.py* ut fra responsen. Disse to scriptene leser parametere fra *appconfig.py* og fyller disse inn i *taskdefinition.yaml* når de sender filen til CloudFormation. CloudFormation begynner så utplasseringen av applikasjonen og pipelinen er ferdig eksekvert.

I listing 11 vises alle parametere Task Defintion trenger. Det har vært viktig under arbeidet med prototypen at utrullingen ikke er bunnet til kun en spesifikk applikasjon, og kan brukes til flere av oppdragsgivers applikasjoner. Derfor ble Task Definition parametrisert, slik at flere prosjekter kan ta nytte av den samme løsningen ved å forandre på en sentral konfigurasjonsfil [2].

```
# -*- coding: utf-8 -*-
#Stack configuration
stack = dict(
  stackname= 'Application'
)
#Cluster configuration
cluster = dict(
  cluster_name = 'Production'
)
#Service configuration
service = dict(
  service_name = '<Name>Service',
  task_amount = '2',
  min_scale = '1',
  max_scale = '10',
  max_percent = '200',
  min_percent = '50'
)
#Task Definiton
task = dict(
  task_cpu= '256',
  task_memory= '512',
  definition= '<Name>Family'
)
#Container Definition
container1 = dict(
  container_cpu_amount= '256',
  container_memory= '512',
  loadbalancer_port= '80',
  container_port= '3000',
  protocol= 'tcp',
  name= '<Name>',
)
```

Listing 11: Eksempel på bruk av Task Definition sin parameter fil (appconfig.py)

Environment Variables

Bitbucket har mulighet for å legge til miljøvariabler som er tilgjengelige for alle stegene under eksekvering av pipeline [58]. Variablene blir brukt av pipeline når den trenger informasjon som varierer mellom applikasjonene, eller når sensitive variabler blir brukt. Bitbucket har en sikkerhetsfunksjon som skjuler variablene, og gjør at disse ikke kan skrives ut. Variablene blir dermed maskerert i alle piplinens logger. Dette legger til rette for at ingen kan se hva variablene inneholder og er godt egnet for autentiseringstokens

og legitimasjon [59]. For at prototypen sin pipeline skulle fungere, krevdes det at disse fire variablene ble definert:

Variabel	Beskrivelse
AWS_ACCESS_KEY_ID	Når en bruker opprettes i AWS IAM konsollen, kan det legges til nøkkeler for programmatisk tilgang. Denne variabelen definerer en bruker-ID og brukes av pipelinen til å knytte seg opp mot en spesifikk bruker i AWS. Denne burde være skjult med Bitbucket krypteringsfunksjon.
AWS_SECRET_ACCESS_KEY	En tilfeldig generert streng som brukes for validering av brukerinnlogging. Denne må være skjult med Bitbucket krypteringsfunksjon.
AWS_DEFAULT_REGION	Regionen applikasjonen skal publiseres. Må være en region der Fargate er tilgjengelig.
REGISTRY_URL	En URL til registeret hvor conaineren skal lagres for å så bli brukt i deploy-fasen. Dette kan være en AWS ECR, Dockerhub eller andre tjenster som tilbyr en *register* løsning.

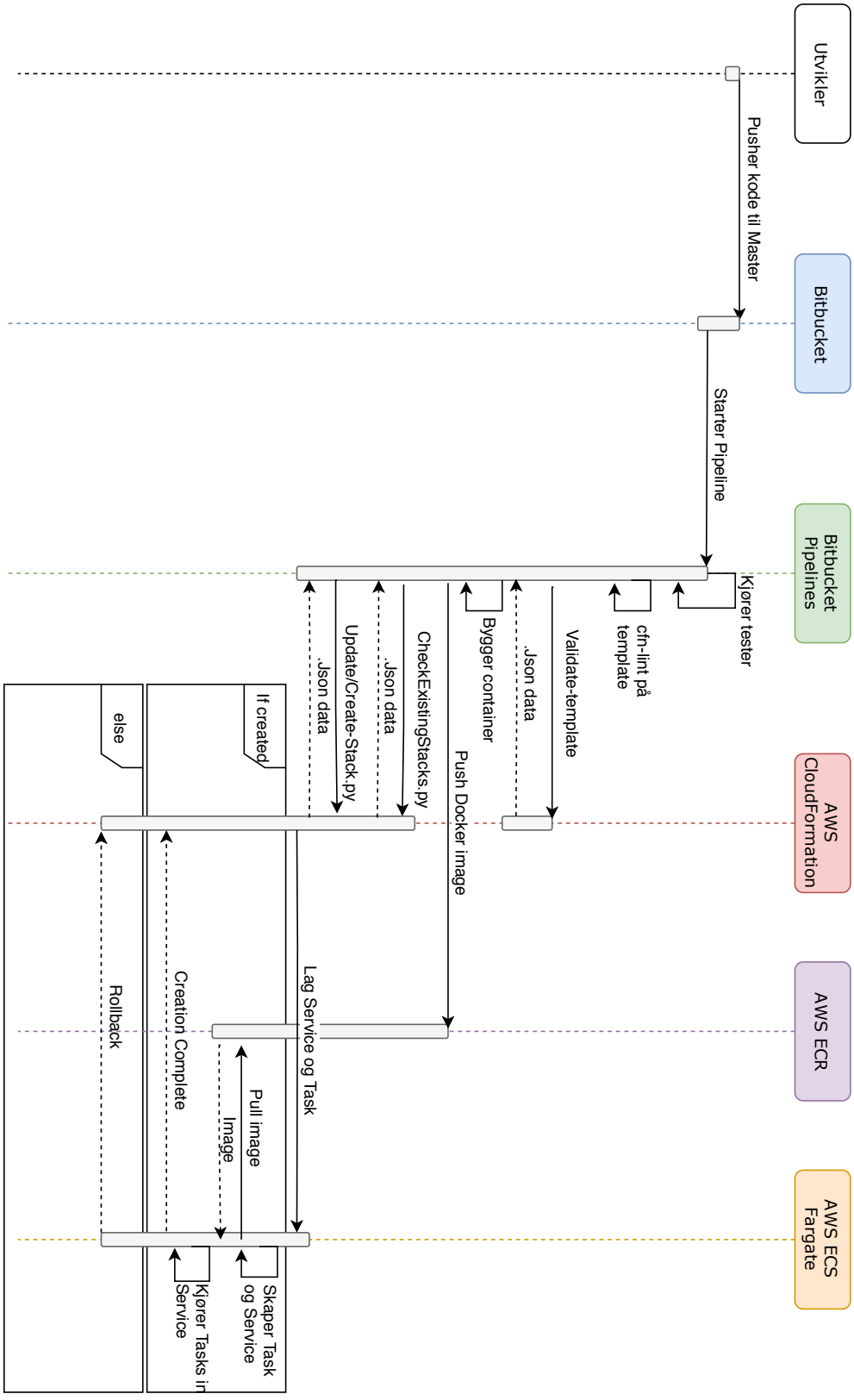
Tabell 16: Pipeline Enviornment Variables

Beste praksis og implementasjon

Ved praktisering av kontinuerlig utplassering er det viktig å følge gode prinsipper og regler for at implementasjonen av ny kode blir utført korrekt. Det er viktig at koden alltid er klar for utrulling. For Bitbucket betyr dette at all kode som *merges* til *master* er validert, testet og klar for utrulling [60]. Anbefalt arbeidsflyt fra Bitbucket er at alle tester kjøres automatisk for hver *commit* til *master*, samt at staging og deploy skal ha manuell godkjenning fra bruker i GUI for å fortsette pipeline. Dette hindrer at feil versjon av applikasjonen blir satt i produksjon. Oppdragsgiver ønsker en kontinuerlig utrulling til produksjon som innebærer at alle stegene i pipeline er uten menneskelig tilsyn. Dette avviker fra anbefalt praksis og kan være en risiko for løsningenes sikkerhet og tilgjengelighet. Å implementere et system som automatisk sender til produksjon setter store krav til utviklerne. Det er viktig med detaljerte tester som stopper pipeline ved feil og at gode rutiner på kvalitetssikringen av *branchen* før *merge*.

Sammenligning av praksiser

Ved å sammenligne det tidligere sekvensdiagram i figur 6 mot prototypens sekvensdiagram i figur 16 fremkommer forskjeller. Utviklerens medvirkningen har blitt redusert fra fire til kun én manuell handling.



Figur 16: Sekvensdiagram for prototypens utplasseringsprosess

8 Sikkerhet

8.1 Identity and Access Management (IAM)

AWS Identity and Access Management er sikkerhets protokoller basert på ”whitelist policies”-konseptet. Dette betyr at den tillater kun det som er spesifisert og at alt annet blir blokkert. Integriteten blir dermed styrket og fører til forbedret sikkerhet i infrastrukturen. IAM har noen hovedaspekter som påvirker tilgangene i AWS, gjengitt i tabell 17.

Identitet	**Beskrivelse**
Account root user	Brukeren som eier AWS kontoen. Den skiller seg fra administrator brukere ettersom at de har færre tilganger [61].
User	Legitimasjon for å gjøre endringer på tjenester fra utsiden av infrastrukturen.
Group	En gruppering av tilganger og roller som en bruker kan bli medelem av.
Role	En gruppering av tilganger for at tjenster skal kommunisere på innsiden av infrastrukturen.
Resource Policies	Tilganger på ressursnivå. Benyttes når en bruker trenger tilgang til en spesifisert ressurs opprettet (ARN) [62].
Capability Policies	Tilganger til forskjellige tjenester i AWS. Knyttes til brukere og tjenester når de skal kommunisere med hverandre.

Tabell 17: Komponenter i IAM

I utførelsen av IAM på dette prosjektet er det to brukere som trengs. En bruker med “Administrativ“ tilgang som kan kjøre ut infrastrukturen, og en bruker for å kjøre applikasjoner ut fra pipline. Brukeren *PipelineUser* må ha tilganger til en god mengde tjenester for å gjøre endringer i infrastrukturen under utrulling av applikasjoner. Brukere tildeles tilganger ved å bli tildelt en gruppe. Dette gjør det enklere å sentralt konfigurere tilganger som er implementert blant flere brukere. Gruppen er det nødt til å forholde seg til ”Least Privilege” ved å bare gi de tilgangene den har bruk for. For å utføre en slik praksis er det best å gi brukeren minst mulig rettigheter, for derretter å legge til flere etter behov [63]. I IAM skal ”*” unngås ettersom den gir ubegrensede rettigheter. I prosjektet ble det ikke brukt ”*”, for å følge anbefalt praksis. Trengs det tilgang til alle tjenestens ressurser kan det gjøres unntak for å redusere antallet policies, men er ikke anbefalt.

Før opprettelse av infrastrukturen, er det enkleste at konfigurasjonsfilene valideres og sendes inn manuelt fra en administrativ bruker. Da slipper man å endre på tilgangskonfigurasjonen under forandring av eksisterende stacks. Tjenestene i AWS kan ikke kommunisere fritt med hverandre. De krever tilhørighet til roller for å kunne utføre arbeid på vegne av andre tjenester. Dette er i hovedsak for å unngå at sikkerhetslegitimasjon blir delt.

Prototypen utviklet i prosjektet krever tre forskjellige roller:

- Rollen *ECSTaskExecutionRole* blir brukt av ECS for å utplassering av Tasks i en klynge.
- Rollen *ECSAutoscalingRole* bruker CloudWatch for å hente ut statistikk fra applikasjonene til bruk for automatisk skalering.
- Rollen *ECSRole* gir ECS muligheten til å håndtere sine egne ressurser. Dette medfører at ressursene holder seg konsistent.

Roller er en policy-basert tilgangskontroll tjenester kan bruke for å eksekvere arbeid i andre tjenester. Den har samme fremgangsmåte for beste praksis som for grupper [63].

8.2 Security Groups

En Security Group er en virtuell brannmur for en eller flere instanser i AWS [64]. Den brukes ikke på subnet nivået, men deklareres for de forskjellige instansene som kjører i subnettet. Om denne gruppen ikke deklareres, vil den bruke *default*. Disse gruppene er bygget opp av et sett *regler* for inngående og utgående trafikk. *Default* gruppen tillater all inngående trafikk fra andre instanser som tilhører samme gruppe, mens all utgående trafikk er tillat. I prototypen er bruken av servere abstraktert bort ved bruk av tjenesten Fargate, men implementasjonen av sikkerhetsgrupper utføres på samme måte.

I konfigurasjonsfilen Task Definition bruker Fargate en gruppe kalt *ContainerSecurityGroup* definert i *vpc.yaml*. Denne gruppen er satt til å tillate all trafikk. Det er viktig at oppdragsgiver ved implementasjon av prototypen endrer sikkerhetsgruppen til å bli mer restriktiv. Dette vil begrense hvem som kan bruke applikasjonen. Beste metodikk under deklareringen av slike sikkerhetstilganger er å følge “IP-whitelisting“ konseptet. Det går ut på å bruke *deny* på all trafikk, og legge til alle *allow* tilgangene ettersom det trengs.

9 Avslutning

9.1 Diskusjon

9.1.1 Abstraksjon og ansvarsforhold

AWS tilbyr et bredt spekter av tjenester med forskjellige nivåer av abstraksjon. Med abstraksjon menes det her å overføre ansvar fra Escio til skytjenesteleverandøren AWS. I prototypeoppsettet er det benyttet tjenester som abstrakterer bort mye av ansvaret til skytjenesteleverandøren.

Det er både fordeler og ulemper ved å abstraktere elementer i infrastrukturen til en tjenesteleverandør. Fordelen med å abstraktere ansvaret til AWS kan være med på å effektivisere hverdagen til de ansatte. Utviklerne kan fokusere på å utvikle applikasjoner, mens skytjenesteleverandøren håndterer de mer operasjonelle oppgavene. En annen fordel med abstraksjon kan relateres til sikkerheten til tjenestene i foreslått oppsett, med tanke på at disse er kontrollert av AWS og blir jevnlig oppdatert.

I prototypen er det noen tjenester som påvirker ansvarsforholdet mellom AWS og oppdragsgiver. Fargate brukes i løsningen til å abstraktere bort serverehåndtering. Dette betyr at AWS har ansvaret for tilgjengeligheten til containerne, men som tidligere nevnt i kapittel 4 skal tjenestenes oppetid tilsvare *99.99%* ifølge SLA [20]. Denne avtalen trygger påliteligheten til AWS sine tjenester. Å tildele mer ansvar til AWS medfører ekstra kostnader, noe som bør tas stilling til.

9.1.2 Låst til leverandør

Siden de fleste tjenestene benyttet i prototypen er AWS-tjenester, har infrastrukturens portabilitet blitt svekket. Prototypen er dermed i prinsippet låst til AWS, som kan medføre ulemper. Det kan eksempelvis bli vanskelig å benytte samme konfigurasjonsfiler hos andre skytjenesteleverandører. Til tross for dette finnes det verktøy som *Terraforming* [65]. Verktøyet kan brukes til å konvertere CloudFormation- til Terraform-konfigurasjonsfiler, som er støttet av andre store skyleverandører som Azure og Google Cloud [66]. Dette gjør at et eventuelt bytte av skytjenesteleverandør likevel kan gjennomføres uten større problemer. Prototypen har likevel noen AWS-spesifikke tjenester som i dette tilfellet må erstattes.

Å låse prototypen til AWS har også en rekke fordeler. AWS er en av verdens største leverandører innenfor skytjenester, og har flere tilgjengelige datasentre i Europa. Alle tjenester som tilbys oppdateres jevnlig, og ny funksjonalitet legges til. Prototypen inkluderer en veldig liten del av alle tjenestene de tilbyr. Om det ønskes å implementere flere tjenester er de som oftest kompatible med hverandre og har lett tilgjengelig dokumentasjon. Tjenestene kan styres sentralt fra CloudFormation, og det blir enklere å

administrere hele infrastrukturen.

Før implementasjon av prototypen, anbefales det å gjøre seg godt kjent med leverandøren med tanke på kostnader, tjenester og alternative løsninger. Det finnes flere verktøy og tjenester som dekker kravenspesifikasjonen til oppdragsgiver. Noen alternativer er beskrevet i vedlegg F.

9.1.3 Kontinuerlig utplassering

I prototypen har det blitt implementert kontinuerlig utplassering. Dette er ikke alltid en anbefalt praksis, men det har blitt implementert da det har vært et ønske fra oppdragsgiver. Ved innføring av denne praksisen, vil bedriften oppnå en automatisk utplassering av applikasjoner til produksjonsmiljøet.

Å implementere en kontinuerlig utplassering bygger videre på CI/CD-metodikken [67]. Når applikasjonen automatisk leveres til kunde på denne måten, vil oppdragsgiver kunne få raskere tilbakemeldinger fra brukerene angående feil ved applikasjonen. Kunden vil kunne se flere regelmessige endringer og mer funksjonalitet over tid. Utviklere kan også lettere se det endelige produktets resultat. For at denne prosessen skal fungere effektivt, må en rekke krav oppfylles av oppdragsgiver. Applikasjonene er avhengige av at Escio har gode rutiner i forhold til testbasert utvikling. Tester i henhold til feil, sikkerhet og integrasjon kreves for at produksjonsfeil skal unngås. I prototypens pipeline-løsning, er det lagt til rette for at slike tester kan implementeres av utviklere. Alle endringer som blir utført på produktene må være godt dokumentert. Dette vil gjøre at utviklere sparer tid på å finne ut hvilken funksjonalitet som burde endres. For å få mest ut av tilbakemeldinger, anbefales det å benytte seg av issue-trackers. Da blir det enklere å respondere raskt til problemer. Prototypen har en egen utviklingsklynge for integrasjonstesting av applikasjoner. Her kan utviklere utnytte samme ressurser for å forsikre seg at applikasjonen kan rulle ut til Fargate.

