Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for informasjonssikkerhet og kommunikasjonsteknologi

Birger Myhrvold
Daniel Bruun
Jostein Vole Hage

# Containerization & CI/CD Pipeline

Bacheloroppgave i IT-drift & Informasjonssikkerhet
Veileder: Erik Hjelmås
Mai 2019

NTNU
Kunnskap for en bedre verden

NTNU
Norwegian University of
Science and Technology

# Containerization og CI/CD pipeline

Forfattere
Birger Myhrvold
Daniel Bruun
Jostein Vole Hage

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

20.5.2019

Veileder Erik Hjelmås

# Sammendrag av Bacheloroppgaven

| | |
|---|---|
| Tittel: | **Containerization og CI/CD pipeline** |
| Dato: | 20.5.2019 |
| Deltakere: | Birger Myhrvold<br>Daniel Bruun<br>Jostein Vole Hage |
| Veiledere: | Erik Hjelmås |
| Oppdragsgiver: | Ellera AS |
| Kontaktperson: | Jørgen Ellingsen, jorgen@ellera.no |
| Nøkkelord: | Containere, CI/CD, pipeline, CMS |
| Antall sider: | 64 |
| Antall vedlegg: | 12 |
| Tilgjengelighet: | Åpen |

Sammendrag:

Informasjonsteknologi er stadig i utvikling. Større etterspørsler og strengere krav åpner stadig for nye veier og muligheter. Det utvikles nye løsninger for å imøtekomme kravene og dette fører til nye metoder for å oppnå bedre resultater. Programmerbar infrastruktur gjør det mulig å behandle og provisjonere servere gjennom konfigurasjonsfiler, og fjerner nødvendigheten for manuelt arbeid mot både logisk og fysisk utstyr. Hva skal til for å kunne løse problemer basert på tilgjengelighet, oppetid av tjenester, automatisk flyt og ta infrastrukturer til nye høyder? Løsningen er å skape en automatisert prosess fra kode til produksjonsklar infrastruktur. Her har de ansatte selv mulighet for å bestemme innholdet og kan lett skalere tjenesten etter ønsket behov. Gjennom prosjektet har vi levert et produkt som tar i bruk metoder for logging, CI/CD, orkestrering og provisjonering av servere, som igjen kan settes ut i et produksjonsmiljø etter mindre justeringer.

# Summary of Graduate Project

| | |
|---|---|
| Title: | **Containerization and CI/CD pipeline** |
| Date: | 20.5.2019 |
| Authors: | Birger Myhrvold<br>Daniel Bruun<br>Jostein Vole Hage |
| Supervisor: | Erik Hjelmås |
| Employer: | Ellera AS |
| Contact Person: | Jørgen Ellingsen, jorgen@ellera.no |
| Keywords: | Containers, CI/CD, pipeline, CMS |
| Pages: | 64 |
| Attachments: | 12 |
| Availability: | Open |

Abstract: Informationtechnology is in continuous growth. Greater demand and stricter requirements unveils new ways and possibilities. New solutions are developed to please the community and this leads to new methods that brings better results. Infrastructure as Code aims to make a way for provisioning and management through configuration files, and removes manual labor from the process of setting up both virtual and physical servers. How can we solve the problems based on availability, up time of services, automatic flow of code and bring infrastructure to new heights? The solution is to create an automated process from writing code to deploying a service. Through code the employees can manage, customize and deploy services based on the demand. By the completion of this project we have developed a product that utilizes methods for logging, CI/CD, orchestration and provisioning of servers, that can be deployed into production just after some minor adjustments.

# Preface

Denne bacheloroppgaven er skrevet ved Institutt for informasjonssikkerhet og kommunikasjonsteknologi ved NTNU i Gjøvik av Jostein Vole Hage, Daniel Bruun & Birger Myhrvold. Vi ønsker å takke vår veileder Erik Hjelmås for samarbeidet og veiledningen gjennom prosjektperioden. Vi ønsker også å takke Ellera og vår kontaktperson Jørgen Ellingsen for å gi oss denne muligheten. Videre ønsker vi å takke alle studenter og ansatte ved NTNU som har bistått oss underveis. Til slutt vil vi takke hverandre for samarbeidet underveis, hvor vi har hjulpet hverandre i motgang og medgang, og all lærdommen som har kommet ut av det hele.

**Birger Myhrvold - 997829**
**Jostein Vole Hage - 999551**
**Daniel Bruun - 473117**

# Innhold

# Figurer

# Tabeller

# Listings

# Akronymer

**API** Application programming interface. 11, 14, 19, 29

**CI/CD** Continuous Integration/Continuous Deployment. 1–3, 12, 15, 27–29, 56

**CMS** Configuration Management System. 1, 12, 14, 18, 22, 56

**DSML** Domain-specific modeling language. 11

**SSH** Secure shell. 17, 22, 23, 38, 44

**SSL** Secure Sockets Layer. 17

**VCS** Version Control System. 9, 10, 15, 16

**VM** Virtual machine. 24, 32

# Ordliste

**ad-hoc** å gjøre reaktive endringer etterhvert som behov oppstår. xiv, 1

**best practice** retningslinjer for gjennomføring av en aktivitet, som er akseptert som en god og/eller effektiv måte å gjøre det på. 1, 2, 38–40, 48, 55

**cluster** er et sett med noder som jobber sammen i en klynge og fungerer som et system. xv, 2, 17, 24, 29, 32, 42

**configuration drift** oppstår når konfigurasjon for komponenter i en infrastruktur blir annerledes på grunn av ad-hoc endringer for problemer eller vedlikehold. 1, 14

**container** er en standarisert metode for innpakkning av en applikasjon, i form av kode,konfigurasjon og avhengigheter. Alt dette gjøres om til et enkelt objekt. Containere deler det underliggende operativsystemet og kjører som isolerte prosesser, som fører til rask, pålitelig og konsistent deployering uavhengig av miljø. xiv, 13, 47, 49

**containere** se container. 3, 6, 15, 17, 24, 30–32, 45, 56

**DevOps** er et fleksibelt forhold mellom utvikling og IT-drift. Målet med DevOps er å endre og forbedre forholdet ved å foreslå bedre kommunikasjon og samarbeid mellom disse to forretningsenhetene[1]. 12

**heat-engine** er ansvarlig for orkestrering og oppsett av definerte ressurser i templates[2]. 16

**idempotent** er å kunne utføre en og samme operasjon flere ganger og forvente samme resultat uavhengig av utgangspunkt. 10, 14, 39

**image** er et komprimert, selvstendig stykke programvare som kjører i en container. xv, 2, 8, 15, 18, 24, 27, 29–31, 45, 46, 49

**infrastrukturobjekter** se infrastrukturressurser. 37

**infrastrukturressurser** typiske komponenter i en IT infrastruktur, (virtuelle) servere, nettverk, lagring. xiv, 14, 21

**kernel** et program som er kjernen i et operativsystem, og har kontroll over alt i systemet. 17

**off-site** For IT-systemer vil off-site bety utenfor en sone eller et system. 30

**plugin** er en tilleggsmodul utviklet for å tilby ekstra funksjonalitet til en programvare. 20, 26

**provisjonere** er i denne rapporten brukt for å gjøre elementer tilgjengelig i form av, for eksempel servere eller konfigurasjonsfiler. 22

**repository** er en datastruktur brukt av versjonskontrollsystem, f.eks Git, til å lagre metadata for et sett filer og/eller kataloger[3]. 4, 8, 12, 13, 15, 22, 27, 30

**scrum** er et iterativt og inkrementelt smidig rammeverk for å utvikle komplekse informasjonssystemer [4]. 3

**service** er en gruppe containere med samme Docker image. 24

**skyløsninger** er en tjeneste som tilbyr hardware ressurser som lagring,applikasjoner og nettverk som en tjeneste. En skyløsning er et produkt av virtualisering. 14, 16

**sprint** er en konkret fase i utvikligen i et smidig utviklingsprosjekt, ofte med en varighet mellom én og tre uker [5]. 4

**stack** er en samling tjenester som sammen utgjør en applikasjon i et bestemt miljø. 29

**swarm** er en gruppe maskiner som kjører docker og er sammenkoblet til et cluster. 17, 23, 24, 29, 43, 49, 54

**templates** er en fil fungerer som et utgangspunkt eller mal for opprettelse av f.eks nye servere eller tjenester. xiv, 1, 16

**Unified Modeling Language** er en industristandard for datarelatert modellering[6]. xv

**use case** er en liste med hendelser i et handlingsforløp, vanligvis brukt for å definere interaksjonen mellom en aktør og et system. *Use cases* blir ofte skissert i et Unified Modeling Language-diagram[7]. 6