Valget om å implementere kontinuerlig utplassering er mer en bedriftsavgjørelse enn en teknologisk. Oppdragsgiver må ta en beslutning på hvilke applikasjoner som passer til metodikken. Det er viktig at bedriften opprettholder kravene for å få best utnytte av kontinuerlig utplassering.

9.2 Evaluering av arbeidet

Organisering av prosjektet

Alle i gruppen er fornøyde med organiseringen av prosjektet. Medlemene har vært konsistente på oppmøte. Noen av medlemmene ønsket muligheter for mer eget arbeid, noe som dette ble innvilget i perioder. Likevel har mesteparten av arbeidet blitt utført i plenum. Samarbeidet med oppdragsgiver og veileder har etter gruppens mening fungert godt.

Prosessen

For utvikling av prototype har utviklingsmodellen Scrum blitt fulgt. Det ble opprettet fire sprinter, og hver sprint startet med Sprint planning og ble avsluttet med Sprint retrospect review. De to første sprintene ble brukt til å gjøre undersøkelse i fagområdet, samt produsere en rapport til oppdragsgiver. De to siste ble brukt til å utvikle prototypen og skrive bachelor-rapport. Gruppen mente at det gikk noe lang tid til skrive den selvstendige rapporten for Escio, og det hadde vært ønskelig om mer av denne tiden kunne gått til å utvikle prototypen.

Arbeidsfordeling

Arbeidet utført iløpet av prosjektet har vært jevnlig distribuert basert på medlemmets rolle. Gruppen har et felles inntrykk av at alle medlemmene har arbeidet i omtrent like stor grad. Ved hjelp av verktøyet Trello og JIRA ble det enkelt å fordele arbeidet i felleskap. Gruppen hadde mer erfaring med bruk av Trello enn JIRA. Alle medlemene har vært villige og engasjerte til å påta seg arbeidsoppgaver.

Måloppnåelse

I forkant av prosjektet ble resultat-, og effektmål opprettet prosjektplanen E. Gruppen mener at målene er nådd, og er fornøyd med resultatet.

Læringsutbytte

Under prosjektets oppstart var medlemenes kunnskap om fagområdet programmerbar-infrastruktur (IaC) og AWS begrenset. Prosjektet har økt medlemmenes kunnskap og praktiske evner innenfor fargområdet i stor grad.

9.3 Fremtidig arbeid

Evaluering av prototype

Før fremmet prototype tas i bruk, anbefales det at en evaluering og test blir utført. Dette kvalitetssikrer løsningen, samt forsikrer bedriften at prototypen er ønskelig å implementere.

Minne skalering

Prototypen skalerer ikke på minnebruk, ettersom at det var størst variasjon i CPU bruk. For applikasjoner hvor det er store svingninger i minnebruk, vil det lønne seg å lage oppskalerings- og nedskaleringspolicies for dette.

Security Groups per applikasjon

Applikasjoner som rulles ut med denne løsningen har en standard sikkerhets gruppe der all trafikk fra alle porter er tillatt. Det burde bli laget en egen sikkerhetsgruppe med egendefinerte tilganger per applikasjon for å hindre uønsket trafikk.

Route 53 for egendefinerte domener

Prototypen skaper et tilknytningspunkt fra en egen DNS. Derfor blir tilknytningspunktet til applikasjonen via last-balanseren veldig langt og vrient å huske. Om bedriften har et eget domene, burde dette domene knyttes opp mot AWS Route 53 slik at last-balanseren kan benytte det.

Sikkerhetsskanning

For å sikre applikasjonene som rulles ut til denne løsningen, vil det lønne seg å implementere en sikkerhetsskanning av containerne. *Twistlock* [68] er et eksempel på et sikkerhetsskanningsverktøy for container-applikasjoner med god støtte for automatisering i CI/CD løsninger.

Varslingssystem - SNS

Prototypen har implementert CloudWatch for monitorering og varsling om hendelser i infrastrukturen. Disse varslingene er kun tilgjengelig i CloudWatch. Ved å implementere et varslingssystem, kan AWS varsle via kommunikasjonsverktøy som for eksempel Slack. AWS SNS er en tjeneste laget for å sende varselmeldinger til eksterne tjenester.

Hybrid-klynger

Det er mulig å implementere EC2-instanser i en klynge for at den skal kjøre applikasjoner på servere. For at servere skal være konsistente, burde det settes opp et serverkonfigurasjonsverktøy. Oppdragsgiver bruker i dag verktøyet Ansible, som er godt egnet til dette formålet.

Kostnad

Det har ikke blitt gjort forsøk på å gjennomføre en kostnad-nytte-analyse for prototypen fremmet i prosjektet. En slik analyse anbefales å gjennomføre.

9.4 Konklusjon

Hvilken løsning kan tilfredsstille oppdragsgivers krav og behov for en automatisert utplassering av container-applikasjoner i en orkestrert infrastruktur i AWS?

Det finnes en rekke løsninger sammensatt av forskjellige aktuelle verktøy og tjenester som kan dekke oppdragsgivers krav og behov. Etter en vurdering konkluderte prosjektgruppen at en serverløs infrastruktur orkestrert av CloudFormation, med en automatisk utplassering av applikasjoner i container-orkestreringstjenesten ECS, vil kunne gi Escio fordeler. Prototypen fremmet i rapporten er godt egnet for bedrifter med ønske om å begrense driftsrelaterte oppgaver, men på samme tid kunne opprettholde oversikt og kontroll. Oppsettet lar seg enkelt videreutvikle og er godt egnet til å kunne tilpasses behov.

Bibliografi

[1] Wikipedia. 2019 (Hentet 05.03.19). Devops. https://en.wikipedia.org/wiki/DevOps.

[2] Morris, K. 2016. *Infrastructure as code: managing servers in the cloud*. O'Reilly.

[3] Andersen, P. B. 2018 (Hentet 05.03.19)). Automatisering. https://snl.no/automatisering.

[4] David Stacey, Mikhail Prudnikov, A. K. X. S. 2017 (Hentet 06.03.19). Practicing continuous integration and continuous delivery on aws. https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf.

[5] Andrew S. Tanenbaum, H. B. 2015 (Hentet 20.03.19). *Modern operating systems*. Pearson.

[6] AWS.inc. 2019 (Hentet 11.03.19). Amazon web services. https://aws.amazon.com/.

[7] AWS.inc. 2019 (Hentet 19.04.19). What is aws cloudformation? https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html.

[8] AWS.inc. 2019 (Hentet 11.03.19). Ecs. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html.

[9] AWS.inc. 2019 (Hentet 08.04.19). Amazon virtual private cloud. https://aws.amazon.com/vpc/.

[10] AWS.inc. 2019 (Hentet 12.05.19). Elastic load balancing. https://aws.amazon.com/elasticloadbalancing/.

[11] AWS.inc. 2019 (Hentet 12.05.19). Amazon ec2. https://aws.amazon.com/ec2/.

[12] Amazon. 2019 (Hentet 11.03.19). Aws fargate. https://aws.amazon.com/fargate/.

[13] AWS.inc. 2019 (Hentet 12.05.19). Amazon cloudwatch. https://aws.amazon.com/cloudwatch/.

[14] AWS.inc. 2019 (Hentet 12.05.19). Aws command line interface. https://aws.amazon.com/cli/.

[15] AWS.inc. 2019 (Hentet 12.3.19). Tools for amazon web services. https://aws.amazon.com/tools/.

[16] Wikipedia. 2019 (Hentet 30.03.19). Event-driven programming. https://en.wikipedia.org/wiki/Event-driven_programming.

[17] Wikipedia. 2019 (Hentet 12.05.19). Function as a service. https://en.wikipedia.org/wiki/Function_as_a_service.

[18] AWS.inc. 2019 (Hentet 12.05.19). Aws lambda. https://aws.amazon.com/lambda/.

[19] Sanjay Chaudhary, Rajkumar Buyya, G. S. 2017 (Hentet 30.03.19). *Research advances in cloud computing*. Springer.

[20] AWS.inc. 2019 (Hentet 01.05.19). Amazon compute service level agreement. https://aws.amazon.com/compute/sla/.

[21] AWS.inc. 2019 (Hentet 6.04.19). Aws documentation. https://docs.aws.amazon.com/.

[22] AWS.inc. 2019 (Hentet 06.04.19). Training and certification. https://aws.amazon.com/training/.

[23] AWS.inc. 2019 (Hentet 06.04.19). Getting started with aws support. https://docs.aws.amazon.com/awssupport/latest/user/getting-started.html.

[24] AWS.inc. 2019 (Hentet 06.04.19). Aws support plan pricing. https://aws.amazon.com/premiumsupport/pricing/.

[25] Fowler, M. (Hentet 28.3.19). About martin fowler. https://martinfowler.com/aboutMe.html.

[26] Atlassian. 2019 (Hentet 16.04.19). Version control software for professional teams. https://bitbucket.org/product/version-control-software.

[27] Semaphore.inc. 2019 (Hentet 20.04.19). Ci/cd environment. https://docs.semaphoreci.com/category/57-cicd-environment.

[28] Fowler, M. 2006 (Hentet 28.03.19). Continuous integration. https://martinfowler.com/articles/continuousIntegration.html.

[29] Fowler, M. 2013 (Hentet 28.03.19). Continuous delivery. https://martinfowler.com/bliki/ContinuousDelivery.html.

[30] AWS.inc. 2019 (Hentet 20.04.19). Working with stacks. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html.

[31] AWS.inc. 2019 (Hentet 20.04.19). Aws cloudformation best practices. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#organizingstacks.

[32] AWS.inc. 2019 (Hentet 20.04.19). Aws cloudformation concepts. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html#w2ab1b5c15c11.

[33] Meyer, C. 2019 (Hentet 23.4.19). Git pre-commit validation of aws cloudformation templates with cfn-lint. https://aws.amazon.com/blogs/mt/git-pre-commit-validation-of-aws-cloudformation-templates-with-cfn-lint/?fbclid=IwAR0h5Uw4iF_vVXgnemrSIVPUWUsZ7Wk7KnSV6n3gXSo0gtON-L6WNu0pXLM.

[34] AWS.inc. 2019 (Hentet 21.04.19). Protecting a stack from being deleted. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-protect-stacks.html.

[35] AWS.inc. 2019 (Hentet 21.04.19). Prevent updates to stack resources. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/protect-stack-resources.html.

[36] AWS.inc. 2019. Protecting a stack from being deleted. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-protect-stacks.html.

[37] AWS.inc. 2018 (Hentet 12.05.19). Deep dive on aws cloudformation. https://www.youtube.com/watch?v=KXUsyApAI3Y&t=.

[38] AWS.inc. 2019 (Hentet 21.04.19). Identity-based policies and resource-based policies. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf.

[39] Kubernetes.inc. 2019 (Hentet 22.04.19). Concepts. https://kubernetes.io/docs/concepts/.

[40] AWS.inc. 2019 (Hentet 14.05.19). Task definition parameters. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html.

[41] AWS.inc. 2019 (Hentet 07.05.19). Fargate task storage. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/fargate-task-storage.html.

[42] AWS.inc. 2019 (Hentet 10.03.19). Services. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_services.html.

[43] AWS.inc. 2019 (Hentet 1.04.19). services. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_services.html.

[44] AWS.inc. 2019 (Hentet 17.04.19). Aws::ecs::service deploymentconfiguration. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ecs-service-deploymentconfiguration.html.

[45] AWS.inc. 2019 (Hentet 17.04.19). Aws::ecs::service. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html#cfn-ecs-service-desiredcount.

[46] Amazon. 2019 (Hentet 05.03.19). Step scaling policies. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-autoscaling-stepscaling.html.

[47] AWS.inc. 2019 (Hentet 18.04.19). Using the awslogs log driver. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using_awslogs.html#specify-log-config.

[48] AWS.inc. 2019 (Hentet 08.04.19). Task networking in aws fargate. https://aws.amazon.com/blogs/compute/task-networking-in-aws-fargate/.

[49] AWS.inc. 2019 (Hentet 9.04.19). Service load balancing. https://docs.aws.amazon.com/AmazonECS/latest/userguide/service-load-balancing.html.

[50] AWS.inc. 2019 (Hentet 09.04.19). Application load balancer. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html.

[51] AWS.inc. 2019 (Hentet 09.04.19). Nettwork load balancer. https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html.

[52] AWS.inc. 2019 (Hentet 13.4.19). Create target group. https://docs.aws.amazon.com/elasticloadbalancing/latest/APIReference/API_CreateTargetGroup.html.

[53] AWS.inc. 2019 (Hentet 14.04.19). What are aws waf. https://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html.

[54] AWS.inc. 2019 (Hentet 14.05.19). What is amazon cloudwatch events? https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/WhatIsCloudWatchEvents.html.

[55] AWS.inc. 2019 (Hentet 18.04.19). What is amazon cloudwatch logs. https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html.

[56] AWS.inc. 2019 (Hentet 10.04.19). Amazon ecr. https://aws.amazon.com/ecr/.

[57] Patel, R. 2017 (Hentet 19.04.19). Comparing container image registries- dockerhub, amazon ec2, and jfrog artifactory. https://www.nirmata.com/2017/03/14/comparing-container-image-registries-dockerhub-amazon-ec2-and-jfrog-artifactory/.

[58] Atlassian. 2019 (Hentet 13.04.19). Variables in pipelines. https://confluence.atlassian.com/bitbucket/environment-variables-794502608.html.

[59] AWS.inc. 2019 (Hentet 2.4.19). Variables in pipeline. https://confluence.atlassian.com/bitbucket/variables-in-pipelines-794502608.html.

[60] Support, A. 2017 (Hentet 28.03.19). Bitbucket deployments guidelines. https://confluence.atlassian.com/bitbucket/bitbucket-deployments-guidelines-941599590.html.

[61] Amazon. 2019 (Hentet 15.04.19). Aws identity and access management (iam). https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html.

[62] AWS.inc. 2019 (Hentet 26.04.19). Aws security best practices. https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf.

[63] Amazon. 2017 (Hentet 15.03.19). Iam best practices. https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html.

[64] Amazon. 2019 (Hentet 10.04.19). Security groups for your vpc. https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html#VPCSecurityGroups.

[65] dtan4. 2019 (Hentet 21.04.19). Terraforming. https://github.com/dtan4/terraforming.

[66] HashiCorp. 2019 (Hentet 14.05.19). Providers. https://www.terraform.io/docs/providers/.

[67] Pittet, S. 2018 (Hentet 12.05.19). Continuous integration vs. continuous delivery vs. continuous deployment. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[68] Twistlock. 2019 (Hentet 14.05.19). Protection for hosts, containers, and serverless. https://www.twistlock.com/.

A Ordliste

Template

Engelsk ord for mal. I sammenheng med denne rapporten har det betydningen konfigurasjonsmal eller konfigurasjonsfil, og benyttes om hverandre.

Repository

Engelsk ord, ofte forkortet til *Repo*. Det er et oppbevaringssted for kode.

Merge

Engelsk ord. Betyr å sammenflette kode.

Push

Engelsk ord. Brukes om å dytte kode ut til oppbevaringssted.

Pull

Engelsk ord. Brukes om å hente ned kode fra oppbevaringssted.

Image

Engelsk ord. En fil som er et bilde av spesifikk data.

VCS

Engelsk forkortelse for Version Control System. På norsk versjonskontrollsystem.

Produksjon

Ord brukt når en applikasjon blir tilgjengelig for kunde/bruker.

Produksjonsangst

Angst for å gjøre endringer i et operativt system, som kan føre til nedetid, eller skade på systemet.

YAML/JSON

Navn på to ulike tekstformater for å representere data i et strukturert format i rene tekstfiler.

Tilstandsløs

Brukes om applikasjoner som ikke lagrer data lokalt.

Policy

Engelsk ord for politikk.

GUI

Engelsk forkortelse for Graphic User Interface. Grafisk brukergrensesnitt.

Runtime

Engelsk ord. Levetiden fra et program eksekveres til det stoppes.

Infrastrukturskjørhet

Ord som beskriver en infrastruktur som er det motsatte av robust.

Epheremal

Engelsk ord for flyktig. Brukes om datalagring, og betyr at lagring kun er midlertidig.

B Statistikk

Figur 17: Statistikk

C Logg

Uke 2 - 7.-13.januar

Den første uka har gått til å planlegge prosjektet. Signering av prosjektavtale og opprettet en gruppeavtale. Gruppen hadde sitt første møte med oppdragsgiver og satt opp en plan for hvordan prosessen videre skulle foregå. Trello, Google drive, og Overleaf ble tatt i bruk.

Uke 3 - 14.-20.januar

Uken har blitt brukt til å skrive prosjektplanen. Mål, rammer, omfanget av rapporten og prosjektorganisering har blitt gjort. Gruppens første veiledningsmøte med Erik Hjelmås ble gjennomført denne uken, her ble det diskutert hva som burde gjøres videre. Boken Infrastructure as Code av Kief Morris ble anbefalt lesning.