**virtualisering** å skape noe som funksjonelt fungerer på lik måte som et fysisk operativsystem, lagringssystem eller lignende, men som ikke krever de samme fysiske komponentene. xv, 14

**virtuell maskin** en Virtuell Maskin også referert til som VM er betegnelsen på en emulert versjon av et operativsystem. Som regel kjører denne på datamaskinarkitekturen til reelle datamaskiner. xv

**virtuelle maskiner** se virtuell maskin. 17, 32, 34

# 1 Introduksjon

## 1.1 Bakgrunn

Det finnes utallige metoder for deployering av IT-infrastrukturer, der det stadig utvikles nye verktøy og teknologier for at dette skal gå så smidig som overhodet mulig. Teknologier gjør det mulig å sette opp skreddersydde infrastrukturer etter behov, der disse kan skaleres og tilpasses. Ettersom det utvikles bedre metoder og nye best practice peker pilen stadig mer mot automatisering og dynamikk, og bort fra manuell konfigurering. Når de største aktørene på markedet garanterer en månedlig oppetid på 99.99%, fører dette til at de mindre aktørene også vil oppnå de samme resultatene [8]. Forventningene er skyhøye og presset for å levere er enormt. Vanligvis vil tjenester gå ned som en følge av oppdateringer eller vedlikehold til brukerens store fortvilelse. Automatisering fører til mer robuste infrastrukturer, der servere kan erstattes uten å måtte tenke på hvordan det påvirker systemet eller tjenesten, og oppdateringer kan rulles ut uten bekymringer for nedetid.

Når Ellera skal sette opp nye tjenester og servere i dag gjøres dette ved manuell konfigurering og valg av servere. Installering av tjenester gjøres ved bruk av skript og shell-kommandoer. Ulempen med dette er at nye versjoner og endringer i programvare kan gjøre skriptene ubrukelige, samtidig som at det ikke finnes noe kontroll av versjoner. På grunn av dette er det sannsynlig at servere blir unike, og får andre avhengigheter, som kan føre til påfølgende problemer.

Basert på disse problemene er Ellera ute etter å få utviklet en infrastruktur som innebærer bruk av prinsippene CI/CD pipeline, med fokus på automatisering og programmerbar infrastruktur.

## 1.2 Problemområde

Manuelt oppsett av infrastrukturer, ing av servere og utrulling av tjenester medfører ofte problemer. Servere som er planlagt å gjøre de samme oppgavene blir ulike fra første stund. configuration drift oppstår ved endringer som gjør at enhver server blir unik. Løsninger på problemer eller retting av feil gjøres ad-hoc, og det benyttes ingen verktøy eller rammeverk. Servere utfører de samme oppgavene, men konfigurasjonen er ikke lenger lik, og ulike avhengigheter oppstår.

Disse problemene kan løses ved bruk av CI/CD pipelines og programmerbar infrastruktur. Servere provisjoneres ved bruk av templates slik at alle har samme utgangspunkt. Konfigurasjonssystem CMS) sørger for at konfigurasjoner holdes like, og overstyrer ad-hoc endringer slik at avhengighetene er de samme. Endringer og oppdateringer gjøres via en felles pipeline, en strukturert prosess som må gjennomføres for å kunne utføre endringer.

## 1.3 Oppgavebeskrivelse

På vegne av Ellera skal gruppen skape en infrastruktur som støtter driften selskapet har i dag på en mer effektiv, oversiktlig og organisert måte. Under prosessen skal det legges vekt på ulike teknologier, og gjøres en begrunnelse for valg av disse, der aktuelle kandidater vil kunne benyttes i utviklingen av en løsning.

Infrastrukturen skal inneholde:

- Metode for logging og monitorering
- Teknologi for orkestrering av containere
- CI/CD pipeline
- Provisjonering av servere

### 1.3.1 Logging og Monitorering

Det skal settes opp et system for logging og monitorering, slik at logger og systeminformasjon hentes ut fra nodene i infrastrukturen. Loggene skal aggregeres til et felles system, hvor systemet skal lagre loggene, og samtidig gi mulighet for søk og analysering.

### 1.3.2 Orkestrering av Containere

Løsningen for orkestrering skal gi mulighet for administrering av cluster med noder, der skalering skal kunne utføres gjennom denne. Fordeling av trafikk skal kunne gjøres ved bruk av last balansering internt i clusteret.

### 1.3.3 CI/CD Pipeline

Bruk av CI/CD pipeline skal gi mulighet for testing og levering av nye versjoner for Docker image. De nye imagene skal gjøres tilgjengelig via Docker Hub eller lignende tjenester. Ved nye versjoner skal disse kunne settes direkte ut i produksjon.

### 1.3.4 Provisjonering av servere

Oppsett av nye servere skal kunne gjøres automatisk. Ønsket operativsystem og programvare skal installeres ved utførelsen av et verktøy. Verktøyet skal forenkle prosessen for oppsett av nye servere og redusere tiden for dette.

## 1.4 Formål

Resultatet av prosjektet har som formål å utvikle en infrastruktur som kan komplementere eller erstatte den Ellera har i dag. Valgene som er gjort og konseptene som er tatt i bruk har som hensikt å illustrere mulighetene som eksisterer i dag og bruksnytten av best practice rettet mot dynamiske infrastrukturer. Valgene og vurderingene av teknologiene som benyttes har som hensikt å begrunne nødvendigheten for bruk av verktøy og hvorfor disse er essensielle for å oppnå automatisering.

Prosjektgruppen valgte denne oppgaven fordi alle medlemmene hadde interesse for drifting av IT-systemer, og hadde erfaringer med dette fra tidligere emner. Det var også et ønske om å utvide forståelsen og kunnskapen tilknyttet emner som programmerbar infrastruktur og dynamiske skyplattformer.

## 1.5 Avgrensning

Det vil bli utviklet en løsning for en infrastruktur med fokus på CI/CD-pipeline, logging, orkestrering, containere og automatisering, samt hvordan dette kan løses i praksis. Med tanke på teknologi står gruppen fritt til å beslutte valgene selv. Vurderingsfasen av aktuelle teknologier skal danne et generelt grunnlag som gir et godt nok inntrykk til å bestemme hvilke av kandidatene som skal benyttes i en mulig løsning.

- For operativsystem vil det kun benyttes Linux-distribusjoner
- Gruppen har ikke foretatt en vurdering av best egnet skyleverandør
- Gruppen velger å ikke foreta en vurdering av alternativer til eksisterende komponeneter i infrastrukturen
- Det skal ikke foretas en sikkerhetsvurdering av CI/CD-pipeline

## 1.6 Målgruppe

Målgruppen for oppgaven er hovedsakelig oppdragsgiver, men også personer som har erfaring eller interesse for IT-drift og tjenester, spesielt rettet mot skyplattformer og programmerbar infrastruktur. For å kunne ta fullt utbytte av rapporten vil det være en fordel med relevant erfaring eller kunnskap innenfor informasjonsteknologi, spesielt rettet mot drift. Rapporten vil være meget relevant for de som jobber daglig med server administrasjon, og de som er ute etter å skape en mer strukturert og solid infrastruktur.

## 1.7 Egen bakgrunn og kompetanse

Samtlige på gruppen hadde generell kompetanse innenfor programmerbar infrastruktur og skyplattformer fra tidligere emner, og samme bakgrunn fra IT-drift og informasjonssikkerhet. Gjennom emner som systemutvikling har gruppen opparbeidet seg kunnskap om systemutviklingsmodeller og hvordan jobbe i større prosjekter med tanke på utvikling.

## 1.8 Metode

Gruppen skal utvikle en løsning for infrastrukturen i løpet av en periode på omtrent fire måneder. Valget av utviklingsmodell ble utført etter drøfting av flere hovedpunkter, som var essensielle rundt prosjektoppgaven og som var relevant for en smidig utviklingsmodell.

- Estimering av varighet for oppgaver
- Smidige modeller åpner for endringer underveis
- Prosjektet har et lite utviklerteam

Til å begynne med hadde Ellera få krav til sluttproduktet, og ga fra begynnelsen av inntrykket av de var veldig fleksible og åpne for dialog underveis. En iterativ modell som scrum vil fungere utmerket for et slikt prosjekt, der nye arbeidsoppgaver vil oppstå underveis. Scrum vil benyttes som utviklingsmodell, utfordingen med dette er at ingen fra gruppen har tidligere erfaringer med bruk av Scrum, spesielt med tanke på omfanget av et slikt prosjekt. Scrum har mange verktøy for smidig utvikling, og en gitt struktur/prosess som deler utviklingsperioden inn i sprinter.