Uke 4 - 21.-27.januar

Denne uken startet første sprint. Gruppen dro ned til Escio for sprint-planning med oppdragsgiver Terje Krogstad. Første del av sprinten gikk ut på å gjøre undersøkelser i forhold til fagfeltet programmerbar infrastruktur, finne aktuelle verktøy og tjenester, samt sette seg inn i dagens praksis. Gruppen fortsatte med prosjektplan og fokuserte på å gjøre den ferdig. Verktøyet JIRA ble tatt i bruk.

Uke 5 - 28.-3.februar

Uken gikk for det meste til å skrive ferdig prosjektplan. En risikoanalyse ble foretatt, samt opprettelse av et Gannt-diagram. Prosjektplanen ble ferdigstilt og ble levert på torsdag. Samtidig benyttet gruppen tiden til å lese faglitteratur. Rapporten til Escio ble opprettet.

Uke 6 - 4.-10.februar

Uken gikk med på å lese faglitteratur og om AWS. Gruppen planla omfang og hvordan rapporten skal bygges opp. Innledningen av rapporten ble påbegynt.

Uke 7 - 11.-17.februar

Gruppen har vært hos Escio for en sprint retrospect review og sprint planning for neste sprint. Arbeidsoppgavene som ble fordelt under første sprint var litt overveldende og gruppen ble ikke ferdig med alle arbeidsoppgaver. Neste sprint ble planlagt og gruppen hadde som mål med å bli ferdig med rapporten til Escio fredag 9. Mars.

Uke 8 - 18.-24.februar

Uken gikk med på å skrive på rapport, og det ble fokusert på å beskrive de forskjellige verktøyene og tjenestene gruppen hadde funnet. Vi begynte smått med å utvikle prototypen.

Uke 9 - 25.-3.mars

Gruppen brukte denne uken til å utvikle en kvantifiseringsmetodikk, etter mye arbeid ble det besluttet at dette ikke fungerte og måtte skrote mye av arbeidet. Rapporten skulle bli

levert fredag denne uken, men dette var ikke mulig. Det manglet fortsatt en god del på beskrivelse av verktøy og tjenester, samt komme frem til en konklusjon.

Uke 10 - 4.-10.mars

Denne uke ble sprint 2 avsluttet, arbeidsoppgavene fordelt på JIRA hadde blitt fullført. Sprint 3 ble opprettet og planlegging de neste ukene ble gjort. Sprinten skulle brukes til å utvikle prototypen. Rapport til Escio ble endelig ferdig, og ble sendt til oppdragsgiver på fredag.

Uke 11 - 11.-17.mars

Uken gikk med til å utvikle prototypen, samt å teste forskjellige tjenester i AWS for å få de til å samhandle. Vi opprettet bachelorrapporten og begynte å planlegge hvordan denne skulle utformes. Innledningen til bacheloroppgaven ble gjort og utvikling av prototypen var kommet godt i gang.

Uke 12 - 18.-24.mars

Vi møtte opp hos Escio og viste hvordan vi rullet ut en applikasjon i AWS. På møte ble det diskutert hva som hadde blitt gjort og hvordan gruppen ligger i forholde til planen. Videre jobbet vi med å skrive på bachelor, kapitler om teori, og hvordan dagens praksis fungerte ble gjort.

Uke 13 - 25.-31.mars

Siste sprint ble opprett og begynte å planlegge hva som måtte gjøres før siste visning av prototype. Resten av uka gikk til å skrive på rapporten, kapittel om teknisk dokumentering av implementasjon. Gruppen fortsettet med å utvikle prototypen.

Uke 14 - 1.-7.april

Siste finpuss på prototypen ble gjort, all kode ble kommentert, og vi tok en siste sjekk for å passe på at applikasjonen rullet ut feilfritt. På rapporten fortsetter vi å skrive teknisk dokumentasjon av implementasjon.

Uke 15 - 8.-14.april

Denne uka ble siste sprint avsluttet. Vi viste Escio prototypen og de var veldig fornøyd med resultatet. Resten av uka gikk til å skrive på rapporten, fokuset var rettet mot å skrive ferdig teknisk dokumentering av implementasjon.

Uke 16 - 15.-21.april

Påskeuken gikk til å jobbe videre med prosjektet og få ferdig et utkast til veileder. Kapittelet om implementasjon var omfattende og var hovedfokuset. På fredag ble første utkast sendt til veileder. Vi tok oss fri siste helgen i påsken til vi fikk tilbakemelding fra veileder.

Uke 17 - 22.-28.april

Denne uka fikk vi tilbakemelding fra veileder og dette ble brukt til å rette opp i diverse feil. Vi hadde ikke opprettet en kravspesifikasjon med use cases. Dette var dermed fokuset.

Uke 18 - 29.-5.mai

Uken har gått til å skrive drøfting og konklusjon for oppgaven, samt fortsette med å skrive kapittelet implementasjon. Implementasjon kapittelet er omfattende og vi brukte mye tid til å ferdigstille dette kapittelet.

Uke 19 - 6.-12.mai

Oppgaven ble sendt til veileder for å få en siste tilbakemelding før vi skal levere oppgaven. Denne uka ble brukt til å lese gjennom rapporten, finskrive alle kapitler og gjøre nødvendig endringer.

Uke 20 - 13.-19.mai

Etter tilbakemelding fra veilder ble endringene gjort. Uken har gått til å lese gjennom rapporten en siste gang før den skulle leveres. Oppgaven ble levert med tilhørende vedlegg i Inspera.

C.1 Tidsbruk for prosjekt

Her er det gjort et estimat over medgått tidsbruk per person for prosjektet. I prosjektplanen ble det avtalt fast møtetid fra mandag til torsdag hver uke med oppmøte kl. 09:00 til 17:00. Estimert tidsbruk for hver avtalte møtedag er 7,5 time, med 30 minutter avsatt til lunsj. Det har også blitt arbeidet noe utenfor denne fastsatte møtetiden. Påsken ble brukt til å arbeide med prosjektet, samt enkelte helger. Alle tall gitt i tabell er i timer.

Uke nummer	**Fast gruppearbeid**	**Møte utenom fast arbeidstid**
Uke 2	30	0
Uke 3	30	7.5
Uke 4	30	7.5
Uke 5	30	0
Uke 6	30	0
Uke 7	30	7.5
Uke 8	30	0
Uke 9	30	0
Uke 10	30	0
Uke 11	30	7.5
Uke 12	30	0
Uke 13	30	0
Uke 14	30	0
Uke 15	30	7.5
Uke 16	30	7.5
Uke 17	30	7.5
Uke 18	30	7.5
Uke 19	30	15
Uke 20	15	0
Sum	**555**	**75**
Timer per person	**630**	

Tabell 18: Gjennomsnitt arbeid

D Oppdragsgivers oppgavebeskrivelse

escio

Vi tar ansvaret!

Automatisert staging av containere

Automatisert staging og skalering
av container-baserte applikasjoner

1 Oppdragsgiver og kontaktinformasjon

Oppdragsgiver

Escio AS
Nedre Torvgate 6 (NT6)
2816 Gjøvik

post@escio.no
Tlf: 47 47 25 80

Kontaktperson

Terje Krogstad, prosjektleder
terje@escio.no
Tlf: 959 72 382

2 Beskrivelse

Escio har behov for å styrke rutinene for staging av container-baserte applikasjoner. I de fleste prosjektene bruker vi Docker for virtualisering av applikasjoner og støttesystemer i utviklermiljøene våre. I denne sammenheng har vi et behov for å strømlinjeforme prosessen med å deploye ny kode ut i staging-miljøene våre i AWS (Amazon Web Services).

Prosjektet går ut på gjøre research i forhold til relevante teknologier og systemer som kan støtte denne prosessen og i tillegg sette opp en prototype på et prosjekt som viser en fullautomatisert flyt fra kode i versjonskontrollsystem til ferdig staget applikasjon med last-balansering og automatisk skalering.

Det kan være fornuftig å ta utgangspunkt i Kubernetes for container-orkestrering.

3 Oppgaver / Mål

- Research i forhold til relevante systemer og teknologier
- Rapport med beskrivelse av systemer og tjenester som bygger opp om en fullautomatisert flyt
- Utvikle en prototype basert på en av Escio sine applikasjonskodebaser som viser en fullautomatisert flyt fra kode til en skalerbar applikasjon i AWS

E Prosjektplan

Prosjektplan for
Automatisert Staging av Containere

Jostein Bergslien
Daniel Monsen
Brage Kleven

January 2019

Contents

1 Mål og rammer

1.1 Bakgrunn

Den lokale bedriften Escio på Gjøvik tilbyr blant annet utvikling av applikasjonener og tjenester for sine kunder. Selskapet leier også ut utviklerteam til kunder som trenger hjelp til egne prosjekter i perioder[4]. Escio er et selskap i vekst og har tatt i bruk skytjenester for å sette deres applikasjoner ut i drift for kundene. Bakgrunnen for denne oppgaven er at Escio har behov for å strømlinjeforme prosessene fra ferdig applikasjonskode i versjonskontrollsystemet til en kjørende applikasjon i skytjenesten Amazon Web Services[2]. Bedriften utfører dette idag manuelt, og søker derfor relevante teknologier og verktøy for å automatisere flyten fra versjonskontrollsystemet til en ferdig kjørende applikasjon i skyen. Tjenstene skal kunne skaleres og lastbalanseres automatisk ved behov. Escio ønsker å få hjelp til å etablere et orkestreringsverktøy som forenkler dagens praksis.

1.2 Prosjektmål

Gruppen har delt opp målene i prosjektet i tre kategorier: Resultatmål, effektmål og læringsmål. Resultatmålene representerer hva bedriften Escio vil at vi skal ha ferdigstilt innen prosjektets tidsavgrensning, altså sluttproduktet. Effektmål representerer den målbare virkningen prosjektet vil ha for virksomheten, i form av ønsket endring fra dagens situasjon. Læremål er våres egne mål om hva vi ønsker å lære igjennom arbeidet med dette prosjektet.

1.2.1 Resultatmål

1. Lage en rapport med beskrivelse av relevante orkestreringsverktøy eller løsninger som bygger opp om en automatisert flyt, samt drøfte disse mot bedriftens behov og gjeldene beste praksis i fagfeltet infrastruktur som kode.

2. Sette opp et prototype-miljø som viser en av Escio sine applikasjoner strømme automatisk fra kode i versjonskontrollsystemet til en skalerbar applikasjon i Amazon Web Services.

1.2.2 Effektmål

- DevOps-teamet hos Escio skal bruke mindre tid på manuelle oppgaver som kan automatiseres og få arbeidsflyten forenklet.

- Systemet skal kunne skaleres ved behov og kunne lastbalansere.

- Forbedret oppetid og redundans.

1.2.3 Læringsmål

- Øke egen kunnskap og forståelse i fagområdet.
- Kunne bruke og bygge på kunnskapen i en fremtidig jobb.
- Opparbeide praktiske evner til å sette opp en automatisert infrastruktur med tilpassede egenskaper.

1.3 Rammer

Rammer sier noe om hva som skal med i prosjektet. Oppdragsgiver har gitt oss rammer som vi skal holde oss innenfor under gjennomførelsen av prosjektet. Vi har valgt å dele opp rammene i to deler, på grunn av at de to første oppgavene er sterkt knyttet sammen.

1.3.1 Rammer for del 1 og 2

Deloppgave 1 og 2 går ut på å utforske relevatne orkestreringsverktøy eller løsninger som oppfyller bedriftens- og fagområdes krav til beste praksis. Deloppgave 1 er en ren utforskningsfase, mens del to skal resultere i en rapport om egnede verktøy eller tjenester.

- Vi skal utforskre orkestreringsverktøy og løsninger i henhold til kravene som oppdragsgiver har satt.
- Rapporten til oppdragsgiver skal gi en oversikt over hvilke verktøy eller tjenester som etter vår mening passer best for bedriften ut fra de gitte krav.
- Formatet på rapporten bestemmer vi selv.

1.3.2 Rammer for del 3

- Kun stateless-applikasjoner skal kjøres i skyen.
- Prototypen skal testes med en av Escios eksisterende applikasjoner.
- Prototypen skal implementeres og testes på Escio sin konto i AWS.
- Prototypen skal legges til rette for at Bitbucket Pipline brukes.

2 Omfang

Omfanget av oppgaven beskriver hva oppgaven prøver å løse. Dette innebærer å definere problemområde og å avgrense problemet til en oppgave vi kan løse i løpet av prosjektperioden.

2.1 Problemområde

Informasjonsteknologi hjelper organisasjoner og dens ansatte å nå sine mål. Når systemene fungerer etter hensikt produserer de verdi, enten direkte eller indirekte. Med dette knyttes IT-drift tett sammen med organisasjonens verdiskapning. Informasjonssamfunnet og skytjenestenes raske fremmars på 2000-tallet endret arbeidsmetodikken fra konvensjonell serverdrift, og ledet til at det oppsto problemer mellom programvareutviklere og driftsavdelinger. Dette førte til at partene satte seg ned for å finne løsninger på problemene som oppsto, og samarbeidet resulterte i "The DevOps movement". DevOps[3] kan beskrives som et sett av praksiser som automatiserer prosessene mellom utviklere og driftavdelinger i den hensikt at tjenester kan bygges, testes og settes ut i produksjonsmiljøet raskere og mer pålitelig. Målet med med DevOps-samarbeidet er å forkorte systemutviklingslivssyklusen samtidig som man leverer nye funksjoner, utfører vedlikehold og oppdaterer tjenesten i tråd med organisasjonens mål.

De fleste organisasjoner i dag har tatt i bruk IT-automatiseringsverktøy og dynamiske infrastrukturplattformer i form av private eller offentlige skytjenester. Som følge av dette har "infrastruktur som kode" fått en nøkkelrolle som beste praksis i utførelsen av DevOps[1]. Automatisering tar sikte på å redusere feil sammenlignet med manuelle oppsett, og tillater at programvare kan settes ut i produksjon med større fleksibilitet, mindre nedetid og til en lavere kostnad for organisasjoner enn tidligere.

Automatisering av kontinuerlig integrasjon og kontinuerlig levering (CI/CD)[5] av programvare, stiller krav til valg av riktige verktøy. I dag finnes det mye og velge i. Ulike verktøy og tjenester har ulike bruksområder, egenskaper og funksjonalitet. Problemet blir å finne den rette løsningen for vår oppdragsgiver.

2.2 Problemavgrensning

Problemavgrensning reflekterer hvilke deler av fagområdet som ikke skal tas med i prosjektet eller som er utenfor våres omfang.

- Bitbucket Pipelines benyttes for testing og bygging av applikasjonene. Dette er en teknologi som ligger godt til rette for å automatisere flyten til applikasjonene og dermed vil det ikke bli gjort forsøk på å erstatte denne teknologien.

- Escio benytter idag Amazon Web Services (AWS), og denne platformen vil bli benyttet under vårt prosjekt.

- Det vil ikke bli gjort forsøk på å implementere et orkestreringsverktøy fra grunn av. Vi ønsker å bygge på tjenester tilgjengelig på markedet i dag.

- Håndtering av databaser i form av lagring og backup av data vil ikke være en del av oppgaven.

- Kun Escios ferdigutviklede applikasjoner skal testes i prototypemiljøet.

2.3 Problemstilling

Hvilke tjenester eller verktøy egner seg best for å automatisere flyten for utrulling av Escio's web-applikasjoner, og hvilke orkestreringsverktøy eller løsninger egner seg best i forhold til bedriftens- og fagområdets krav til beste praksis?

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

Oppgaven skal utføres av gruppens tre medlemmer. For å forsikre oss om at gruppearbeidet fungerer har vi definert regler, ansvarsområder og roller for hvert medlem.

Overordnet ansvarsforhold:

- Gruppeleder: Jostein Bergslien.
  - Ansvar for å opprettholde kontakt med oppdragsgiver og veileder.
  - Ansvarlig redaktør for tekst produsert under arbeidet.
  - Ansvar for å følge opp at tildelt arbeid blir utført, og at det skrives logg fra hvert møte.
- Nestleder: Daniel Monsen.
  - Ansvar for JIRA og Trello.
  - Ansvar for at bestilling av grupperom blir gjort ved hvert møte.
  - Sørger for at alle har arbeidsoppgaver å jobbe med.
- Gruppemedlem: Brage Kleven.
  - Ansvarlig Scrum-master.

- Ansvar for utvikling av prototype.
- Ansvar for kommunikasjonen med driftsteamet ved Escio.

Hvert medlem i gruppen har felles ansvar for å utføre oppgaver, produsere merverdi for oppdragsgiver, øke egen kunnskap i fagfeltet, samt styrke et godt gruppearbeid og overholde gruppens fastsatte regler.

3.2 Rutiner og regler i gruppa

Hvert møte skal gruppen gjennomgå følgende agenda:

- Sjekke status og bestille grupperom to uker frem i tid.
- Gjennomgå arbeidsoppgaver i Trello og JIRA
- Se over utført arbeid, og planlegge videre arbeid.
- Fordele nye arbeidsoppgaver, eller gjøre nødvendige justeringer.
- Ta backup av LaTeX-prosjektet.

Se vedlagt dokument av grupperegler her: 7.1

4 Planlegging, oppfølging og rapportering

4.1 Hovedinndeling av prosjektet

Første del av januar blir brukt til å gjøre seg kjent med oppgaven, sette seg inn i problemstillingen, og opprette dialog med oppdragsgiver og veileder. Vi vil også benytte denne tiden til å arbeide med prosjektplanen, samt å etablere faste rutiner, møtedager og grupperegler. Videre vil vi gjøre oss kjent med aktuelle hjelpeverktøy og programvare som skal brukes under prosjektet, samt gjøre våre egne faglige undersøkelser. Hovedinndelingen av prosjektet er satt opp til fire sprinter på 2 uker hver. Mellom disse sprintene har vi to dager der vi kan arbeide med forefallende arbeid eller gjøre videre undersøkelser av aktuelle tema eller problemstillinger som oppstår under utførelsen av prosjektet.

4.1.1 Prosesstyringsverktøy

Escio ønsker at vi følger den smidige systemutviklingsmetodikken Scrum. Argumentet for dette er at bedriften bruker smidig rammeverk for all sin utvikling og at vi har noe erfaring med dette fra tidligere. Product-Owner er Escio, og våres oppdragsgiver og kontaktperson er Terje Krogstad. Gruppen har bestemt at Brage Kleven blir Scrum-master, og at Jostein Bergslien og Daniel Monsen har status som utviklere i Scrum-teamet.
Hver sprint innledes med Scrum-planning møte, der gruppen sammen med kontaktpersonen i Escio definerer konkrete mål for hver sprint. Arbeidsoppgaver Scrum-teamet skal jobbe med defineres og legges i en Product-Backlog i

verktøyet JIRA. For hvert delmål/oppgave, benyttes estimeringsverktøyet Planning-Poker. I estimeringsprosessen velges først en referansevanskelighetsgrad, som alle i teamet kjenner. Videre estimeres hvert delmål fra Product-Backlog mot den felles kjente referansen. Når en oppgave er estimert, må personen med den laveste og høyeste vanskelighetsgrad forkare hvorfor de har estimert som de har gjort. Når diskusjonen er ferdig, estimeres det på nytt og resultatet av den estimerte vanskelighetsgraden noteres for det spesifikke delmålet i JIRA. Prosessen gjentas til alle delmålene i Product-Backlog er estimert.

4.2 Plan for statusmøter og beslutningspunkter i perioden

Vi har avtalt møte med NTNU veileder Erik Hjelmås hver onsdag, fra kl. 10.00 - 10.30. De avtalte møtetidene kan endres ved andre prioriteter, men vi ønsker å ha tett dialog med veileder under hele prosjektperioden.

Møter med oppdragsgiver:

- Sprint 1 starter den 07.02.19.
- Sprint 2 starter den 28.02.19.
- Sprint 3 starter den 21.03.19.
- Sprint 4 starter den 11.04.19.

Hvert sprint avsluttes med Sprit-retrospect/review møter på følgende datoer:

- Sprint 1 avsluttes den 07.02.19.
- Sprint 2 avsluttes den 28.02.19.
- Sprint 3 avsluttes den 21.03.19.
- Sprint 4 avsluttes den 11.04.19.

5 Organisering av Kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

All kode, script og annen dokumentasjon produsert eller funnet i løpet av prosjektperioden skal dokumenteres. Kode skal kommenteres og følge normal kodepraksis, samt sjekkes for sårbarheter ved bruk av relevante verktøy.

5.2 Konfigurasjonsstyring

- BitBucket skal brukes for versjonskontroll av kildekode.
- JIRA blir brukt for å holde oversikt over oppgaver og fremdrift i prosjektoppgaven.
- Oppgaven skrives i ShareLaTeX (Overleaf), som har versjonskontroll.

5.3 Risikoanalyse

Vi har identifisert og analysert følgende risiko ved prosjektgjennomføringen og har benyttet risikomatrise for å knytte risikoene mot sannsynlighet og konsekvens. Tilslutt har vi planlagt tiltak mot de risikoene som vi kan gjøre noe med. Våres mål med tiltakene er å få alle risikoer så nær grønn sone som mulig.

- Grønn sone: Risiko der sannsynlighet og konsekvens gir en risikoscore innenfor de grønne sonene, der risikoen for prosjektet er definert som lav. Vi ser på identifiserte risikoer som estimeres til grønn sone som akseptable. Tiltak er ikke nødvendig, men bør vurderes dersom dette kan kansellere risikoen helt.
- Gul sone: Risiko der sannsynlighet og konsekvens gir en risikoscore innenfor de gule sonene, der risikoen for prosjektet er definert som moderat. Tiltak bør etableres der identifiserte risikoer estimeres til gul sone. Dersom ikke ytterligere tiltak kan etableres basert på en kost-nytte vurdering må de identifiserte risikoene i gul sone aksepteres.
- Rød sone: Risiko der sannsynlighet og konsekvens gir en risikoscore innenfor de røde sonene, der risikoen for prosjektet er definert som ikke akseptabel. Tiltak må etableres der identifiserte risikoer estimeres til rød sone, som skal resultere i en redusert risikoscore.

5.3.1 Identifiserte risikoscenarioer

Teknologi

1. Tap av data. Eks. ShareLaTeX prosjektet med alt vi har skrevet blir slettet eller utilgjengelig, som fører til at vi ikke blir ferdige eller får levert.
2. PC eller mobiltelefoner blir ødelagt/stjålet som fører til at vi får problemer arbeidet eller kommunikasjon.
3. Prototype/okestreringsverktøy skaper problematikk i bedriftens infrastruktur.

Forretningsmessig

4. Prototypen blir ikke ferdigstilt før fristen.
5. Sluttresultatet tilfredsstiller ikke kravene til oppdragsgiver.
6. Brudd på taushetsplikt.

Prosjekt

7. Noen i gruppen blir langvarig syke, slik at de ikke kan delta i arbeidet.
8. Uenigheter i gruppen fører til dårlig arbeidsmoral og dårlig result.

9. Mangel på kompetanse innad i gruppen.
10. Uforutsette hendelser som fører til utsettelse av arbeidsoppgaver.
11. Dårlig kommunikasjon mellom gruppa og oppdragsgiver fører til mistolkning av oppgaven.
12. Dårlig organisering arbeidet fører til dårlig resultat.

5.3.2 Estimert risiko før tiltak

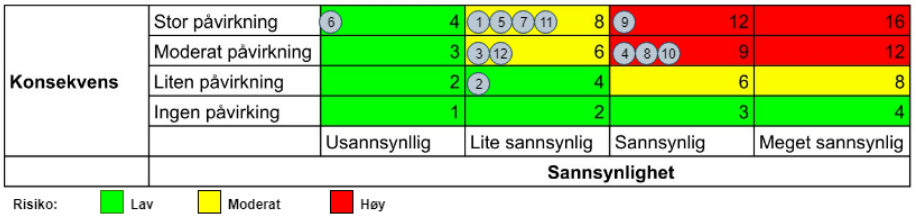


Figure 1: Risiko før tiltak

5.3.3 Risikoscenarioer med tiltak

ID	Scenario	Tiltak
1	Tap av data. Eks. ShareLaTeX prosjektet med alt vi har skrevet blir slettet eller blir utilgjengelig, som fører til at vi ikke blir ferdige eller får levert.	Det blir daglig gjort en lokal backup av prosjektet til en Raspberry Pi. Annen dokumentasjon lagres i Google Drive og på en lokal enhet.
2	PC eller mobiltelefoner blir ødelagt/stjålet som fører til at vi får problemer med arbeidet eller kommunikasjon.	Ingen tiltak, aksepterer risikoen.
3	Prototype/orkestreringsverktøy skaper problematikk i bedriftens infrastruktur	Tildelt eget arbeidsmiljø som er skilt fra Escio eget produksjonsmiljø.
4	Prototypen blir ikke ferdigstilt før fristen.	Opprettholde god dialog med oppdragsgiver og veileder. Opprettet krav til arbeidstid for alle gruppemedlemmene. Se grupperegler.
5	Sluttresultatet tilfredsstiller ikke kravene fra oppdragsgiver.	Holde en god dialog med oppdragsgiver. Opprettet krav til arbeidstid for alle gruppemedlemmene. Se grupperegler.

6	Brudd på taushetplikt.	Signert taushetserklæring med Escio. Alle i gruppen er klar over konsekvensen av en brutt taushetserklæring.
7	Noen i gruppen blir langvarig syke, slik at de ikke kan delta i arbeidet.	Gi beskjed til veileder og oppdragsgiver, og sammen finne gode løsninger på problemet.
8	Uenigheter i gruppen fører til dårlig arbeidsmoral og dårlig resultat.	Saklige diskusjoner. Fokus på problemløsning. Ingen personangrep.
9	Mangel på kompetanse innad i gruppen.	Søke hjelp i relevant litteratur eller med fagfolk, bruke veileder og arbeidsgiver. Arbeide minimum 30 timer pr. person, hver uke. Fokus på problemløsning.
10	Uforutsette hendelser som fører til utsettelse av arbeidsoppgaver.	Følge prosjektplan og gruppereglene. Etablere eventuelle møtetider utenom normal fremdriftsplan dersom dette blir nødvendig.
11	Dårlig kommunikasjon mellom gruppa og oppdragsgiver fører til mistolkning av oppgaven.	Etablert jevnlige møter og god bruk av arbeidsverktøyene JIRA og Trello.
12	Dårlig organisering av arbeidet fører til dårlig resultat.	Hver enkelt gruppemedlem sørger for å utføre arbeidsoppgaver ihht. arbeidsverktøyene JIRA og Trello.

5.3.4 Estimert risiko etter tiltak

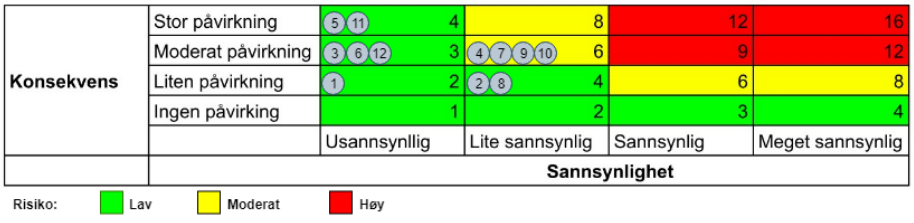
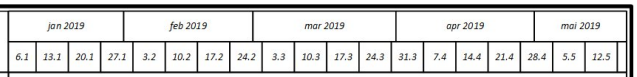


Figure 2: Risiko etter tiltak

6 Plan for gjennomføring

6.1 Gannt-diagram

ID	Task Name	Start	Finish	Duration
1	Prosjektavtale, gruppeavtale og regler.	07.01.2019	16.01.2019	8d
2	Møte med arbeidsgiver Escio	09.01.2019	09.01.2019	1d
3	Gjøre dypdykk i emnet, ved å lese Infrastructure as Code av Kief Morris.	07.01.2019	31.01.2019	19d
4	Utarbeiding av prosjektplan	07.01.2019	31.01.2019	19d
5	Sprint 1 – Planning Meeting	22.01.2019	22.01.2019	1d
6	Sprint 1 – Kravspesifisering og research	23.01.2019	06.02.2019	11d
7	Sprint 1 – Retrospect/Review Meeting	07.02.2019	07.02.2019	1d
8	Kvalitetssikring av research	08.02.2019	11.02.2019	2d
9	Sprint 2 – Planning Meeting	12.02.2019	12.02.2019	1d
10	Sprint 2 – Rapport om egnede orkestrer-ingsverktøy ihht. kravspesifisering	13.02.2019	27.02.2019	11d
11	Sprint 2 Retrospect/Review Meeting	28.02.2019	28.02.2019	1d
12	Forberedelse til implementasjon av orkestreringsverktøy	01.03.2019	04.03.2019	2d
13	Sprint 3 Planning Meeting	05.03.2019	05.03.2019	1d
14	Sprint 3 – Implementasjon av orkestreringsverktøy.	06.03.2019	20.03.2019	11d
15	Sprint 3 Retrospect/Review Meeting	21.03.2019	21.03.2019	1d
16	Testing av orkestreringsverktøy.	22.03.2019	25.03.2019	2d
17	Sprint 4 Planning Meeting (Escio)	26.03.2019	26.03.2019	1d
18	Sprint 4 – Automatisere flyt med skalering.	27.03.2019	10.04.2019	11d
19	Sprint 4 Retrospect/Review Meeting	11.04.2019	11.04.2019	1d
20	Testing og Oppgaveskriving	12.04.2019	15.05.2019	24d



6.2 Milepæler og beslutningspunkter

- 01.02.19 levert prosjektplanen.
- 07.02.19 utført kravspesifisering og research av egnede orkestreringsløsninger som bygger opp om en automatiserbar og skalerbar infrastruktur.
- 28.02.19 utført rapport om egnede orkestreringsverktøy som oppfyller kravspesifiseringen.
- 21.03.19 implementert valgt orkestreringsverktøy basert på rapport.
- 11.04.19 ferdig med prototype av en automatisert flyt.
- 15.05.19 ferdig testet prototype og levert oppgaven.

References

[1] Kief Morris. *Infrastructure as Code.* [*Managing Servers in the Cloud*]. O'Reilly Media Inc, USA ,2016.

[2] Amazon Web Services
`https://aws.amazon.com`

[3] Atlassian: DevOps: Breaking the Development-Operations barrier,
`https://www.atlassian.com/devops`

[4] Escio: Vi hjelper selskaper å lage gode digitale tjenester,
`https://www.escio.no/om-escio`

[5] Wikipedia: CI/CD
`https://en.wikipedia.org/wiki/CI/CD`

7 Vedlegg

7.1 Gruppeavtale

Reglement for bacheloroppgave

§1.0.0 **Møter:**

§1.1.0 **Tidspunkt**:

§1.1.1 Møter foregår ukentlig (kl 9:00 - 17:00) henvist til de dagene på timeplan i trello (man-tors).

§1.1.2 Tidspunkt kan endres ved gruppeavstemning eller ved spesielle tilfeller som må avklart med gruppen i forkant.

§1.2.0 **Organisasjonsstruktur:**

§1.2.1 Møter avholdes i grupperom på skolen eller anvist sted. Tider for Gruppemøte skal bli kommunisert på kommunikasjonsplattformene.

§1.2.2 Gruppen skal forholde seg til kommunikasjons plattformene: Trello, Jira, Facebook og E-mail.

§1.2.3 Organisering av oppgaver blir gitt til vedkommende person via Trello, alle oppgavekravene skal følges når det gjelder tidsfrister og skal kvalitets kontrolleres av et annet gruppemedlem. (Se ansvarsforhold prosjektplan).

§1.2.4 Om oppgaven som utføres er av teknisk sort, skal det dokumenteres hva denne løsningen gjør og hvordan denne løsningen implementeres.

§1.2.5 Sensitiv informasjon i form av gruppearbeid skal lagres i Sharepoint.

§1.3.0 **Oppmøtekrav**

§1.3.1 Møt opp på møtene (se paragraf § 1.1.0)

§1.3.2 Ved mer enn 15 minutter forsinkelse, vil det bli gitt advarsel. Etter gjentagende forsentkomminger vil det bli tatt opp i felleskap og avklart sanksjoner.

§1.3.3 Ved sykdom eller fravær plikter gruppemedlemmer å melde av på forhånd om de ikke kan delta (helst 24 timer). Fraværet skal dokumenteres.

§1.3.4 Om nødvendig kan man møte opp via Skype/Discord om man ikke har mulighet til å møte opp i person. Dette skal bare brukes som nødløsning under sykdom eller andre spesielle tilfeller.

§1.3.5 Dersom et medlem ikke møter opp 3 møter sekvensielt, vil sanksjoner bli gjennomført og veileder kontaktes.

§1.3.6 Fraværsmengden skal ikke overskride 20%. Om et medlem har mer fravær enn dette skal det diskuteres hvilke sanksjoner som skal iverksettes.

§2.0.0 **Arbeidskrav:**

§2.1.0 **Utsettelse:**

§2.1.1 Det er lov til å utsette arbeid så lenge dette er godkjent under ett gruppemøte, eller ved å få godkjenning av gruppelederen. Ny frist skal etableres omgående.

§2.2.0 **Arbeidsmengde:**

§2.2.1 Alle gruppemedlemmer plikter til å gjøre de tildelte arbeidsoppgavene, samt å arbeide minimum 30 timer per person i uka, gjennom hele prosjektperioden. Ytterligere krav til arbeid kan tilkomme ved behov.

§3.0.0 **Sanksjoner:**

§3.1.0 **Konsekvenser:**

§3.1.1 Om man ikke følger reglene risikerer minste sin plass i gruppen etter samtaler med veileder.

§4.0.0 **Regelendringer:**

§4.1.0 Disse reglene kan forandres, men da skal alle i gruppen være enige og alle skal bli opplyst om forandringene som blir gjort.