### 1.8.1 Bruk av Scrum i praksis

Gruppen valgte å ha sprinter med et intervall på to uker som tilsvarer 10 arbeidsdager. Prosjektperioden vil være sammensatt av 5 sprinter.

- Sprint 1: 23.1 - 13.2
- Sprint 2: 13.2 - 10.3
- Sprint 3: 10.3 - 31.3
- Sprint 4: 31.3 - 21.4
- Sprint 5: 21.4 - 20.5

På grunn av vanskeligheter rundt estimering av tidsbruk og omfanget av arbeidsoppgaver ble det nødvendig å utføre endringer på fremdriftsplanen underveis, noe som førte til at sprintene hadde en tendens til å bli lenger enn planlagt.

Vanlig praksis for bruk av Scrum inkluderer Daily Scrum Meeting, som er et møte på 10-15 minutter for å holde alle i utviklerteamet oppdatert på hva som ble utført av arbeidsoppgaver dagen før, og hva som er planen for gjeldende dag. Ved fullføring av en sprint holdes det et Retrospective møte for alle i utviklerteamet der det diskuteres hva som gikk bra, hva kan forbedres til neste sprint, og hva kan videreføres med inn i neste sprint.

### 1.8.2 Arbeidsflyt

Alt arbeid som utføres skal dokumenteres, slik at alle har tilgang på dokumentasjon som forteller hva som er blitt gjort i løpet av en sprint. For føring av timelister har hvert enkelt gruppemedlem selv ansvar for at dette utføres. Referater fra møter med veileder og oppdragsgiver lagres på Dropbox i felles mappe for gruppen.

### 1.8.3 Kildekode

Kildekode lagres i separate Bitbucket repository der endringer, ny kode og forbedringer av eksisterende kode tilføyes ved bruk av git. Commits skal ha en god beskrivelse slik at det er god oversikt på hva slags endringer som er utført ved nye commits. Koden i seg selv skal følge standarden for gjeldende kodespråk, utover dette skal det være god struktur med en tydelig sammenheng mellom kode og kommentering.

## 1.9 Om rapporten

I rapporten har det blitt valgt å bruke engelske betegnelser, siden det er disse som oftest er brukt i fagmiljøet, og er derfor lettere gjenkjent.
Testing er ikke et fokus i denne rapporten, og kildekoden er lagt ved innlevering av rapporten som en ZIP-fil

**Rapportstruktur**

Rapporten er inndelt i ti kapitler, som omhandler disse hovedemnene:

- Krav fra oppdragsgiver
- Grunnleggende teori om prinsipper innenfor infrastruktur
- Introduksjon til relevante teknologier
- Vurderinger og valg, gjengir begrunnelse og bruk av teknologiene

- Design og implementasjon, gjennomgang av praktisk bruk av verktøy og teknologier
- Oppsummering og anbefaling til videre arbeid

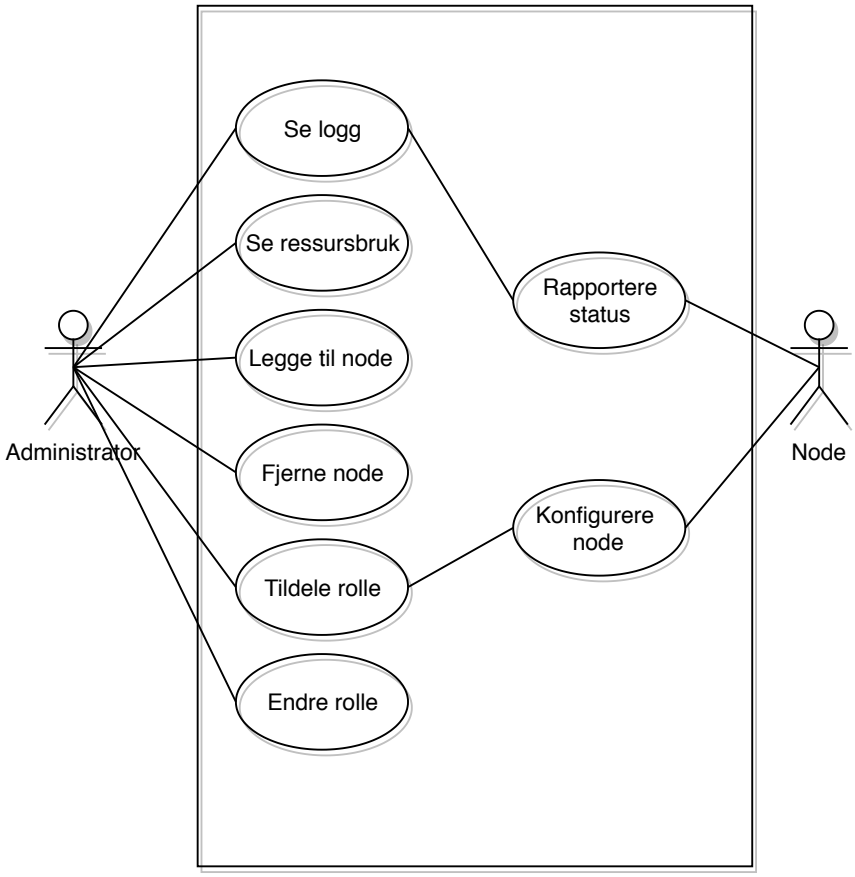
# 2 Kravspesifikasjon

Ved utarbeid av kravspesifiseringen hadde oppdragsgiver allerede en liste med ønsker for infrastrukturen. Det ble behov for å utarbeide og konkretisere en egen spesifikasjon til utvikling.

- Automatisk oppsett av infrastruktur, som klargjør systemet for utvikling.
- Deployment pipeline: Staging og production deployment med automatisk testing basert på commits/merges i Bitbucket.
- Docker containere: Container management & redundancy.
- Monitorering

## 2.1 Funksjonelle Krav

For å best mulig illustrere og forklare de ønskede handlingene som skal være tilgjengelig via systemet ble det utarbeidet use case og et use case-diagram som følge av krav fra oppdragsgiver.




Figur 1: Use case-diagram

### 2.1.1 Høynivå use case-beskrivelser

| | |
|---|---|
| Use Case | Se Logg |
| Aktør | Administrator |
| Hensikt | Vise logger til admin |
| Beskrivelse | Drifteren eller administratoren av infrastrukturen skal kunne se logg for alle nodene i infrastrukturen. Dette kan være feilmeldinger o.l. |

Tabell 1: Use Case Se Logg

| | |
|---|---|
| Use Case | Tildele Rolle |
| Aktør | Administrator |
| Hensikt | Gi noden en rolle |
| Beskrivelse | Ved tilføying av flere noder skal administratoren ha mulighet til å tildele denne en rolle slik at noden blir automatisk konfigurert. |

Tabell 2: Use Case Tildele Rolle

| | |
|---|---|
| Use Case | Endre Rolle |
| Aktør | Administrator/Utvikler |
| Hensikt | Modifisere eksisterende rolle |
| Beskrivelse | Det skal være mulig å endre eksisterende roller. Dette vil føre til automatisk oppdatering av konfigurasjonen for de aktuelle nodene. |

Tabell 3: Use Case Endre Rolle

| Use Case | Legge til node |
|---|---|
| Aktør | Administrator |
| Hensikt | Legge til ny node/server i infrastruktur |
| Beskrivelse | Administrator skal kunne legge til en ny node i en eksisterende infrastruktur. |

Tabell 4: Use Case Legge til node

| Use Case | Fjerne node |
|---|---|
| Aktør | Administrator |
| Hensikt | Fjerne node fra infrastruktur |
| Beskrivelse | Ved behov skal administrator kunne fjerne en node fra infrastrukturen. |

Tabell 5: Use Case Fjerne node

| Use Case | Oppdatering av Docker image |
|---|---|
| Aktør | Administrator |
| Hensikt | Publisere nye Docker images |
| Beskrivelse | Det skal være mulig å konfigurere Docker image og ved endringer skal disse sendes fra repository over til Docker Hub. Ved publisering på Docker Hub skal disse være tilgjengelig for bruk av container tjenesten. |

Tabell 6: Use Case Publisere Docker Image

| Use Case | Skalere antall containere |
|---|---|
| Aktør | Administrator |
| Hensikt | Øke/Redusere antallet tilgjengelige containere for en tjeneste |
| Beskrivelse | Gjennom orkestrering av container tjenesten skal administrator ha mulighet for å justere antallet med ønskede containere. Antallet containere skal ha mulighet for å justeres opp eller ned, der disse blir fordelt utover tilgjengelige noder. |

Tabell 7: Use Case Skalere tjenesten

## 2.2 Ikke-funksjonelle krav

- Levering av oppdateringer uten nedetid
- Det skal eksistere ekstern backup

## 2.3 Tekniske Krav

- Trafikk mellom servere skal være kryptert
- Caching ved bruk av Redis
- Oppsett av Craft CMS sammen med MySQL

## 2.4 Operasjonelle Krav

- Monitorering skal alltid vise informasjon om tilgjengelige noder
- Tilgang til systemet skal kun være tilgjengelig for administratorer
- Logger fra noder skal lagres eksternt
- Konfigurasjonsfiler skal lagres i et VCS

# 3 Teori

## 3.1 Programmerbar infrastruktur

Programmerbar infrastruktur er forvaltning og utvikling av infrastruktur på samme måte som programvare forvaltes og utvikles. Komponenter i infrastrukturen styres ved å lage konfigurasjonsfiler, som blir brukt av et konfigurasjonssytem til å realisere disse spesifiserte endringene. Prinsippet med programmerbar infrastruktur er å lage abstrakte definisjoner av ressurser. Verktøy muliggjør det å erstatte skript med konfigurasjonsfiler skrevet i et høynivå kodespråk for provisjonering av servere og tjenester. Ved å ta i bruk slike verktøy vil operasjonene i større grad være idempotent, og prosesser kan automatiseres med mindre feilmargin.