Navn: Daniel Monsen **Dato:** 28.01.19

Navn: Jostein Bergsliem **Dato:** 28.01.19

Navn: Brage V. Kam **Dato:** 28.01.19

7.2 Prosjektavtale

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Terje Krogstad,
Escio AS (oppdragsgiver), og

Daniel Monsen 470129
Jostein Bergslien 473136
Brage Veiseth Kleven 470813 (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 07.01.19 til 20.05.19.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Erik Hjelmås

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): Jostein Berglien dato 09.01.19

Bryge V. Klemm dato 09.01.19

Daniel Monsen dato 09.01.19

_____ dato _____

Oppdragsgiver (signatur): Turje Vestad dato 9/1 2019

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

F Escio rapport

Tjenester og verktøy som støtter automatisering

Brage Kleven
Daniel Monsen
Jostein Bergslien

Mars 2019

Contents

1 Sammendrag

Rapporten har beskrevet skytjenestemodeller og relaterte AWS-tjenester. Det er i tillegg beskrevet og drøftet enkelte utvalgte verktøy fra andre produsenter i henhold til sine respektive bruksområder. Fellenevneren for disse er at de kan være ledd i å bygge oppe en automatisert flyt. Rapporten gir blant annet leseren en oversikt over aktuelle AWS-tjenester og hvor i kjeden disse kan benyttes i en CI/CD-pipeline. Verktøy for infrastrukturorkestrering, serverkonfigurasjon og containerorkestrering har blitt omtalt i hvert sitt kapittel.
Siden verktøyene omtalt i rapporten måtte oppfylle sine respektive krav for å bli beskrevet, resulterte dette også i at verktøyene i all hovedsak egner seg til formålene og bruksområdet. Det ble derfor vanskelige og kvantifisere disse på en tilfredstillende måte med den tid som sto til rådighet. På dette grunnlag ble det besluttet at verktøy og tjenesters kompatibilitet mot AWS, god dokumentasjon og størrelse på brukermiljø, samt enklest mulig operasjon for driftspersonell skulle vektes høyest.
Slutningene trukket for skytjenestene omtalt i rapporten er at AWS tilbyr godt egnede kandidater til å forenkle og automatisere driftsoppgaver, og kan med dette forbedre dagens flyt. Fordelen med å benytte AWS-tjenester er at bedriften i dag kjører majoriteten av sine applikasjoner i AWS. Tjenestene som tilbys er svært godt dokumentert og har bred støtte i AWS-miljøet. Det tilbys også omfattende kundeservice, dersom bedriften får behov for dette. Ulempen ved bruk av AWS-tjenestene er at bedriften knytter seg sterkere til én leverandør.
For orkestrering av infrastrukturer ble to verktøy beskrevet. HashiCorp Terraform, og AWS CloudFormation. Fordelen med CloudFormation er at dette verktøyet er optimalisert for AWS, og har best støtte blant relaterte tjenester innad i miljøet. Terraform er i motsetning et bedre alternativ dersom Escio på sikt ønsker å orkestrere ressurser på tvers av skytjenesteleverandører.
Escio benytter i dag Ansible til serverkonfigurasjon. Det ble sett på to alternativer, Chef og Puppet. Alle disse verktøyene er godt egnet til formålet, og oppfyller alle krav. Tjenesten AWS Fargate har blitt anbefalt, derfor er ikke et server-konfigurasjonsvertkøy nødvendig.
For container-orkestrerngsverktøy ble to verktøy beskrevet, Docker Swarm og Kubernetes. Av de to førstnevnte kandidatene, anbefales Docker Swarm. Dette fordi det er et enklere verktøy og betjene, har god dokumentasjon og et stort brukermiljø. For orkestrering av containere har likevel ECS blitt anbefalt, dette fordi tjenesten støtter Fargate og enkel å bruke.
Under konklusjon i denne rapport er det beskrevet en mulig løsning til en prototype for CI/CD. Denne prototypen baserer seg på bruk av Bitbucket pipelines, CloudFormation, ECS og Fargate.

2 Introduksjon

Rapporten er produsert for bedriften Escio og skal gi beskrivelse av relevante verktøy og tjenester som bygger opp om en automatisert flyt i henhold til gjeldende beste praksis i fagfeltet infrastruktur som kode. Rapporten skal være et ledd i oppbyggingen av en prototype som viser en av Escio sine applikasjoner flyte fra kode i versjonskontrollsystemet Bitbucket til en ferdig kjørende applikasjon i skytjenesten AWS. Systemet skal kunne skalere og lastbalanseres automatisk basert på trafikk.

3 Problemstilling

Rapporten skal gi en beskrivelse av relevante verktøy og tjenester som bygger opp om en automatisert flyt i henhold til gjeldende beste praksis i fagfeltet infrastruktur som kode, samt gjenspeile kandidatenes bruksområder.

4 Kravspesifikasjon

For å sikre at de verktøy og tjenester som blir beskrevet i denne rapport er i samsvar med prinsipper og mål i fagområdet programmerbar infrastruktur, stilles det krav ved både valg av skyptjenestemodeller og relaterte verktøy benyttet til formålet. I tillegg kommer krav fra oppdragsgiver.

4.1 Krav Escio

- Verktøy/tjeneste skal støttes av Amazon Web Services (AWS).
- Bitbucket benyttes som versjonskontrollsystem for selskapets applikasjoner.
- Verktøy og tjenester skal være godt dokumentert, ha et aktivt brukermiljø og være enkelt å lære og i bruk.

4.2 Krav til dynamiske infrastrukturplattformer

Hensikten med infrastruktur som kode er å behandle infrastrukturer som programvare. Dette krever at en dynamisk infrastrukturplattform[1] må tilfredsstille enkelte kriterier.

Kriterier	**Forklaring**
Programmerbar	En dynamisk infrastrukturplattform må være programmerbar. De fleste plattformer har i dag et brukergrensesnitt, men det skal også være mulig å benytte script, programvare og verktøy til å kommunisere med plattformen. Til dette trenges et programmeringsgrensesnitt (API). Eks. REST-based API.
Behovsbasert	Dynamiske infrastrukturplattformer skal gjøre det mulig å tildele og fjerne ressurser på et øyeblikk.
Selvbetjent	En bruker skal raskt kunne tilpasse og skreddersy ressurser etter eget behov.

[1] https://en.wikipedia.org/wiki/Dynamicinfrastructure

4.3 Krav til verktøy som støtter automatisering

- Verktøyet må behandle APIer og kommandolinje-verktøy som primær form for kommunikasjon.
- Skript skal kunne trigges og eksekveres av andre skript eller verktøy.
- Kommandolinjeverktøy skal kunne kjøre selvstendig i et shell eller batch skript uten brukerinteraksjon.
- Verktøy skal kunne konfigureres eksternt.

5 Kvalifiseringsmetodikk

Denne rapporten skal beskrive flere kandidater av ulike typer verktøy og tjenester som bygger opp om en automatisert flyt og som har forskjellige egenskaper og bruksområde. Det finnes et stort antall verktøy og tjenester på markedet i dag. Med den tid som sto til rådighet for utarbeidelse av denne rapporten ble det for omfattende å sette opp og vurdere hvert enkelt verktøy eller tjeneste manuelt. På bakgrunn av dette ble det nødvendig å utarbeide en kvalifiseringsmetodikk for å kunne velge ut relevante tjenester og verktøy for videre beskrivelse. For å verifisere og kvantifisere at kandidatene tatt med i rapporten er relevante og har et større brukermiljø ble det benyttet Google Trends og Stack Overflow Tags. Dokumentasjon finnes under vedlegg.

5.1 Kvalifisering

For at en kandidat skulle bli kvalifisert, måtte kandidaten oppfylle sine respektive krav i kapittel 4, kravspesifikasjon.

- Alle kandidater må oppfylle Escio sine krav i henhold til kapittel 4.1.
- Skytjenestemodeller må oppfylle krav i kapittel 4.2 som omhandler dynamiske infrastrukturplattformer.
- Verktøy/tjeneste må oppfylle krav i kapittel 4.3 som omhandler automatisering.

6 Mål for drøfting

Målet for drøftingen i denne rapporten var å belyse relevante kandidater innenfor sitt respektive bruksområde som bedriften Escio kan dra nytte av i sitt virke. Med nytte menes forbedring, forenkling eller automatisering av arbeidsprosesser relatert til drift av infrastrukturer, eller en kombinasjon av disse. Utgangspunkt

7 Avgrensning

Av hensyn til den tid som sto til disposisjon til utarbeidelse av denne rapport, ble det nødvendig å avgrense omfanget. Rapporten tar ikke med med følgende:

- Versjonskontrollsystemer og pipeline-verktøy. (Escio benytter Bitbucket, som har innebygget pipeline funksjonalitet)
- Kostnader for verktøy og tjenester.
- Skytjenster levert fra andre aktører enn AWS.

8 Definisjoner, teori og praksis

8.1 Automatisering

"Automatisering er teknikken å få systemer til å fungere uten, eller med liten grad av menneskelig medvirking[20]."

8.2 Orkestrering

"Orkestrasjon er den automatiske konfigurasjonen, koordinasjonen og styringen av datasystemer og programvare[21]."

8.3 Provisjonering

I kontekst av denne rapport, benyttes Kief Morris tolkning av provisjonering. "Med provisjonering menes det å gjøre et infrastrukturelement som en server eller en nettverksenhet klar til bruk[2]."
Avhengig av hvilket infrastrukturelement som er gitt, kan dette innebære:

- Tilordne ressurser til elementet.
- Instansiere elementet.
- Installere programvare på elementet.
- Konfigurere elementet.
- Registrere elementet med infrastrukturtjenester.

8.4 Infrastruktur som kode

Infrastruktur som kode, eller programmerbar infrastruktur, er en prosess der man administrerer og provisjonerer datasentre gjennom maskin-leselige definisjonsfiler i stedet for fysisk maskinvarekonfigurasjon, eller ved bruk av interaktive konfigurasjonsverktøy[2].

8.4.1 Prinsippene for infrastruktur som kode

1. Systemene kan enkelt reproduseres.
2. Systemene kan og er tenkt til å kastes etter bruk.
3. Systemene er konsistente.
4. Prosesser er repeterbare.
5. Design er kontinuerlig under endring.

8.4.2 Mål med infrastruktur som kode

- Infrastrukturen støtter utførelse av endringer kontinuerlig.
- Endringer i systemet er rutine, og skal ikke medføre produksjonsangst.
- Ansatte bruker tiden til faglig utvikling og ikke på repeterbare rutineoppgaver.
- Brukere skal enkelt kunne bruke og benytte seg av ressursene systemet tilbyr uten at dem trenger hjelp av IT-tjenesten.
- Driftspersonell kan raskt og enkelt gjenopprette feil, i stedet for å anta at feil kan forhindres helt.
- Utbedringer av applikasjoner eller tjenester kan gjøres ved kontinuerlige iterasjoner.
- Løsninger på problemer blir gjennomført ved implementasjon, testing og målbarhet, i motsetning til å diskutere dem i møter eller i dokumenter.

8.5 VCS for administrering av infrastruktur

Et versjonskontrollsystem (VCS) er helt sentralt i fagfeltet infrastruktur som kode[2].

Egenskaper	**Forklaring**
Sporing av endringer	VCS lagrer endringer i forhold til hva som ble endret, hvem som gjorde det og i kontekst hvorfor det blir gjort en endring, Sporing er essensielt med tanke på feilsøking av problemer.
Tilbakerulling	Når en eller flere endringer lager et problem, er det nyttig å kunne gjenopprette tilbake til fungerende funksjonalitet.
Sammenheng mellom endringer av elementer i infrastrukturen	Når skript, konfigurasjoner og artefakter er samlet i VCS samt relaterer med hverandre gjennom tags eller versjonsnummer, kan det bli brukt for å finne eller løse andre mer komplekse problemer.
Synlighet	Alle på prosjektet kan se endringer som er gjort på VCS, som vil kunne hjelpe situasjonsforståelsen for gruppen. Hvis et problem oppstår, vil brukerne være oppmerksomme på at de siste endringene kan ha forårsaket problemet.
Automatisk trigge handlinger	VCS kan automatisk trigges av handlinger når en endring er gjort. Dette er nødvendig for å få en automatisert CI/CD pipelines.

8.5.1 VCS bruksområde

En VCS bør innholde følgende elementer:

- Konfigurasjonsdefinisjonsfiler (f.eks cookbookes[2], playbooks[3])
- Konfigurasjonsfiler og maler[4].
- Test av kode.
- CI/CD definisjoner.
- Hensiktsmessige skript.
- Kildekode for fungerende applikasjoner.
- Dokumentasjon.

8.6 Kontinuerlig- integrasjon og leveranse (CI/CD)

Kontinuerlig integrasjon(CI) er praksisen ved å regelmessig integrere og teste alle endringer gjort i et system ettersom de blir utviklet[2]. Målet med CI er å raskt avdekke feil, forbedre kvaliteten på programvare og redusere den tiden det tar å validere og utgi den.
Kontinuerlig leveranse(CD) er praksisen hvor endringer i programvare automatisk bygges, testes og klargjøres for produksjon[49]. Kontinuerlig leveranse ekspanderer omfanget av kontinuerlig integrasjon, ved at alle endringer i programvaren distribueres til et testmiljø, produksjonsmiljø eller begge. Kontinuerlig leveranse kan fullautomatiseres.

Fordeler med kontinuerlig leveranse:

- Automatisere prosessen med leveransen av programvare, ved at koden automatisk bygges, testes og klargjøres for produksjonsmiljøet, før den utgis.
- Forbedre utviklernes produktivitet, ved at utviklere frigjøres fra manuelle oppgaver.
- Forbedre kodekvalitet, ved at det oppdages feil på et tidlig stadie gjennom hyppig testing.
- Utgi oppdatering raskere.

[2] https://docs.chef.io/cookbooks.html
[3] https://docs.ansible.com/ansible/latest/user_guide/playbooks.html
[4] https://no.wikipedia.org/wiki/Template

8.6.1 CI/CD Pipeline

CI/CD visualiseres ofte som en "pipeline", hvor ny kode testes gjennom en kjede av steg. Hvert steg i kjeden er strukturert som en logisk enhet i utgivelsesprosessen.

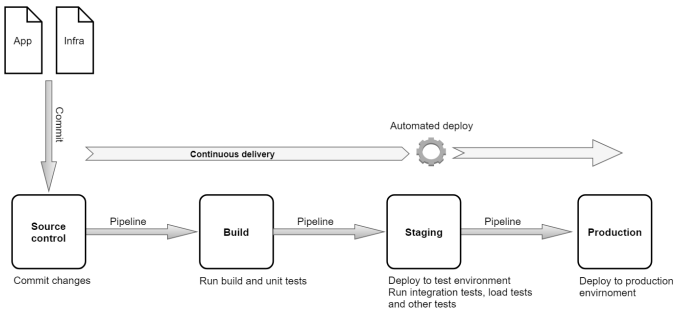


Figure 1: CI/CD Pipeline

9 Skytjenester

9.1 Definisjon

Skytjenester (Cloud computing) muliggjør en allestedsnærværende skalerbar og behovsbasert tilgang til en delt pulje av konfigurerbare databehandlingsressurser over internett (for eksempel: nettverk, servere, lagring, applikasjoner og tjenester) som raskt kan tilrettelegges og frigjøres med minimal ledelsesinnsats eller tjenesteyterinteraksjon.

9.1.1 Egenskaper

Egenskaper	**Forklaring**
Behovsbaserte	Skytjenester blir levert etter hvert som man har brukt for dem. De kan etableres raskt og kunden betjener seg selv etter eget behov (for eksempel servertid eller lagring), uten å måtte involvere leverandøren.
Levert over internett	Tjenesten er tilgjengelig over internett, og kunden får tilgang til standard-mekanismer som støtter bruk av heterogene tynne eller tykke klientplattformer - fra mobiltelefoner, nettbrett og PC-er.
Delte ressurser	Leverandøren kan fordele dataressursene dynamisk etter de ulike kundenes behov. Kundedata kan isoleres og flyttes mellom forskjellige datasentre leverandøren eier innenfor tilgjengelige regioner eller land.
Umiddelbar fleksibilitet	De tjenester eller ressurser kunder bruker kan skaleres opp eller ned etter hva kunden har behov for, og i noen tilfeller automatisk, slik at ressursene i praksis blir opplevd som uendelige.
Betaling etter bruk	Ressursbruken blir målt, kontrollert og rapportert, og er gjennomsiktig for både kunden og leverandøren av tjenesten. Målingen av ressursbruk kan gjøres ved bruk av langring, databehandling/prosesseringkraft, benyttet båndbredde eller aktive brukekontoer.

9.1.2 Leveransemodeller

Leveransemodeller	Forklaring
Privat Sky - Private Cloud	Skyinfrastrukturen er klargjort for eksklusiv bruk av en enkelt organisasjon og deres brukere/kunder. Infrastrukturen kan være eid, styrt og operert av organisasjonen, en tredjepart, eller en kombinasjon av disse. Infrastrukturer kan enten ligge på organisasjonens eiendom, eller utenfor.
Allmenn Sky - Public Cloud	Skyinfrastrukturen er gjort tilgjengelig for allment bruk, og selges på det åpne marked. Infrastrukturen er eid, styrt og operert av en virksomhet, et akademi, eller et myndighetsorgan, eller en kombinasjon av disse. Infrastrukturen ligger på eiendommen til en eller flere av disse.
Gruppesky - Community Cloud	Lignende virksomheter går sammen om egen sky.
Hybrid sky - Hybrid Cloud	Skyinfrastrukturen er en kombinasjon av to eller flere distinkte leveransemodeller som er knyttet sammen gjennom standardisert eller proprietær teknologi som gjør data og applikasjoner portabel.

9.1.3 Tjenestemodeller

- Infrastruktur som tjeneste (IaaS): refererer til en skybasert infrastrukturtjeneste som leverer virtuell teknologi for å hjelpe organisasjoner til å bygge og administrere servere, nettverk, operativ systemer og datalagring [3].
- Platform som tjeneste (PaaS): refererer til en skybasert tjeneste som leverer utviklere med et rammeverk de kan bruke for å utvikle, administrere og kjøre applikasjoner, uten å måtte tenke på å bygge og vedlikeholde infrastrukturen[4].
- Funksjon som tjeneste (FaaS): refererer til en skybasert tjeneste som leverer en platform for brukere til å utvikle, kjøre og administrere applikasjonsfunksjonalitet uten å måtte tenke på å bygge og vedlikeholde infrastrukturen. Oppbygging av denne type platform er å oppnå en serverløs arkitektur, og blir mest brukt til mikrotjenester[5].
- Programvare som tjeneste (SaaS): er en tjenestemodell hvor tjenesteleverandøren distribuerer programvaren til brukere som får tilgang ved å abonnere på applikasjonen. Tjenesteleverandører hoster applikasjonen i sine data sentre og gir adgang til brukere via web eller applikasjon [6].
- Container som tjeneste (CaaS): er en skybasert tjeneste som gir utviklere muligheten til å gi ut, organisere, kjøre, skalere, administrerer eller stoppe

kontainere ved hjelp av leverandørens API eller gjennom en webportal-grensesnitt. som de fleste sky tjenestene betaler du kun for ressursene som blir brukt[7].

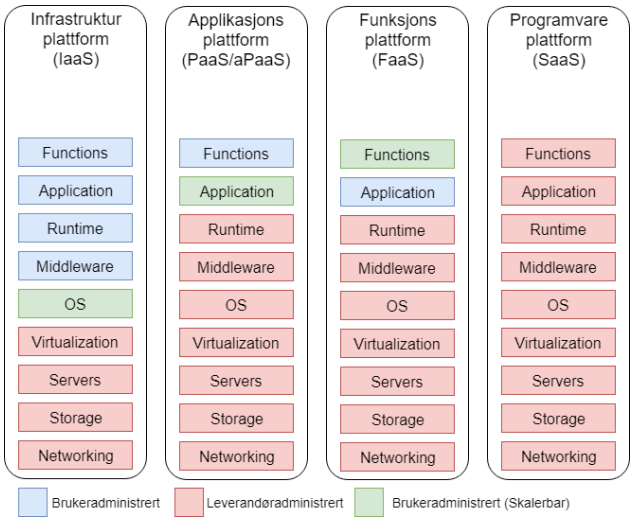


Figure 2: Servicemodeller

9.2 Beskrivelse av tjenestemodellers egenskaper

For valg av tjenestemodell har gruppen sett på hvor mye kontroll Escio ønsker over infrastrukturen. Tjenestemodellene som har blitt vurdert er de tre hovedmodellene IaaS, PaaS og SaaS, samt hybrid versjoner som CaaS og FaaS. Deretter vil det komme beskrivelse og forslag til AWS tjenester som sammenfaller under de forskjellige tjenestemodellene.

Table 5: Beskrivelse av tjenestemodellers egenskaper

Modell	**Fordeler**	**Ulemper**
IaaS	• Leverandør ansvar for hardware • Lett skalerbar • Fleksibel og høy tilpasningsmuligheter • Potensielt lav kostnad • Enkel integrasjon med andre systemer • Ikke vendor lock-in	• Bruker administrerer programvare • Kostnad kan være uforutsigbar • Krever dyktige driftspersonell • Totalkostnad kan bli høy.
CaaS	• Kjører stateless og statefeull applikasjoner • Kan kjøre orksetreringsverktøy (lokalt og skyen) • Kontroll over kontainere • Kjøre komplekse tjenester	• Utviklere trenger kunnskap om kontainere • Vendor lock-in

PaaS	• Leverandør ansvar for hardware og programvare • Utviklere ikke ansvar for infrastruktur • Automatisk skalering • Automatisk oppdatering av programvare • kostnadseffektiv	• Låst til en spesifikk programvare, språk, brukergrensesnitt • Begrenset skalering (bedrift vokser) • Vendor lock-in • Lite fleksibel • Kostnader kan være uforutsigbare
FaaS	• Betaler run time av kode • automatisk skalering • Rask levering av kode • Kostnadseffektiv • passer for mikrotjenester	• Begrenset større applikasjoner • Begrenset til tilstandsløs applikasjoner • Vendor lock-in
SaaS	• Leverendør tar seg av tjenesten • Tilgjengelig over nett-side • Kostnadene er forutsigbare • Skalerbar	• Lite tilpasningsmuligheter • Leverandør tilgang til data • Ikke noe kontroll over tjenesten • Ikke kompatibel med andre tjenester • Vendor lock-in.

9.3 Beskrivelse av AWS-tjenester

9.3.1 IaaS

EC2 Elastic Compute Cloud leverer en skalerbar databehandlingskapasitet i AWS. Ved bruk av EC2 vil du unngå å investere i maskinvare, og kan dermed utvikle og distribuere tjenester raskere. EC2 gir brukeren muligheten til å kjøre et fritt antall virtuelle servere basert på behov, samt administrere sikkerhet, nettverk og lagringsplass samt muligheten til å skalere opp eller ned tjenesten etter brukerens ønske.[33].

S3 Simple Storage Service er en lagringstjeneste basert på objekter som tilbyr høy skalering, tilgjengelighet, sikkerhet og ytelse. Uavhengig av størrelsen på organisasjonen kan brukeren lagre og sikre alt av data innenfor bruksområder som nettsider, mobil applikasjoner, sikkerhetskopi og arkivering. Tjenesten tilbyr enkel funksjonalitet for å organisere data og konfigurere tilgangen til bedriftens ønsker[40].

9.3.2 PaaS

Elastic Beanstalk Elastic Beanstalk er en tjeneste som gir brukeren muligheten til å utvikle og administrere applikasjoner i AWS skyen, uten å måtte tenke på infrastrukturen applikasjonen kjører på. Tjenesten vil hjelpe utviklerne med å gjøre administrering enklere uten å begrense utviklerens kontroll, samt at Elastic Beanstalk automatisk håndterer provisjonering, lastbalansering, skalering og applikasjonens tilstand. Tjenesten støtter applikasjoner skrevet i Go, Java, .NET, Node.js, PHP, Python og Ruby. når brukeren velger å kjøre ut koden vil Elastic Beanstalk bygge applikasjonen og provisjonere ressurser, som f.eks EC2 instanser[37].

Fargate Fargate er en tjeneste hvor brukeren kan bruke ECS til å kjøre containere uten å måtte tenke på å administrere servere eller klynger i en EC2 instans. Med Fargate vil tjenesten håndtere provisjonering, konfigurering og skalering av klynger. Fargate vil ta ansvaret vekk fra brukeren og automatisk velge servere, når klynger skal skaleres og optimalisere bygging av klynger. For å kjøre applikasjonen må brukeren pakke applikasjonen i containere, spesifisere CPU og krav til minne, definere nettverk og IAM[5] regler og starte applikasjonen. Hver enkelt oppgave er isolert og deler dermed ikke, maskinvare, CPU, minne ressurser eller andre nettverk med andre oppgaver[35].

[5]https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

9.3.3 CaaS

ECS Elastic Container Service er en container-orkestreringstjeneste med god skalering og høy ytelse med støtte til docker-orkestrering, samt gi brukerne muligheten til å kjøre og skalere containerbaserte tjenester i AWS. Ved bruk av ECS vil brukeren se bort fra å installere og behandle orkestreringsverktøyet samt administrere og skalere klynger i Virtuelle maskiner med enkle API-kall[34].

EKS Elastic container service for Kubernetes er en administrert tjenester som gjør det enkelt å kjøre Kubernetes på AWS uten å måtte sette opp eller vedlikeholde Kubernetes. Kubernetes er et åpen kildekode plattform som automatiserer levering, skalering og administrering av containerbaserte applikasjoner. For å sikre høy tilgjengelighet bruker EKS å kjøre Kubernetes instanser på tvers av tilgjengelighetsoner, EKS vil automatisk detektere og erstatte defekte instanser samt oppgradere og oppdatere instansene[36].

9.3.4 FaaS

Lambda Lambda er en tjeneste som lar brukeren kjøre kode uten å tenke på provisjonering eller administrering av servere. Tjenesten kjører koden bare hvis den blir tilkalt og skaleres automatisk uavhengig om det er få eller flere tusen forespørsler i sekundet samt betaler brukeren for kjøretiden av koden. Lambda kjører koden på en infrastruktur med høy tilgjengelighet og håndterer seg alt av administrasjon av ressurser som servere, operativ systemet, provisjonering, skalering og loggføring. For å kjøre koden må den støtte en av følgende språk: Node.js, Python, Java, Go, .NET[38].

9.3.5 Andre AWS tjenester

CloudFormation Cloudformation er en tjeneste som hjelper brukeren med å strukturere og sette opp AWS sine ressurser for å gi brukeren mer tid til å fokusere på applikasjonen enn å administrere ressursene. Ved hjelp av manualer kan brukeren beskrive hvilke ressurser den ønsker og tjenesten vil ta seg av provisjoneringen og konfigurering av dine valgte ressurser[39].

CodeBuild Codebuild er en kontinuerlig integrasjonstjeneste som kompilerer kildekoden, kjører enhetstester og produserer programvare pakker som er klare til å distribueres. Tjenesten behandler provisjonering, administrering og skalering av servere, Codebuild skalerer kontinuerlig og bygger applikasjonene parallelt. CodeBuild kommer med ferdigutviklet pakker eller brukeren kan tilpasse sitt eget byggemiljø med egne verktøy[41].

CodeDeploy CodeDeploy er en utleveringstjeneste som automatiserer distribuering av programvare til en rekke skytjenester som EC2, Fargate og Lambda. Tjenesten bidrar til å levere nye funksjoner hyppig, forhindre nedetid og håndterer oppdateringer av brukerens applikasjoner[42].

9.4 Drøfting av tjenester i AWS

Det finnes en rekke aktuelle tjenester i AWS for å bygge opp en CI/CD-pipeline. Tjenestene kan være aktuelle kandidater i målet om å forenkle arbeidsprosesser relatert til drift. Fordelen med å benytte AWS sine tjenester til automatiseringsformål er at disse er godt dokumentert, har bred støtte i AWS-miljøet og tar vekk mye av tiden som går med til vedlikehold av verktøy. Ulempen er at man binder seg tettere til leverandøren, og at dette på sikt kan føre til at det blir vanskeligere å bytte til andre tilbydere på et senere tidspunkt. Under er en illustrasjon over hvilke tjenester som er tilgjengelig i forskjellige steg av en CI/CD-pipeline.

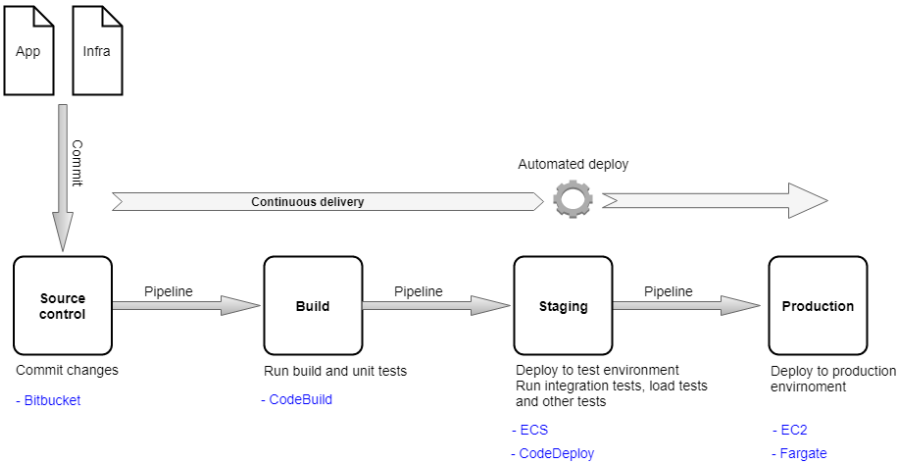


Figure 3: Tilgjengelige AWS-tjenester i en CI/CD-pipeline

10 Infrastruktur-definisjonsverktøy

10.1 Definisjon

Infrastruktur-definisjonsverktøy, ofte kalt orkestreringsverktøy, er et verktøy som blir brukt til å definere, implementere og oppdatere infrastrukturarkitekturen. Infrastrukturen er spesifisert i deklarative konfigurasjon-definisjon-filer. Verktøyet bruker disse filene til å allokere ressurser, modifisere dem, legge til, eller fjerne elementer i infrastrukturen slik at den samsvarer med spesifikasjonen. Orkestreringsverktøyet kommuniserer med skyplattformens brukergrensesnitt (API) til å utføre de aktuelle oppgavene[2].

10.1.1 Konfigurasjonsregistre

Et konfigurasjonsregister er en katalog med informasjon om elementer i en infrastruktur. Registeret holder på informasjon som skript, applikasjoner og tjenester trenger til å kontrollere og integrere med infrastrukturen. Dette er spesielt viktig i en dynamisk infrastruktur siden informasjonen endres kontinuerlig når nye elementer legges til eller fjernes.[2]

Det finnes forskjellige måter man kan implementere konfigurasjonsregistre:

- For enkle infrastrukturer kan orkestreringsverktøyet bruke konfigurasjon-definisjon-filene direkte. Når orkestreringsverktøyet kjører, har det tilgang til informasjon den trenger i konfigurasjon-definisjon-filene. Problemet med denne metoden er at det ikke skalerer godt.
- For mer avanserte infrastrukturer kan man benytte konfigurasjonregister-applikasjoner, som Zookeeper[6] og Consul[7]. Mange server konfigurasjonsverktøy tilbyr også sitt eget konfigurasjonregister, for eksempel Chef Server[8], PuppetDB[9] og Ansible Tower[10]. Disse applikasjonene er designet for å lett la seg integrere med orkestreringsverktøy.
- Lettvektsregistre, kan være en tjeneste som AWS S3 bucket[11], eller en form for versjonskontrollsystem (VCS) der konfigurasjonsfiler legges i et sentralt, delt område. Denne metoden gir enkel administrasjon, er feiltolerant, og er lett og skalere.

[6] https://zookeeper.apache.org/
[7] https://www.consul.io/
[8] https://docs.chef.io/servercomponents.html
[9] https://docs.puppet.com/puppetdb/
[10] https://www.ansible.com/products/tower
[11] https://aws.amazon.com/s3/

10.2 Beskrivelse av verktøy

Gruppen har valgt ut to verktøy godt egnet til bruk i skytjenesten AWS. HashiCorp Terraform og AWS CloudFormation er to verktøyene som oppfyller kravene under punkt 4 i denne rapporten, samt Escio sitt krav om at de skal støttes av AWS. Felles for begge er at brukeren definerer en mal som beskriver en ønsket tilstand for infrastrukturen. Verktøyene oppretter, oppdaterer og sletter skyressurser automatisk slik at den ønskede tilstanden opprettholdes.
For å kunne skille disse verktøyene har vi valgt å sette opp en tabell, slik at det blir enklere å se likheter og forskjeller mellom de to kandidatene.

Egenskaper	**AWS CloudFormation**	**HashiCorp Terraform**
Omfang	CloudFormation støttes kun i AWS, og er kompatibel med de aller fleste tjenestene tilgjenglige i AWS.	Terraform er støttet hos flere skytjenesteleverandører, samt de fleste tjenester disse leverandørene tilbyr[12].
Lisens	Tjenestesten CloudFormation er gratis å bruke i AWS, og det betales kun for ressurser som er i bruk[13].	Terraform er et åpenkildekode prosjekt tilbudt av selskapet HashiCorp og er gratis å bruke. Det må påregnes kostnader for benyttede ressurser hos skytjenesteleverandøren.
Brukerstøtte	Kan kjøpes og dekker CloudFormation. Det finnes flere avtaler å velge mellom.[14]	HashiCorp tilbyr Pro og Entreprise utgave, som gir brukerstøtte mot betaling[15].
Tilstandsstyring	CloudFormation styrer alle ressurser i bruk i infrastrukturen, og holder orden på tilstanden i såkalte "stacks" direkte[16]. Dette forhindrer konflikter ved endring.	Terraform styrer alle ressurser i bruk i infrastrukturen, og lagrer tilstanden på en lokal disk som standard, men kan modifiseres til å benytte AWS S3[17].

[12] https://www.hashicorp.com/partners
[13] https://aws.amazon.com/cloudformation/
[14] https://aws.amazon.com/premiumsupport/plans/
[15] https://www.hashicorp.com/products/terraform
[16] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html
[17] https://www.terraform.io/docs/state/

Modularisering	CloudFormation støtter modularisering av infrastrukturressurser og kan utføres på fler forskjellige måter.	Terraform støtter modularisering ved bruk av pakker med konfigurasjoner som kan styres som en gruppe.
Verifisering ved endring	Tjenesten CloudFormation tilbyr "change sets" som kan brukes til å verifisere endringer før de utføres på infrastrukturen [18].	Terraform tilbyr en kommando "plan" som ved bruk gir en detaljert oversikt over hva som vil bli modifisert på infrastrukturen før endringen utføres [19].
Ventetilstand	CloudFormation støtter bruk av "breakpoints" i konfigurasjonmaler. Dette kan for eksempel brukes til å koordinere oppgaver [20].	Ventetilstander støttes ikke av Terraform, uten modifikasjon.
Rullende oppdateringer	Ved bruk av en oppdateringspolicy, vil CloudFormation kjøre en rullende oppdatering i en "Auto Scaling Group[21]", inkludert en tilbakerulling dersom oppdatering feiler.	Ikke støtte for rullende oppdateringer av "Auto Scaling Group", uten modifikasjon.
Tilbakerulling	Dersom CloudFormation feiler med å modifisere infrastrukturen, ruller den tilbake til den forrige fungerende tilstanden automatisk.	Terraform støtter ikke automatisk tilbakerulling, uten modifikasjon.
Håndtering av eksterne ressurser	Det er ikke mulig å styre eller integrere eksisterene ressurser inn i CloudFormation. Den eneste mulighet er ved bruk av "input parameters" for integrasjon med eksisterende ressurser.	Terrafor støtter import av eksisterende ressurser.