Siden programmerbar infrastruktur tar i bruk prinsipper fra programvareutvikling er det viktig å sikre god kvalitet av det som produseres. Dette åpner opp for at konfigurasjonsfiler og definisjoner kan være lagret i et felles versjonskontrollsystem (VCS). Hensikten med dette er å gjøre all konfigurasjon oversiktlig, slik at endringer som utføres er tilgjengelig. Programmerbar infrastruktur er populært og brukes av store selskaper som Amazon, Netflix og Google, og gir muligheter for høy pålitelighet og oppetid. Det vil også være fordelaktig for mindre selskaper som også vil være konkuransedyktige å ta i bruk dette prinsippet [9].

### 3.1.1 Automatisering

Automatisering er bruk av programvare for å lage repeterbare instruksjoner og prosesser som er ment for å erstatte manuelt arbeid, og redusere den menneskelige interaksjonen med IT-systemer. Automatisering har kommet langt forbi å være kun en trend, og blir sett på som en ren nødvendighet. Mye skyldes økt press i forhold til hvordan løsninger skal kunne skaleres og driftes uten nedetid.

Det er mulig for en IT-ansatt å konfigurere et fåtall servere manuelt, men så fort antallet stiger blir dette tidkrevende og lite lønnsomt, og derfor ikke lenger et alternativ. Automatisering gir mulighet for å kunne sette opp hundrevis av servere uten å måtte øke mengden ansatte. Det er her egne verktøy kommer inn i bildet [10].

## 3.2 Dynamisk infrastrukturplattform

En dynamisk infrastrukturplattform er et system som tilbyr ressurser i form av databehandling, som f.eks servere, lagringsenheter og nettverk på en måte som gjør det mulig å programmatisk allokere og administrere disse.

Siden programmerbar infrastruktur omhandler behandling av infrastrukturen i form av

kode, betyr dette at den dynamiske infrastrukturplattformen må oppfylle visse krav [9].

Plattformen må være:

- Programmerbar
- Etter behov
- Selvbetjent

**Programmerbar**

Plattformen må være programmerbar. Selv om et grafisk grensesnitt kan være praktisk, så er ikke det veien å gå for å oppnå programmerbar infrastruktur.

For å i større grad kunne automatisere må skript og andre verktøy kunne kommunisere med plattformen, og dette krever et API.

**Etter behov**

Det er essensielt for en programmerbar infrastruktur at den dynamiske infrastrukturplattformen har mulighet til å opprette og slette ressurser umiddelbart etter eget behov.

**Selvbetjent**

I tillegg til å kunne allokere ressurser etter egne behov, må brukere av infrastrukturen ha muligheter til å skreddersy og tilpasse disse ressursene til bedriftens behov.

## 3.3 Modelldreven utvikling

Modelldreven utvikling fokuserer på å abstrahere bort den logiske tankegangen ved programmering ved å deklarere en modell med ønskede spesifikasjoner over elementene og attributtene man trenger.

I modelldreven utvikling kombinerer: Domenespesifik modelleringspråk (*Domain-specific modeling language* Domain-specific modeling language (DSML)) og transformasjonsmoterer og generatorer (*Transformation engines and generators*). Når det i DSML spesifiseres ønskede elementer, vil generatoren sørge for at underliggende kode, data og andre nødvendige elemter blir generert.

Modelldreven utvikling benyttes av konfigurasjonssytem for konfigurasjon av komponenter i IT-infrastrukturer, ved at ressursene og elemente som er ønskelige deklareres i form av roller eller klasser (se seksjon 3.5) [11, 12].

## 3.4 Infrastrukturdefinisjonsverktøy

Infrastrukturdefinisjonsverktøy brukes til å administrere og spesifisere infrastrukturressurser, og hvordan disse skal konfigureres ved å følge prinsippene fra programmerbar infrastruktur. Et slikt verktøy bruker en dynamisk infrastrukturplattform til å implementere de gitte spesifikasjonene.

## 3.5 Konfigurasjonssystem

Konfigurasjonssystem (CMS) er en standarisert praksis og rammeverk for håndtering av ansvar, planlegging, identifisering, styring og endringer, overvåking og revisjon av en bedriftsprosess [13]. Konseptet baserer seg på å behandle alt som er nødvendig ved et system eller prosjekt. Dette kan være alt fra kildekode, konfigurasjonsfiler, servere og verktøy. Innenfor informasjonsteknologi vil CMS omhandle de gjenstandene som må konfigureres og behandles for at et system skal kunne driftes [14].

Prosessen ved CMS inneholder i hovedsak tre aktiviteter som må utføres:

1. **Konfigurasjonsidentifikasjon**
   I det første steget må hva som skal konfigureres identifiseres slik at disse kan behandles. Dette kan utføres av en manuell prosess, eller så kan det benyttes automatiske systemer eller tjenester. Hovedmålet er å skaffe en oversikt over hva som skal inngå i CMS.

2. **Konfigurasjonskontroll**
   Når alle elementene som skal behandles er identifisert, finnes det ingen garanti for at disse ikke endres. Ettersom at det er stor sannsynlighet for at disse blir utsatt for endring er det nødvendig å få på plass en mekanisme for å kontrollere endringene.

3. **Konfigurasjonsrevisjon**
   Selv om det er implementert mekanismer for å beskytte mot uforventede endringer, er det fortsatt behov for å kvalitetssikre ved bruk av testing og gjennomgang av kode.

## 3.6 CI/CD pipeline

CI/CD er en metode for å levere ofte til kunder ved å innføre automatisering inn i stadier av applikasjonsutviklingen. Dette er en kombinasjon av praksisene Continuous integration, Continuous delivery og Continuous deployment. CI/CD er en løsning på problemer som integrering av ny kode kan forårsake for utvlikling og drift av systemer.

CI/CD presenterer en prosess innenfor utvikling som innebærer automatisk testing og integrering av ny kode. CI/CD pipeline er syklusen som har endret måten programvare blir testet og utviklet. Dette er kjennetegnet og beinmargen i moderne DevOps.
CI/CD består av praksisene:

**Continuous Integration**

Continuous integration består av et sett av praksiser og en kode-filosofi, hvor utviklere går sammen for å integrere kode i et delt repository. Hver endring er verifisert gjennom automatisert testing av programvaren som hjelper med å oppdage problemer tidlig [15].

**Continuous delivery**

Continuous delivery er en naturlig utvidelse av continuous integration for å sørge for at man raskt og enkelt kan slippe nye endringer til kundene. Dette betyr at på toppen av å

ha automatisert testing, har man også en automatisert leveringsprosess som ligger klar til å levere ved hjelp av et tastetrykk [16].

#### Continuous deployment

Continuous deployment er continuous delivery tatt et steg lenger. Hvis en endring passerer alle stegene i pipelinen slippes den ut i produksjonsmiljøet uten menneskelig inngrep [17].

#### Pipeline

En pipeline er en sammensetning av en rekke prosedyrer som startes ved et nytt commit til et felles repository. Koden kompileres og tester utføres. Alle prosessene utføres automatisk, der koden enten slipper igjennom eller stoppes av en test. Ved feilfri gjennomgang av pipeline vil koden være klar for produksjonmiljøet.

## 3.7 Container Technology

Container technology, også omtalt som container er en metode for å pakke inn applikasjoner slik at de kan kjøre med sine avhengigheter i et eget lukket miljø. Uttrykket kommer fra shipping industrien, der containere er standardisert for å passe alle ulike transportmidler. For å slippe at alt må transporteres ved hjelp av spesielle metoder, blir helt like containere benyttet. Innenfor informasjonsteknologi blir uttrykket containere brukt for å referere til den samme standardiserte prosessen. For å slippe at ulike avhengigheter og ulike systemer fører til trøbbel for tjenesten, shippes applikasjonene i containere. Containere inneholder applikasjonen, samt alle avhengighetene, biblioteker, og konfigurasjonsfiler i ett. Dette fjerner problemene som kan oppstå på ulike maskiner, som for eksempel ved ulikt OS og maskinvare.

# 4 Vurdering for valg av komponenter

Dette kapittelet inneholder en oversikt over komponentene som skal være en del av infrastrukturen, samtidig som det utføres en vurdering av hvorfor disse bør implementeres.

## 4.1 Orkestrering

For at oppdragsgiver skal oppnå en programmerbar infrastruktur og kunne foreta endringer på infrastrukturen som en helhet, må verktøy med en slik evne benyttes. Infrastruktur definisjonsverktøy, også kalt orkestreringsverktøy gjør det mulig å spesifisere hvilke infrastrukturressurser som skal provisjoneres, endres eller fjernes [9]. De fleste orkestreringsverktøy forutsetter en dynamisk infrastrukturplattform med støtte for et API.

### 4.1.1 Hvorfor benytte orkestreringsverktøy?

For å holde orden og oversikt over infrastrukturen brukes orkestreringsverktøy. Slike verktøy er designet til å automatisere koordinering og administrering av servere og andre infrastrukturer på en idempotent måte. Ved å ta i bruk dette reduseres eller fjernes de manuelle prosessene for tildeling av nye servere. Fordelene med orkestrering er muligheten til å automatisere flere oppgaver og oppnå en bedre arbeidsflyt, slik at gjentagende prosesser går automatisk og at disse blir utført likt hver gang de utføres. Leveringen av en tjeneste kan gjøres raskere. Økt produktivitet og standardiserte prosesser er også fordeler for bruk av orkestrering.

## 4.2 Provisjonering

Et konfigurasjonssystem (CMS) benyttes for automatisering og vedlikehold av maskiner og programvare, for å holde de i en definert tilstand. CMSet lar brukeren skreddersy den ønskede tilstanden til systemet ved hjelp av et kjent kodespråk. Agenten til CMSet vil kontinuerlig jobbe for å opprettholde den definerte tilstanden, samtidig som gjeldende status vil rapporteres for hver gang agenten kjøres. Ettersom at tilstanden defineres i konfigurasjonsfiler, åpner dette for versjonskontroll og testing, samtidig som at den er lesbar for brukeren. CMS fører til konsistent konfigurasjon gjennom hele livssyklusen av utviklingen. Håndheving av ønsket tilstand for å forhindre configuration drift.

### 4.2.1 Hvorfor benytte et konfigurasjonssystem?

Hvorfor skal bedriften benytte seg av et CMS og hva er fordelene ved dette? skyløsninger og virtualisering gjør det mulig å opprette nye servere raskere enn noen gang. Dette kan føre til at server-miljøet vokser raskere enn evnen til å administrere og vedlikeholde disse etter formålet. Når dette problemet oppstår kan det bli vanskelig å opprettholde kvaliteten til serverne. Ved at enkelte servere blir utelatt. kan utdatert programvare åpne opp for sårbarheter mot systemet. Forskjeller i konfigurasjon mellom servere kan føre til at skript ikke lenger kjører som ønskelig. Dette problemet kalles for configuration drift. Selv om servere er satt opp ut ifra samme oppskrift, og om dette er ved bruk av skript eller manuelt arbeid, vil endringer skje gjennom serverens levetid. Ulikhetene kan oppstå som

følge av manuelle endringer eller at ulike metoder blir brukt for å installere de samme tjenestene på ulike maskiner. For at en infrastruktur skal fungere optimalt er det viktig å kunne erstatte en server uten å måtte bekymre seg over hvordan dette skal gjøres [9].

Implementeringen av et CMS vil gjøre at konfigurasjonen av systemene vil være mer oversiktlig da denne vil være definert i konfigurasjonsfiler. I tillegg vil disse være tilgjengelige for hele teamet ved bruk av VCS. CMS vil kunne kjøre over faste intervaller der konfigurasjonen vil verifiseres opp mot den ønskede spesifiseringen slik at den alltid er korrekt. Andre fordeler vil kunne være raskere gjenoppretting av en tjeneste ved nedetid, ettersom at konfigurasjonen er godt dokumentert og prosessen er automatisert. Skalering vil også kunne utføres mer dynamisk, samtidig som at arbeidet for å konfigurere flere servere ikke blir mer krevende enn ved konfigurasjon av bare én [18].

## 4.3 Logging og monitorering

For enhver infrastruktur bør det finnes en praktisk og oversiktlig håndtering av logger. Selv om en infrastruktur er liten, vil det spare tid og krefter å ha en form for logg-aggregering. Oppdragsgiver er klar over denne mangelen, og ønsker at dette blir implementert i løsningen. Med logger for en eller flere komponenter samlet på et sted, blir feilsøking og responstid for eventuelle problemer i infrastrukturen raskere for administrator. Ved å legge til et system som kategoriserer loggene ved hva slags type logg det er, og når det skjedde, vil gjøre navigasjon til relevant logg lettere.

## 4.4 Message-queue og caching

To komponenter i infrastrukturen vil spesielt ha svakheter i forhold til potensielt tap av data under drift. Dette er databasen og komponenten for logg-prosessering.

- Cache lagrer frekvent brukt data og reduserer tiden for å aksessere dette [19].
- Forvaltning av logger kan påvirkes ved stor pågang i systemet. Logger sendt for filtrering kan overstige kapasiteten til loggfiltreringsenheten, og blir droppet. Dette løses ved bruk av en message-queue [20, 21].

## 4.5 CI/CD

CI/CD funksjonalitet er et ønske fra oppdragsgiver, men det er også en stor fordel for bruk av containerization. Infrastrukturen vil bestå av flere containere som baseres på ulike image. Disse vil holdes oversiktlig i et Git-repository. For generell forvaltning av image, vil alle endringer testes og oppdateres i infrastrukturen under et kontrollsystem. Fordelene for å ta i bruk CI/CD vil altså være en sentralisert oversikt over konfigurasjon med versjoner, tryggere endringer under drift og en enklere og mer automatisert prosess for forvaltning av komponentene i infrastrukturen.

# 5 Teknologier

I IT-verdenen finnes det utallige teknologier der flere sikter mot å løse de samme problemstillingene. Dette fører til at det finnes en rekke alternativer å velge mellom der de fleste dekker de største behovene. Basert på forrige kapittel (se kapittel 4) vil dette kapittelet fokuserer på å gi en generell innføring for teknologier som kan brukes innenfor området for denne oppgaven.

## 5.1 Orkestreringsverktøy

Ellera benytter i dag ServeTheWorld som sin leverandør av tjenester, men har planer om å gå over til UniWeb. Etter å ha vært i kontakt med begge leverandører, har det blitt konkludert med at ingen av disse støtter opp under kravet programmerbar. Dette er et essensielt kriterie for å oppnå en programmerbar infrastruktur som benytter et orkestreringsverktøy.

Nedenfor er det en oversikt over de orkestreringsverktøyene gruppen har valgt å se nærmere på.

### 5.1.1 OpenStack Heat

OpenStack er et program for bygging og administrering av skyplattformer innenfor offentlige og private skyløsninger. Heat er OpenStack sin versjon av orkestrering. Det lar brukerne beskrive/definere distribusjoner av komplekse sky-applikasjoner i tekstfiler som kalles templates/maler. Disse malene blir deretter analysert og utført av heat-engine [2].

OpenStack er primært tenkt på som en privat skyløsning og for dette kreves det en god del ressurser innad i en bedrift eller organisasjon.

### 5.1.2 CloudFormation

CloudFormation er utviklet og driftes av Amazon og er deres versjon av orkestreringsverktøy. Det er tett integrert med og støtter kun AWS. Denne tette integrasjonen gir et praktisk brukergrensesnitt med oversikt over stack til en bestemt konto. Dette forenkler også referering på tvers av stackene, som er en enorm gevinst for modularitet og bryter opp monolitten.

### 5.1.3 Terraform

Terraform er i likhet med teknologiene nevnt over et orkestreringsverktøy. Terraform har støtte for utallige leverandører, hvorav 93 av leverandørene med offisiell støtte [22], samtidig rundt 111 med støtte fra samfunnet (community) [23]. Dette strekker seg fra skygiganter som AWS, Azure og Google til VCS, nettverk, database og monitoreringssystem. Terraform har mulighet til å benytte eksisterende skytjenester på tvers av leverandørene, samt kombinere disse med interne løsninger som OpenStack og Azure stack.