[18] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html
[19] https://www.terraform.io/docs/commands/plan.html
[20] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-waitcondition.html
[21] https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html

10.3 Drøfting av infrastruktur-definisjonsverktøy

Basert på tabellen over, vil det anbefales å benytte tjenesten CloudFormation. Dette er fordi verktøyet er optimalisert for AWS og støtter mange relaterte tjenester. Verktøyet er enkelt å bruke og trenger ikke vedlikehold. Siden Escio alt benytter seg av tjenester i AWS, vil ikke et valg av Terraform ha noe for seg. Dersom Escio på lenger sikt ønsker å flytte infrastrukturen til en annen skyleverandør, eller implementere ressurser fra andre aktører hadde valget måtte bli sett i lys av dette. Valget ville da ha falt på Terraform.

11 Server-konfigurasjonsverktøy

11.1 Definisjon

Er verktøy spesielt utviklet for konfigurasjon av servere. Det benytter eksterne konfigurasjonsfiler med et domenespesifikt språk (DSL) utviklet for serverkonfigurasjon. Verktøyet leser definisjonene i konfigurasjonsfilene og utruster serveren med den relevante konfigurasjonen[2].

Vanlige bruksområder kan være:

- Allokering av maskinvare.
- Partisjonering av diskplass.
- Laste inn operativsystem.
- Laste inn programvare.
- Konfigurere nettverk og DNS.

11.2 Beskrivelse av verktøy

Det finnes en rekke verktøy på markedet i dag som oppfyller kravene i kapittel 4.3. Det ble derfor nødvendig å foreta en utvelgelse. Escio stiller krav til at verktøyene blant annet har god dokumentasjon og et aktivt brukermiljø. På bakrunn av dette ble det besluttet å ta for oss to av det mest brukte verktøyene på markedet. Den aktuelle kandidaten Ansible ble ikke beskrevet, grunnet at Escio bruker dette verktøyet i dag.

11.2.1 Chef

Generelt
Chef ble originalt skapt av Adam Jacob som et verktøy i et konsulteringselskap, her bygget de ende-til-ende kommunikasjon for server/utplasserings verktøy. Det ble i 2009 lansert for første gang offisielt av Chef Inc[14]. I dag brukes Chef som et serverprovisjoneringsverktøy som skal holde serverne konsistente. I AWS OpsWorks ligger Chef Automate, som tilbyr en fullstendig administrativ Chef Automate server og automatiseringsverktøy for kontinuerlig distribusjon, automatisk testing av sikkerhet og et brukergrensesnitt som viser status på dine noder[23].

Dokumentasjon
Chef har en stor mengde dokumentasjon, alt fra installasjon til plugins, og hvordan disse implementeres og brukes[24]. Det finnes flere andre tjenester Chef tilbyr, som også er dokumentert oversiktlig på samme sted.

Chef er godt dokumentert og tilbyr en egen læringsplattform [25] som heter Learn Chef Rally. Plattformen tilbyr oppgaver relatert til Chef sin funksjonalitet, tilpasset det aktuelle miljøet verktøyet skal implementeres under. Tilpasningen kan eksempelvis være hvilken skytjeneste Chef skal implementeres i (Azure, AWS etc.) eller hvilke operativsystemer Chef skal kjøre på (Linux, Windows etc.)[25].

Utvikling
Chef har all sin kildekode åpent i sin Github[26]. Utviklingen av programvaren er basert på et samarbeid mellom selskapet og brukermiljøet. Brukermiljøet kan lett opprette feilmeldinger og løsningsforslag for vurdering. Under prosjektenes repository ligger det milepæler i form av "prosjekter" med tidsfrist for funksjonens implementasjon. F.eks "Self documenting resources" i Chef repository[27].

Arkitektur
Arkitekturen til Chef er delt opp i tre hovedkomponenter: [28]

- Chef Workstation: Lokasjonen der brukeren samhandler med Chef. Her kan brukeren skrive og teste konfigurasjoner samt samhandle med Chef via kommandolinje-grensesnittet. Dette kan f.eks være en maskin, VM eller Git.
- Chef Client nodes: De maskinene som Chef skal håndtere må ha Chef Client installert av kommunikative grunner. Nodene kan kjøres som mer selvstendige agenter der de henter nye versjoner av konfigurasjonen selv.
- Chef Server: Dette er en hub for all konfigurasjonsdata. Her ligger all dataen som er skapt av Workstation og det er her Chef Client nodes henter informasjon om arbeid som skal bli utført, som f.eks templates og oppskrifter(recipes).

Komponenter
Chef har mange komponenter som kan påvirke effektiviteten til programmet. Dette er de mest sentrale: [28]

- Chef Development Kit: Dette er en pakke som inneholder alt som trengs for at Chef skal kunne benyttes. Denne pakken inneholder Chef DSL, chef-client, chef, knife(CLI-verktøy), test verktøy, Inspec og alt som trengs for å kunne skrive "cookbooks" og laste dem opp til Chef server.
- Cookbooks: En fundamental enhet for konfigurasjon og policy distribuering. Den definerer et scenario og alt som trengs for at scenarioet skal gå feilfritt. Dette innholder som regel recipes, attributtverdier, fil distribuering, templates og utvidelser. Dette er definert i språket Ruby og med noen spesifikke DSL ressurser.

- Policy: Kartlegger forretning- og driftskrav, prosesser, arbeidsflyt til innstillinger og objekter lagret på Chef serveren. Det inneholder som regel server-rolle, server-miljø, cookbook versjon og en run-list som nodene skal kjøre.
- Ohai: Et verktøy som samler på konfigurasjonsdata etter cookbooks har blitt tatt i bruk. Den samler blant annet informasjon på nettverk, disk, CPU og minne til senere analyse.

Sikkerhet
Kommunikasjon med Chef-server krever en sertifiseringsnøkkel ettersom at den bare utfører spørringer fra autentiserte brukere. Den tar i bruk en RSA public key kryptering. Her kan man kjøre chef-validator eller knife for å autentisere seg mot Chef master og bruke Chef sitt API.[29]

11.2.2 Puppet

Generelt
Puppet er et open-core[22] konfigurasjonsverktøy, og ble lansert i 2005[30]. Verktøyet gir brukeren en automatisk metode for å inspisere, levere, drifte og sikre infrastrukturen, og levere alle brukerens applikasjoner raskere. Puppet er kjent for å være grunnmuren for de fleste provisjoneringsverktøy, hvor andre leverandører har brukt Puppet sine prinispper for å utvikle andre provisjoneringsverktøy[15].

Dokumentasjon
Nettsiden til Puppet tilbyr dokumentasjon til alle de eksisterende tjenestene de har ansvaret for[31]. Dokumentasjonen er godt detaljert med konfigurasjonseksempler, dette for å gjøre det enklere for nybegynnere å installere og bruke programvaren. I tillegg finnes det også informasjon beregnet til mer erfarne brukere.

Puppet tilbyr også en egen læringsside[32],hvor brukeren kan laste ned et Puppet Learning VM. siden er et effektivt verktøy til å introdusere og lære opp nye brukere i programvaren. Puppet tilbyr også en rekke nettstudier og kurs, samt ta flere offisielle sertifiseringer.

Utvikling
Puppet har kildekoden sin åpen i Github[48]. Utviklingen av programvaren er basert på samarbeidet mellom selskapet og brukermiljøet. Brukermiljøet kan lett opprette feilmeldinger og løsningforsalg for vurdering. Dette gjør det enkelt å se hvem som har vært med under utviklingen og hvilke funksjoner som har blitt implementert under de forskjellige versjonene.

Arkitektur
Arkitekturen til Puppet er sentralstyrt av en Puppet "Master". En Puppet

[22]https://en.wikipedia.org/wiki/Open-core_model

Master holder på alle konfigurasjonene, mens Puppet "Agent" nodene sender forespørseler til Master for konfigurasjonene den trenger. Agentene kjøres på nodene som bakgrunnsprosesser som periodevis kommuniserer med "Catalogs" til Master[43].

Komponenter

- Puppet-språket: Puppet har et eget konfigurasjonsspråk [44]. Dette språket gir restriksjoner for hva Puppet kan gjøre. Språket simplifiserer konfigurasjoner som ellers hadde krevd mye egenskrevet kode. Poenget med språket er å definere ressurser, klassifisere ressursgrupper og node-roller.
- Catalogs: Dokumentet som agenten henter fra Master og eksekverer. Den forklarer hvilken tilstand noden skal ha og hvilke avhengigheter den trenger. En catalog kan referere til et eller flere manifester som konfigurerer node-miljøet [45].
- Modules: All Puppet kode kan ligge i moduler [46]. Hver modul tar hånd om en oppgave i infrastrukturen, som f.eks installering og konfigurering av en programvare. Moduler er de grunnleggende byggeblokkene for Puppet og er til gjengjeld mulig å brukes på nytt og deles. Moduler kan lages selv eller hentes fra Puppet Forge, som er et offentlig oppbevaringsted for ferdiglagde moduler fra Puppet og brukermiljøet.

Sikkerhet
Puppet har SSL-sertifikater og kommuniserer over HTTPS [47]. Når nodene skal kobles opp, gjør agenten en sertifiserings forespørsel til master. Master produserer sertifikatet slik at kommunikasjonen er autentisert.

11.3 Drøfting av server-konfigurasjonsverktøy

På grunn av anbefalt løsning vil ikke server-konfigurasjonsverktøy være nødvendig. Ved bruk av tjenesten AWS Fargate vil behovet for å konfigurere servere forsvinne. Hvis bedriften ønsker å ta i bruk EC2-instanser vil Ansible være anbefalt da de allerede har kjennskap til verktøyet. I dette tilfellet vil et bytte av verktøy ikke vil være hensiktsmessig.

12 Container-orkestreringsverktøy

12.1 Definisjon

Containerorkestrering er automatisering av alle aspekter ved koordinering og styring av containere, og har fokus på styre livssyklusen til containere og deres dynamiske miljøer[12].

12.2 Beskrivelse av verktøy

Det ble besluttet å beskrive to aktuelle verktøy. Kubernetes etter ønske fra Escio, og Docker Swarm som en alternativ kandidat.

12.2.1 Kubernetes

Generelt
Kubernetes er et open-source container-orkestreringsverktøy som er laget for automatisering av applikasjons-utplassering, skalering og styring. Kubernetes er et portabelt verktøy som håndterer containere og mikro tjenester [17]. Verktøyet er laget for å legge til funksjonalitet for all applikasjonshåndtering i et klynge-miljø. De fleste skytjenesteleverandører tilbyr ofte en PaaS løsning av dette systemet grunnet dens kompleksitet. Systemet ble originalt skapt av Google i 2014 og håndteres nå av Cloud Native Computer Foundation[50].

Dokumentasjon
Kubernetes har en egen nettside for dokumentasjon[51]. Her ligger all informasjon for å sette opp egenkonfigurert Kubernetes eller for spesfikke PaaS løsninger gitt av skytjeneste-leverandører. Nettstedet inneholder dokumentasjon fra både de offisielle skaperne og brukermiljøet.

Opplæring i Kubernetes er en egen del av dokumentasjonen[51]. Her finner man steg-for-steg guide på hvordan sette opp en enkel Kubernetes løsning. De har også gitt ut en minimal versjon av Kubernetes kalt Minikube. Minikube lærer bort de fundamentale funksjonene som trengs når en skal sette opp en container-klynge.

Utvikling
Utviklingen av Kubernetes og dens komponenter blir utviklet med kildekoden åpen på Github[52], for det meste i språket Go. Her kan brukermiljøet delta for å hjelpe prosjektene med å nå sine milepæler. Det er et aktivt brukermiljø med mange problemer og løsninger rapportert ukentlig.

Arkitektur
Arkitekturen baserer seg på en mester og flere arbeidsnoder[53]. Mesteren er en API-server med ”etcd”, kontroller-manager og scheduler. Den kontrollerer og overvåker delegeringen av arbeid og arbeidets tidsramme i en sentralstyrt

arkitektur. Arbeidet delegeres til nodenes Kublet og kjøres i Poden.

Komponenter

- etcd: Et enkelt nøkkelbasert lagringsted mesteren bruker til å lagre konfigurasjoner og tjenesteoppdagelse[53]. Dette brukes i hovedsak for back-up av klyngedata og deling av klyngekonfigurasjon over flere plattformer.
- Docker: Kubernetes bruker Docker som container-håndterings verktøy[53]. Docker brukes til å laste ned images og starte de relevante containerne.
- Kublet: En agent som kjører på noden som tar imot konfigurasjoner fra mesteren sitt API[53]. Den snakker også med etcd for å sjekke eventuelle andre tjenester som skal kjøres.
- Pod: Dette er en elastisk applikasjonshåndterer som ligger på nodene[53]. Ofte så er dette en sammensetning av containerer som deler på maskinressursene.

Sikkerhet
Kubernetes kjører TLS bootstrap sertifisering for autentisering mellom arbeider nodene og master API'et[54]. Når verifiseringen er gjort blir nøkkeler delt og kommunikasjonen åpen mellom de to partene. Kubernetes støtter også SSH tilkoblinger om port 22 er åpen[54]. Dette er effektivt for manuell konfigurerering, men ikke anbefalt for nodene og master med mindre brukeren har mye erfaring.

12.2.2 Docker Swarm

Generelt
Docker er et program som virtualiserer operativsystem-nivået til en maskin i form av containere[56]. Dette gjør at man kan kjøre flere virtuelle maskiner med delt kjerne.
Docker Swarm er Docker sin løsning på å håndtere delegering av container-arbeid over flere maskiner[18]. Docker Swarm fokuserer på skalering og last balansering i et desentralisert miljø.

Dokumentasjon
Docker Swarm har dokumentasjonen sin sammen med Docker dokumentasjonen [57]. Den spesifiserer initielle krav før oppsetting av verktøyet, samt de resterende stegende for å starte en fullstendig applikasjon. Docker tilbyr også gode kommunikasjonsplattformer for å samhandle med brukermiljøet i form av blog, møter og forum[58].

Utvikling
Docker Swarm er et open-source program under utvikling i Github[59]. Det er

en oversiktlig og enkelt å navigere seg frem til versjoner, endringer, milepæler og brukermiljøets løsninger. Brukere kan lett bidra med problemer og løsninger til evaluering av neste utgivelse.

Arkitektur
Arkitekturen er basert på noder som er en instanse av Docker Engine som kjører [18]. Dette kan enten være en Manager node eller Worker node. Manager utsender arbeidsenheter kalt tasks til Worker nodene. Det er Manager som har ansvaret for å opprettholde korrekt tilstand og orkestrere hele klyngen. Worker noden mottar tasken og kjører det på sin Docker Engine [18]. Ettersom at Manager har Docker Engine, kan den selv utføre oppgaver om nødvendig.

Komponenter

- Swarmkit: Et toolkit for orkestreringdistribueringene. Den automatiserer last-balansering og skalering innad i Swarmen. Sikkerheten øker ettersom den gir automatisert autentisering, autorisering og kryptering. [55].
- Docker Engine: Programvare som må kjøre for at en oppgave kan bli utført. [18].
- Service: Arbeidet som skal bli utført av Worker nodene. Det inneholder images og kommandoer som skal kjøre på innsiden av containeren[18].

Sikkerhet
Sikkerheten mellom Worker nodene og Manager nodene baserer seg på en mutual TLS for å autentisere, automatisere og kryptere kommunikasjonen. Validering av sertifikatene styres av Manager når nodene tilkobles for første gang[60].

12.3 Drøfting av Container-orkestreringsverktøy

Ut i fra disse to verktøyene er Docker Swarm å anbefale, da dette er enklere å lære og ta i bruk. Kubernetes er et kraftigere verktøy, men er mer komplekst og tidkrevende å sette opp. Bedriften ønsker å fokusere på applikasjonsutvikling for sine kunder og begrense tidsbruk relatert til driftsoppgaver. Derfor anbefales det å bruke en tjenesten ECS. ECS er enkel å bruke og er godt støttet av tjenesten Fargate. Ved å ta i bruk disse kan bedriften se bort ifra å installere og vedlikeholde verktøyet Docker Swarm.