## 5.2 Konfigurasjonssytem

Nedenfor er det oversikt over konfigurasjonssystemene som er skal vurderes.

### 5.2.1 Puppet

Puppet er et rent konfigurasjonsverktøy som baseres på server/agent arkitektur. Mange konfigurasjonsverktøy benytter en slik løsning, hvor det installeres puppet-master på en maskin, samtidig som puppet-agenter installeres på resterende maskiner i infrastrukturen. All kommunikasjon mellom mester og agent foregår over SSL. Før en vellykket utveksling av informasjon kan finne sted må sertifikater mellom mester og agent signeres.

Forholdet mellom server og agent fungerer slik at serveren sitter på all konfigurasjon som skal hentes av agentene [24]. Ved initiell konfigurasjon vil mester pushe ut til agentene. Deretter vil disse kjøre periodisk der konfigurasjonen sammenlignes, og nye definisjoner hentes fra en sentral server.

### 5.2.2 Ansible

Ansible er et komplett provisjoneringsverktøy. Det gjør alt fra orkestrering til konfigurering av servere, og er basert på en agentløs arkitektur hvor kun kontroll serveren trenger å installere programvare. De resterende maskinene trenger python og å kunne nås med SSH. De slipper å ha en agent kjørende som sørger for at nåværende tilstand sammenlignes opp mot en sentral servers ønsket tilstand. I motsetning til Puppet benytter Ansible som standard en-veis kommunikasjon, push-modell, til å provisjonere konfigurasjonen ut til serverne i infrastrukturen. Dette gjøres fra en sentral server som sørger for å oppnå ønsket tilstand. All kommunikasjon foregår over SSH.

Sammenlignet med Puppet, som kompilerer koden før den kjøres, utfører derimot Ansible arbeidsoppgaver i en spesifikk rekkefølge. Slik at ved syntaksfeil vil Puppet feile, mens Ansible vil eksekvere koden helt til det møter på syntaksfeilen.

## 5.3 Containersystem

### 5.3.1 Docker

Docker er et verktøy som ble laget for å gjøre bruken av containere enklere. Måten Docker fungerer på gjør det mulig å lage, levere og kjøre applikasjoner ved bruk av containere. En container kan sammenlignes med virtuelle maskiner, med unntaket at containere ikke er avhengig av en egen kernel. Containere deler og kjører på samme kernel som det underliggende operativsystemet, og dette er grunnen til at ressursbruken kan minimaliseres. Brukervennligheten Docker har og tilbyr for administrering av containere er grunnen til at Docker er blitt meget populært de siste årene, og er en av de mest brukte teknologiene for containere [25].

#### Docker Swarm

Docker swarm er et orkestreringsverktøy for Docker containere. Med swarm kan IT-administratorer og utviklere etablere og administrere et cluster av Docker noder som et enkelt virtuelt system. En swarm er med andre ord en gruppe noder som kjører Dock-

er, og som er forenet til et cluster [26].

Swarm manageren er ansvarlig for initiering av swarmen, eksekvering av kommandoer og autorisere andre maskiner/noder. En node kan enten tildeles rollen manager eller worker.

### 5.3.2 Kubernetes

Kubernetes er en platform for containere og mikrotjenester, og tilbyr en god del muligheter utover disse. Det er et open source system for automatisk levering av applikasjoner, skalering og administrering. Målet er å skape et system for applikasjoner som kan fungere på tvers av flere ulike cluster. Kubernetes kan benyttes sammen med flere typer containertjenester, som f.eks Docker.

## 5.4 Docker Hub

Docker Hub er verdens største bibliotek og samfunn for container image [27]. Det fungerer på samme måte som Git-repositories, men er laget for konfigurasjonsdata for containere. I tillegg til lagring av data, kan Docker Hub også teste og bygge images ut ifra konfigurasjonsdata fra en bruker.

## 5.5 Logging og monitorering

Dersom en infrastruktur skal fungere bør det være mulig å hente informasjon fra de forskjellige komponentene. For eksempel, dersom en database går ned, må dette opplyses om til en administrator og/eller eventuelt et CMS. Logging vil si å lagre data som gir informasjon om hva som foregår i en komponent. Feilsøking vil gå raskere, og problemer kan da utbedres snarest.

Monitorering er å overvåke tilstanden og ressursbruken til et system. Det er derfor et godt verktøy for å opplyse om uventede eller unormale tilstander. Det er mulig å sette opp grafer som illustrerer ressursene og hvordan disse endres over tid. Samtidig vil et varslingssystem være konfigurert slik at unormale verdier gir utslag og et varsel mottas [28].

### 5.5.1 Elastic Stack

Elastic Stack er et logging og monitoreringssystem bestående av flere deler. De opprinnelige delene av Elastic Stack var Elasticsearch, Logstash og Kibana. Derav det tidligere navnet ELK-stack. I senere tid har verktøyet blitt brukt mer og mer, og forskjellige utbedringer og spesielle funksjonaliteter har blitt implementert, som blant annet Beats. På grunn av flere komponenter ble akronymet ELK ikke like lett å ta i bruk for flere komponenter, og navnet Elastic Stack ble tatt i bruk istedenfor [29].

### 5.5.2 Prometheus

Prometheus er et open-source monitorering- og varslings-verktøy med et aktivt utviklings- og brukermiljø. Den har i likhet med Elastic Stack, flere komponenter som kan velges ut ifra hva som kreves. Prometheus server, Alertmanager og Push gateway er noen av disse komponentene. Prometheus har fokus på pålitelighet og avhenger ikke av annet enn sin

egen server for å monitorere systemer [30].

## 5.6 Message queue og cache

Cache og message queue er like teknologier ved at de begge er en lagringsenhet, men de kan skilles ved funksjonalitet og hensikt. Hovedfunksjonalitet for caching er å redusere tiden det tar å hente data lagret i andre tregere lagringsenheter. Den gjør dette ved gjenbruk av tidligere prosessert data [31]. Hovedfunksjonalitet for message queue er å mitigere tap av data, ofte på grunn av tidvis manglende kapasitet i systemet. Den bidrar til løse koblinger mellom enheter i et system, og jevnere flyt i kommunikasjon mellom disse [32].

### 5.6.1 RabbitMQ

RabbitMQ er en message queue, som vil si at den forvalter kommunikasjon mellom applikasjoner i form av meldinger. En stor fordel ved bruk av RabbitMQ er muligheten for asynkronitet, som vil si at den ikke behøver å kommunisere med sender og mottaker til samme tid. Det er laget for pålitelighet og fleksibillitet, og kan tilføye en infrastruktur med gode muligheter for skalering eller håndtering av større kapasitet for kommunikasjon [33].

### 5.6.2 Redis

Remote Dictionary Server kan bli brukt til mye innenfor lagring. Redis er en open source nøkkelverdi-database og beskrives som en data struktur server. Det er et simpelt, men meget effektivt lagringsverktøy, og brukes til caching, message-queue, og ikke minst atomiske operasjoner, ved for eksempel inkrementering av tellere.

Spesielle funksjoner for Redis er:

- Høy-nivå datastrukturer
- Høy ytelse for lese- og skrive-operasjoner
- Lettvekt, og ingen avhengighet av annen programvare
- Høy tilgjengelighet med innebygget støtte for asynkronitet

Redis brukes av kjente selskaper som Twitter, Pinterest og Github [34].

### 5.6.3 Kafka

Kafka er beskrevet som en distribuert streaming plattform. Resultatmessig fungerer den på samme måte som en message queue, men den prosesserer data og struktureres annerledes. Lagring av data struktureres i form av topics. Kafka kjøres som et cluster på en eller flere servere hvor delene i clusteret har forskjellige jobber for lagring og prosessering av data.
Kafka består av fire APIer:

- Producer, forvalter lagring av data til en eller flere topics
- Consumer, forvalter lesing og henting av data til en eller flere topics
- Streams, samler lagring og lesing av data. Altså forvalte både lesing og skriving til en eller flere topics

- Connector, forvalter og lager nye Producers og Consumers

De spesielle egenskapene til Kafka gjør at den er meget skalerbar, og strukturen for lagring av data har større garanti for at sammehengende data er lagret på samme sted [35].