13 Konklusjon

Gjennom denne rapporten er det beskrevet og drøftet en rekke skytjenestemodeller med relaterte tjenester i AWS, samt noen utvalgte verktøy med forskjellige bruksområder, som kan være ledd i å bygge opp en CI/CD-pipeline. Den har også gitt ett kort innblikk i fagfeltet infrastruktur som kode. Rapporten har vektlagt blant annet kompatibilitet mot AWS, god dokumentasjon og et aktivt brukermiljø. I tillegg vil enklest mulig operasjon for driftspersonell kunne frigjøre tid for bedriftens ansatte. Denne tiden kan benyttes til verdiskapning i tråd med selskapets fokusområder. Løsningen gruppen har foreslått legger også vekt på at majoriteten av applikasjonene til Escio driftes i AWS. Derfor anbefales følgende komponenter illustrert under.

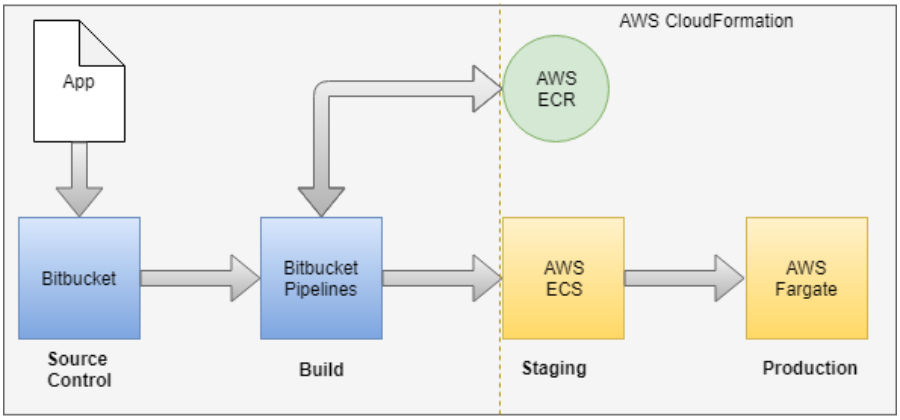


Figure 4: endret versjon

Som vist på figuren er majoriteten av tjenestene i pipeline representert av AWS, dette er fordi AWS tilbyr et bredt utvalg av tjenester som kan brukes til å bygge opp en CI/CD-pipeline. Alle tjenester benyttet er kompatible med hverandre og oppfyller bedriften sine ønsker og behov. Ved å benytte seg av denne løsningen vil bedriften spare tid ved å ikke måtte installere og vedlikeholde verktøy som Ansible og Docker Swarm. Løsningen er basert på at Escio utvikler container-baserte applikasjoner, derfor er tjenesten ECS benyttet med Fargate. Fargat har en serverløs tilnærming og vil ta vekk ansvaret med å provisjonere servere. Hvis bedriften ønsker å kjøre applikasjoner i virtuelle maskiner kan dette gjøres ved å opprette og provisjonere EC2 instanser med verktøyet Ansible.

References

[1] Amazon(2019). *Amazon Web Services*. Tilgjengelig fra:
https://aws.amazon.com
(Hentet: 14.01.2019)

[2] Kief Morris(2016). *Infrastructure as Code: Managing Servers in the Cloud*. California, USA: O'Reilly Media.

[3] Sophia Bernazzani(2018). *IaaS vs. PaaS vs. SaaS: Here's What You Need to Know About Each*. Tilgjengelig fra:
https://blog.hubspot.com/service/iaas-paas-saas
(Hentet: 03.02.2019)

[4] Wikipedia(2019). *Platform as a Service*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Platform_as_a_service
(Hentet: 04.02.2019)

[5] Wikipedia(2018). *Funtion as a Service*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Function_as_a_service
(Hentet: 04.02.2019)

[6] Wikipeida(2019). *Software as a Service*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Software_as_a_service
(Hentet: 05.02.2019)

[7] SearchItoperations(2019). *Container as a Service (CaaS)*. Tilgjengelig fra:
https://searchitoperations.techtarget.com/definition/Containers-as-a-Service-CaaS
(Hentet: 13.02.2019)

[8] Stephen Watts(2017). *Multi-Cloud Blog SaaS vs PaaS vs IaaS: What's The Difference and How To Choose*. Tilgjengelig fra:
https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/
(Hentet: 08.02.2019)

[9] Nilesh Suryavanshi(2017). *What are Cloud Computing Services [IaaS, CaaS, PaaS, FaaS, SaaS]*. Tilgjengelig fra:
https://medium.com/@nnilesh7756/what-are-cloud-computing-services-iaas-caas-paas-faas-saas-ac0f6022d36e
(Hentet: 11.02.2019)

[10] Cynthia Harvey(2018). *IaaS vs PaaS vs SaaS: Which Should You Choose?*. Tilgjengelig fra:
https://www.datamation.com/cloud-computing/iaas-vs-paas-vs-saas-which-should-you-choose.html
(Hentet: 11.02.2019)

[11] Atlassian(2019). *What is version control*. Tilgjengelig fra:
https://www.atlassian.com/git/tutorials/what-is-version-control
(Hentet: 12.02.2019)

[12] AVI Networks(2019). *Container Orchestration Definition*. Tilgjengelig fra:
https://avinetworks.com/glossary/container-orchestration/
(Hentet: 13.02.2019)

[13] International Organization for Standardization(2019). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Tilgjengelig fra:
https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en
(Hentet: 19.02.2019)

[14] Wikipedia(2019). *Chef (software)*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Chef_%28software%29
(Hentet: 22.02.2019)

[15] Puppet Enterprise(2019). *How Puppet works*. Tilgjengelig fra:
https://puppet.com/products/how-puppet-works
(Hentet: 23.02.2019)

[16] Patrick Ogenstad(2019). *What is Ansible?*. Tilgjengelig fra:
https://networklore.com/ansible/
(Hentet: 23.02.2019)

[17] Kubernetes(2019). *What is Kubernetes?*. Tilgjengelig fra:
https:
//kubernetes.io/docs/concepts/overview/what-is-kubernetes/
(Hentet: 23.02.2019)

[18] Docker(2019). *Swarm mode key concepts*. Tilgjengelig fra:
https://docs.docker.com/engine/swarm/key-concepts/
(Hentet: 23.02.2019)

[19] Wikipedia(2019). *Vendor lock-in*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Vendor_lock-in/
(Hentet: 24.02.2019)

[20] STORE NORSKE LEKSIKON(2019) *Automatisering*. Tilgjengelig fra:
https://snl.no/automatisering
(Hentet: 25.02.2019)

[21] Wikipedia(2019) *Orkestrering*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Orchestration_(computing)
(Hentet: 25.02.2019)

[22] Wikipedia(2019). *Server configuration management*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Software_configuration_management
(Hentet: 26.02.2019)

[23] Amazon(2019). *AWS OpsWorks for Chef Automate*. Tilgjengelig fra:
https://aws.amazon.com/opsworks/chefautomate/
(Hentet: 27.02.2019)

[24] Chef Software, Inc.(2019). *Learn Chef*. Tilgjengelig fra:
https://docs.chef.io
(Hentet: 27.02.2019)

[25] Chef Software, Inc.(2019). *Learn Chef Rally*. Tilgjengelig fra:
https://learn.chef.io/#/
(Hentet: 27.02.2019)

[26] Chef Software, Inc.(2019). *Chef Software, Inc.*. Tilgjengelig fra:
https://github.com/chef
(Hentet: 27.02.2019)

[27] Chef Software, Inc.(2019). *Projects*. Tilgjengelig fra:
https://github.com/chef/chef/projects
(Hentet: 27.02.2019)

[28] Chef Software, Inc.(2019). *An Overview of Chef*. Tilgjengelig fra:
https://docs-archive.chef.io/release/server_12-4/auth.html
(Hentet: 27.02.2019)

[29] Puppet Inc.(2019). *Puppet (software)*. Tilgjengelig fra:
https://docs.chef.io/chef_overview.html
(Hentet: 27.02.2019)

[30] Chef Software, Inc.(2019). *Authentication, Authorization*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Puppet_%28software%29
(Hentet: 27.02.2019)

[31] Puppet Inc.(2019). *Welcome to Puppet 6 documentation*. Tilgjengelig fra:
https://puppet.com/docs/puppet/6.3/puppet_index.html
(Hentet: 27.02.2019)

[32] Puppet Inc.(2019). *Welcome to Puppet Training!*. Tilgjengelig fra:
https://learn.puppet.com/
(Hentet: 27.02.2019)

[33] Amazon EC2.(2019). *What Is AWS EC2?*. Tilgjengelig fra:
https:
//docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
(Hentet: 27.02.2019)

[34] Amazon ECS.(2019). *What Is Elastic Container Sservice?*. Tilgjengelig fra:
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html
(Hentet: 27.02.2019)

[35] Amazon Fargate.(2019). *What Is AWS Fargate?*. Tilgjengelig fra:
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Fargate.html
(Hentet: 27.02.2019)

[36] Amazon EKS.(2019). *What Is AWS EKS?*. Tilgjengelig fra:
https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html
(Hentet: 27.02.2019)

[37] Amazon Elastic Beanstalk.(2019). *What Is AWS Elastic Beanstalk?*. Tilgjengelig fra:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html
(Hentet: 27.02.2019)

[38] Amazon Lambda.(2019). *What Is AWS Lambda?*. Tilgjengelig fra:
https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
(Hentet: 27.02.2019)

[39] Amazon CloudFormation.(2019). *What Is AWS CloudFormation?*. Tilgjengelig fra:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html
(Hentet: 27.02.2019)

[40] Amazon S3.(2019). *AWS S3*. Tilgjengelig fra:
https://aws.amazon.com/s3/?nc2=h_m1
(Hentet: 27.02.2019)

[41] Amazon CodeBuild.(2019). *What Is AWS CodeBuild?*. Tilgjengelig fra:
https://docs.aws.amazon.com/codebuild/latest/userguide/welcome.html
(Hentet: 27.02.2019)

[42] Amazon CodeDeploy.(2019). *What Is AWS CodeDeploy?*. Tilgjengelig fra:
https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html
(Hentet: 27.02.2019)

[43] Puppet Inc.(2019). *Overview of Puppet's architecture*. Tilgjengelig fra:
https://learn.puppet.com/
(Hentet: 27.02.2019)

[44] Puppet Inc.(2019). *Language: Basics*. Tilgjengelig fra:
https://puppet.com/docs/puppet/6.3/lang_summary.html
(Hentet: 28.02.2019)

[45] Puppet Inc.(2019). *Static catalogs*. Tilgjengelig fra:
https://puppet.com/docs/puppet/6.3/static_catalogs.html
(Hentet: 28.02.2019)

[46] Puppet Inc.(2019). *Module fundamentals*. Tilgjengelig fra:
https://puppet.com/docs/puppet/6.3/modules_fundamentals.html#concept-1234
(Hentet: 28.02.2019)

[47] Puppet Inc.(2019). *Certificate authority and SSL*. Tilgjengelig fra:
https://puppet.com/docs/puppet/6.3/ssl_certificates.html
(Hentet: 28.02.2019)

[48] Puppet Inc.(2019). *Puppet*. Tilgjengelig fra:
https://github.com/puppetlabs
(Hentet: 28.02.2019)

[49] Amazon (2019)*Practicing Continuous Integration and Continous Delivery on AWS*. Tilgjengelig fra:
https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf
(Hentet: 28.02.2019)

[50] Wikipedia(2019). *Kubernetes*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Kubernetes
(Hentet: 28.02.2019)

[51] Kubernetes(2019). *Kubernetes Documentation*. Tilgjengelig fra:
https://kubernetes.io/docs/home/
(Hentet: 28.02.2019)

[52] Kubernetes(2019). *Kubernetes*. Tilgjengelig fra:
https://github.com/kubernetes
(Hentet: 28.02.2019)

[53] Nune Isabekyan(2016). *INTRODUCTION TO KUBERNETES ARCHITECTURE*. Tilgjengelig fra:
https://x-team.com/blog/introduction-kubernetes-architecture/
(Hentet: 28.02.2019)

[54] Kubernetes(2019). *Master-Node communication*. Tilgjengelig fra:
https://kubernetes.io/docs/concepts/architecture/master-node-communication/
(Hentet: 28.02.2019)

[55] Docker(2019). *docker/swarmkit*. Tilgjengelig fra:
https://github.com/docker/swarmkit/
(Hentet: 28.02.2019)

[56] Docker(2019). *Docker (software)*. Tilgjengelig fra:
https://en.wikipedia.org/wiki/Docker_(software)
(Hentet: 28.02.2019)

[57] Docker(2019). *Getting started with swarm mode*. Tilgjengelig fra:
https://docs.docker.com/engine/swarm/swarm-tutorial/
(Hentet: 28.02.2019)

[58] Docker(2019). *Docker Community*. Tilgjengelig fra:
https://www.docker.com/docker-community
(Hentet: 28.02.2019)

[59] Docker(2019). *docker/swarm*. Tilgjengelig fra:
https://github.com/docker/swarm
(Hentet: 28.02.2019)

[60] Docker(2019). *Manage swarm security with public key infrastructure (PKI)*. Tilgjengelig fra:
https://github.com/docker/swarm
(Hentet: 01.03.2019)

Vedlegg

A Google trends

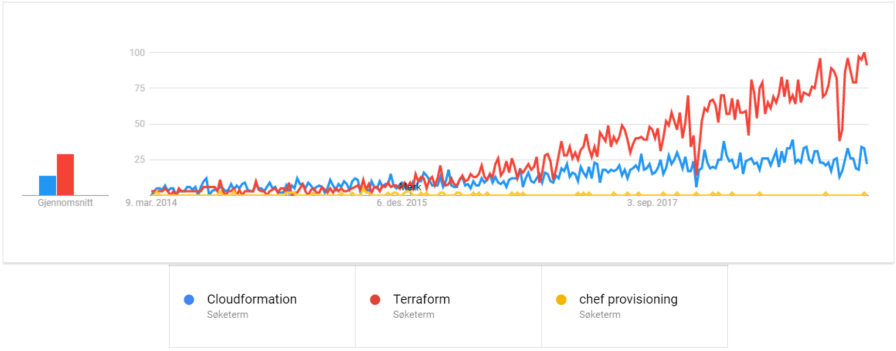


Figure 5: Trend for infrastruktur-konfigurasjonsverktøy siste 5 år

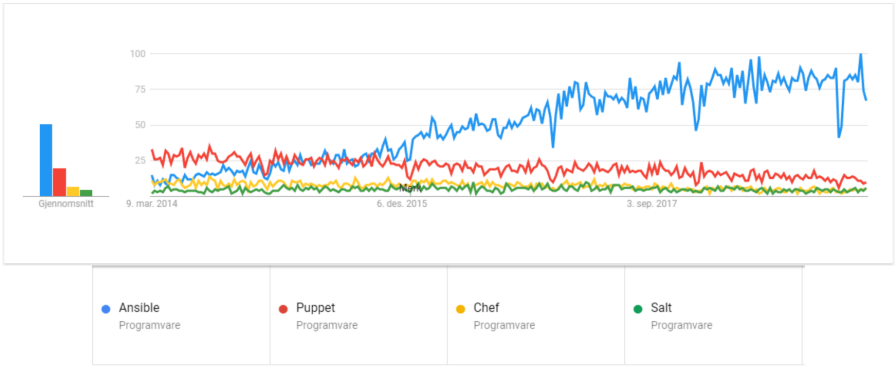


Figure 6: Trend for Server-konfigurasjonsverktøy siste 5 år

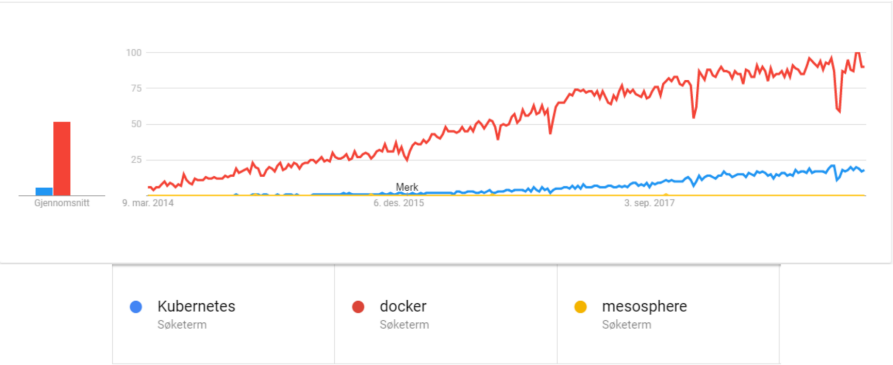


Figure 7: Trend for Container-orkestreringsverktøy siste 5 år

Tall hentet fra:

- https://trends.google.com/trends/explore?cat=32&date=today%205-y&q=Cloudformation,Terraform,chef%20provisioning
- https://trends.google.com/trends/explore?cat=32&date=today%205-y&q=%2Fm%2F0k0vzjb,%2Fm%2F03d3cjz,%2Fm%2F05zxlz3,%2Fm%2F0hn8c6s
- https://trends.google.com/trends/explore?cat=32&date=today%205-y&q=Kubernetes,docker,mesosphere

B Stack overflow tags

Verktøy	**Stack overflow tags**
Infrastruktur-konfigurasjonsverktøy	
CloudFormation	3071
Terraform	2420
Chef provisoning	9
Server-konfigurasjonsverktøy	
Ansible	9283
Puppet	3554
Chef	5936
Salt	1047
Container-orkestreringsverktøy	
Docker	53125
Kubernetes	14811
Mesosphere	1201

Tall hentet fra: https://stackoverflow.com/tags