## 5.7 CI/CD Pipeline

### 5.7.1 Jenkins

Jenkins er et open source automasjonsverktøy som er skrevet i Java, der hensikten er rettet mot continuous integration. For programvareutvikling benyttes Jenkins som et verktøy for å bygge og teste prosjekter kontinuerlig, noe som gjør det lettere å integrere endringer. Jenkins bruker plugins for at det skal kunne utføre continuous integration, for ulike verktøy eksisterer det egne plugins for Jenkins. Dette kan være verktøy som git, Amazon EC2 og HTML. Fordelene med Jenkins er mange. Jenkins er open source slik at hvem som helst kan benytte seg av det og helt gratis. Fler enn 1000 ulike plugins eksisterer allerede og er klare til bruk. Dette gjør at Jenkins kan integreres sammen med mange andre verktøy. Hvis nødvendige plugins ikke eksisterer, så er det mulig å utvikle disse selv for å så dele de med resten av samfunnet [36].

### 5.7.2 Bitbucket

Bitbucket er en av flere typer git-repositories. Det fungerer som et samlingspunkt for data, hovedsakelig i form av kode, men også for andre formål som konfigurasjonsfiler. Bruk av git-repositories gjør deling, endring og henting av kode til en velfungerende prosess mellom flere brukere [37]

## 5.8 Databasesystem

Et databasesystem består av database og databasehåndteringssystem(DBMS). En database er der data lagres. Hvordan det lagres og struktureres styres av DBMS. DBMS forvalter og gjør data tilgjengelig for å leses, skrives og oppdateres [38].

Databaser kan strukturere data på 2 måter:

- Relasjonelt(SQL)
- Ikke-relasjonelt(NoSQL)

### 5.8.1 SQL

Structured Query Language også kjent som SQL er et databasespråk som brukes for å kommunisere med relasjonsdatabaser. Relasjonsdatabaser bygger på tabeller som er forbundet med hjelp av nøkler mellom seg. Hver tabell består av rader og kolonner der hver rad har en unik verdi som identifiserer raden, også kalt nøkkel. Fordelene med SQL og relasjonsdatabaser er at data kan lettere kategoriseres, slik at spørringer kan benyttes for å hente ut og filtrere eksakt data. SQL benyttes i hovedsak for å hente ut, legge til, slette, eller oppdatere informasjon [38].

# 6 Begrunnelse for valgt teknologi

## 6.1 Orkestrering

Felles for orkestreringsverktøyene er at disse brukes til å deklarativt definere en infrastruktur og dens ressurser med hjelp av definisjonsfiler. Ved å foreta en vurdering av disse finner man ut hvilken som kunne egnet seg i en løsning. Siden oppdragsgiver selv benytter en offentlig skytjeneste, og ikke har nok fysiske ressurser til å ta nytte av en privat løsning, velger gruppen å utelukke OpenStack Heat fra denne vurderingen.

### 6.1.1 Hvorfor ikke Ansible?

Som nevnt i seksjon 5.2.2 er Ansible et *komplett provisjoneringsverktøy*. og har moduler til å jobbe opp mot skyleverandører for å provisjonere og opprette en infrastruktur. Spørsmålet en da kan stille: *Hvorfor ikke bruke Ansible til konfigurering og orkestrering av tjenester?* Grunnen til å ikke bruke Ansible til oppsett og provisjonering av infrastrukturressurser, er fordi et konfigurasjonsverktøy er designet for å konfigurere og oppnå en ønsket tilstand ved å gjøre endringer på eksisterende infrastrukturressurser. Slike verktøy beholder ikke en historikk over tidligere endringer. Problemet med å ikke ha en slik oversikt over tilstanden til infrastrukturen er at det blir vanskelig å foreta endringer.

Selv om det kan virke enklere å forholde seg til kun et verktøy for oppsett og konfigurasjon av infrastruktur, vil det kunne føre til en mer kompleks løsning. Noe som ikke ville vært tilfellet ved bruk av et verktøy med evnen til å bevare tilstanden til infrastrukturen [39, 40].

### 6.1.2 CloudFormation og Terraform

Tidligere kjennskap av teknologi vil ikke bli vektlagt i dette tilfellet, ettersom hverken CloudFormation eller Terraform er teknologier gruppen har erfaring med fra tidligere.
I likhet med hverandre er både CloudFormation og Terraform orkestreringsverktøy brukt til å automatisere distribueringen av servere og infrastruktur. Det er likevel forskjeller som muliggjør et valg av disse.

CloudFormation er et mer modent alternativ som har mulighet for automatisk tilbakerulling til siste fungerende versjon av infrastruktur, hvis endringer måtte forårsake feil. I og med at det er tett integrert med Amazons mange tjenester, tilbyr de også teknisk støtte og kompetanse med *AWS support*. Sammenlignet med CloudFormation så har ikke Terraform i dag muligheter for automatisk tilbakerulling, slik at skulle det oppstå uønskede konsekvenser må en slik tilbakerulling skje manuelt. Terraform har derimot evnen til å håndtere og integrere med flere skyleverandører. En slik fleksibilitet lar oppdragsgiver fritt velge leverandør basert på deres behov, i stedet for å være låst til en skyleverandør.

Det har blitt sett på hovedforskjellen mellom CloudFormation og Terraform, ettersom

man vil oppnå samme resultat ved bruk av begge kandidatene har det derfor blitt utelatt å se på de tekniske ulikhetene. Selv om Terraform fortsatt er i utvikling og umodent i forhold til CloudFormation, med tilnærmet 1500 GitHub issues, er det fortsatt mye kompetanse tilgjengelig i form av dokumentasjon og forumer. Å ha mulighet til å deployere ressurser på tvers av skyleverandør er en av de store fordelene med Terraform, og er grunnen til at det konkluderes med at en løsning med Terraform antas å være fordelaktig [41, 42].

## 6.2 Provisjonering

Til å provisjonere ut installasjon og konfigurasjon av programvare på serverne vil det bli benyttet et CMS. For å kunne ta i bruk et slikt verktøy var det viktig for gruppen å komme i gang med vurderingen, og valgte derfor å se nærmere på Ansible og Puppet, etter en utvalgsprosess (se vedlegg A).

### 6.2.1 Pull vs Push

Ansible er push-basert og Puppet er pull-basert, men begge har muligheter til å benytte hverandres metode. Før valg av CMS kan ble det først vurderes om push- eller pull-arkitektur passer best for infrastrukturen.

#### Push

Push konfigurasjon baserer seg på en sentral server ansvarlig for å provisjonere ut og konfigurere servere i en infrastruktur. Dette gjøres ved å koble seg til serverne over nettverket, vanligvis ved bruk av SSH. Kun den sentrale maskinen bestemmer når det skal synkroniseres og orkestreres [43, 44, 45].

Fordeler

- Enkel å sette opp
- Mer kontroll på konfigurasjonsstyring

Ulemper

- Begrenset håndtering av skalering
- Begrenset håndtering av automatisering

#### Pull

Pull konfigurasjon bruker en agent installert på administrerte servere for å planlegge og synkronisere endringer. Agenten sjekker opp mot en mester eller et sentralt repository, for å laste ned de siste konfigurasjonsdefinisjonene og deretter bruke dem [43, 44, 45].

Fordeler

- Gode muligheter for automatisering
- Mer skalerbarhet enn push

Ulemper

- Eget konfigureringsspråk
- Høyere vanskelighetsgrad på forvaltning enn push

#### Sikkerhet

Pull-modellen har en enklere sikkerhetsmodell sammenlignet med push. Med push-modellen må hver administrerte server gjøre det mulig for mesteren å koble seg til og fortelle hvordan den skal konfigurere seg selv. Dette kan åpne opp for ondsinnede angrep. Utfordring som må vurderes: Sørge for nettverks konnektivitet og kommunikasjon tilgjengelig mellom mester og node, samt begrense tilgang gjennom nettverket og brannmur.
Med et pull system er det fortsatt nødvendig å sikre et sentralt repository som inneholder konfigurasjon definisjoner. Administrerte servere trenger ikke å åpne opp porter for innkommende forespørsler, samtidig som innloggingsinformasjon ikke blir gjort tilgjengelig.

#### Sammendrag

En pull-basert arkitektur håndterer skalering bedre med tanke på at hver administrerte server automatisk henter(puller) konfigurasjonen, og har muligheter for lastbalansering om det er nødvendig [46].

Push- og pull-arkitektur skalerer ikke like bra. Det kan bli problematisk i en større push-infrastruktur med et stort antall servere [47]. Ikke nødvendigvis på grunn av belastning på push-masteren, men heller tiden det ville tatt å utføre en kjøring på alle serverne. Selv om dette ikke vil påvirke Ellera i nærmeste fremtid, nevnes det for å vise til løsninger for å håndtere større infrastrukturer med en push-arkitektur. Pipelining kan brukes for å fremskynde SSH-tilkoblnger, forking for håndtering av parallelle prosesser og asynkronisert kjøring av oppgaver.

Infrastrukturen skal være så enkel og automatisert som mulig. Ved å velge bort pull-modellen velges en enklere infrastruktur, men med mangler for skalering og automatisering for oppdatering av programvare styrt av CMSet. På den annen side er det ikke nok pågang på systemet til Ellera til at skalering er nødvending i dag. Automatiseringen fra CMSet ville bare styrt oppdatering av Docker-programvare, og en jevnlig synkronisering fra en master i en pull-arkitektur, vil være unødvendig. Derfor vil det i dette tilfellet fokuseres på enkel bruk og en mindre kompleks push-arkitektur.

### 6.2.2 Sammenligning

Med push-modellen finnes det muligheter både for Ansible og Puppet. Som standard er Puppet pull-basert, men har en egen push-basert variant ved navn Bolt [48]. Ansible er et mye omtalt og populært verktøy, og gruppen har kjennskap og erfaring med Puppet fra tidligere i studiet. Med tanke på tidsaspektet til prosjektet ville det vært fordelaktig å velge Puppet, som allerede er kjent for gruppen, til konfigurering av servere.

Selv om Puppet har et rikere utvalg av moduler enn Ansible, og mange av disse er skrevet av Puppet selv, falt det endelige valget for bruk av konfigurasjonssytem på Ansible. Puppet Forge [49] sin Docker swarm modul støttet ikke opp under gruppens krav til en dynamisk løsning og håndtering av *swarm join-tokens*. Hovedsakelig var den avgjørende forskjellen variabel håndtering. Sammenlignet med Puppet, har Ansible innebygd funksjonalitet som enkelt kan hente ut og lagre returverdier fra modulen i variabler.

Skulle Ellera i fremtiden skalere med flere servere, slik at en foreslått løsning med push-

basert Ansible ikke lenger vil være like aktuell som i dag, bør det foretas en revurdering på dette området.

## 6.3 Containertjeneste

Valget og ønsket om containertjenesten ble satt i kravspesifikasjonen av oppdragsgiver. Oppdragsgiveren ville at det skulle benyttes Docker som teknologi og verktøy for å kjøre applikasjoner i containere. Docker kan benyttes selvstendig eller i samarbeid med andre teknologier som f.eks Kubernetes. Selv om det virker som at Docker og Kubernetes er tilsvarende teknologier, så opererer de på ulikt nivå i stacken.

### 6.3.1 Utfordringer

Ved administrering av flere containere er det en rekke utfordringer. Hvis Docker skal kjøre flere containere på flere maskiner samtidig så kan det fort oppstå problemer. De må ha mulighet for kunne snakke med hverandre, og de kan befinne seg på samme VM eller ulike maskiner. All informasjon lagret i en container vil være tapt dersom den skulle feile og ikke kunne startes igjen. Om dette måtte håndteres manuelt ville det vært utrolig komplisert og lite brukervennlig for administrering.

### 6.3.2 Orkestrering av Containere

For orkestrering av containere har Docker en egen teknologi som heter Docker swarm. Verktøy som Kubernetes kan også benyttes for orkestrering av containere. Docker Swarm er et orkestreringsverktøy for containere og blir sett på som en lettere og raskere vei for orkestrering enn Kubernetes. En swarm er et sett med noder som orkestreres av en master node, og inneholder minimum en master node. Som standard vil en manager også fungere som en worker. Det er mulig å ha flere manager noder for redundans, men det er ikke anbefalt med fler enn syv. I et cluster med flere noder vil det som regel ikke bli satt flere arbeidsoppgaver på en manager node, da disse vil forbeholdes til worker nodene. For at en Docker Swarm skal utføre den ønskede oppgaven defineres en service, denne inneholder et Docker image som skal benyttes av containerne og kommandoene som skal utføres. For å sette opp et miljø i Docker kreves kun kjennskap til Docker sitt CLI og verktøy [50].

Nøkkelfunksjoner ved Docker swarm er:

- Administering av cluster
- Skalering
- Bevaring av ønsket tilstand
- Lastbalansering
- Sikkerhet

Kubernetes er laget for å støtte store antall containere, gjerne over flere cluster, med fokus på automatisk administrering, levering og skalering av container-baserte applikasjoner. Containere blir gruppert sammen i logiske grupper for administrering. Kubernetes krever manuell installasjon, samtidig som skikkelig planlegging må utføres for at det skal kunne brukes. Installasjon er forskjellig for ulike operativsystemer, noe som krever ekspertise innenfor flere felt [50].

Nøkkelfunksjoner ved Kubernetes er:

- Administring av lagring
- Resource monitoring
- Helsesjekk av containere
- Skalering
- Tjenesteoppslag
- Rullende distribusjon og tilbakerulling

### 6.3.3 Hvorfor benytte Docker Swarm

Funksjonalitetsmessig tilbyr Docker swarm og Kubernetes mye av det samme. Hovedargumentet for å bruke Docker swarm over Kubernetes går på kompleksitet, her har Docker Swarm store fordeler. Kubernetes er kanskje den beste løsningen for bedrifter som har nok resursser og virkelig kan investere i det på lang sikt [51]. For en mindre bedrift som Ellera, som har få ansatte vil det være lønnsomt å starte med Docker Swarm da det er mulighet for å få igang noe fort og som likevel kan tilfredsstille behovene. Den tenkte infrastrukturen til Ellera kommer til å inneholde et fåtall noder, noe som styrker valget av Docker Swarm. Fremtidige krav til skalering og antall noder vil ikke være en utfordring da Docker Swarm støtter opp mot tusen noder uten problemer [26].

## 6.4 Logging og monitorering

En infrastruktur i drift krever oversikt over ressurser og trafikk. Det er allikevel en balanse for hva som trengs av oversikt, og hva som er unødvendig. Tracing, logging og monitorering krever ressurser for å bearbeide informasjonen de håndterer. Infrastrukturens gjøremål påvirker det som er nødvendig å ha en oversikt på, og hvilke teknologier som passer for å styre dette [52].

### 6.4.1 Tracing, logging og/eller monitorering?

Det finnes tre hovedkategorier for å ha kontroll på en infrastruktur.

**Tracing** er en kontinuerlig oversikt på hva spesifikke brukere gjør i en applikasjon eller et system. Det brukes for ytelsesoptimalisering [52, 53].
**Logging** håndterer hovedsakelig feilmeldinger, og data relatert til om systemet fungerer som det skal [52, 53].
**Monitorering** handler om å samle data og bruke den samlede dataen til å enten varsle, analysere og/eller visualisere dataene [52, 53].

Tracing er ikke nødvendig for dette prosjektet, men feilmeldinger ved logging er essensielt for drifting av infrastrukturen over lenger tid. Sammen med aggregering og visualisering ved bruk av monitorering, vil det være mulighet for oversikt over logger og varsling for kritiske feil, samlet på et sted.

### 6.4.2 Elastic Stack og Prometheus

På grunn av tidligere erfaring med verktøyene Prometheus og Elastic Stack, velges disse som kandidater til håndtering av logger og/eller metrikk i infrastrukturen. Komponentene i infrastrukturen skal driftes i Docker, og verktøyene må også være kompatible med Docker. Både Prometheus [54] og Elastic Stack [55] oppfyller dette kravet.

### Prometheus

Gihub stars: 23 987 [56]

Beskrivelser fra brukere på Stackshare [57]

- Kraftig og lett å bruke
- Fleksibelt språk for spørringer
- Dimensjonsmodellering

### Elastic Stack

Github stars: Fra 10 000 og høyere [58, 59, 60]

Beskrivelser fra brukere om delene av Elastic stack [61]

- Kraftig API
- Gratis
- Lett å bruke

Infrastrukturen skal inneholde en webserver, som vil si at den skal være mulig å nå fra internett. Informasjon om brukernes innlogging, og om innloggingsprosessen fungerer som det skal, vil håndteres som hendelseslogger. Både fra et driftsperspektiv, men også fra et sikkerhetsperspektiv, vil derfor hendelseslogger være nødvendige deler av kontinuerlig drift. Det vil si at bare metriske data ikke er nok for å ha oversikt over for eksempel feilede innloggingsforsøk, og utvidet informasjon om denne hendelsen. Prometheus håndterer hovedsakelig metrikk, og skaperne av Prometheus sier selv at Elastic Stack i stedet for bør brukes på dette bruksområde [62]. Derfor vil Elastic Stack brukes videre i prosessen.

## 6.5 Message Queue for Elastic Stack

Kafka, RabbitMQ og Redis er tre ofte nevnte teknologier for message queue for Elastic Stack [63, 64, 65] For å velge en teknologi brukes brukeropplevelser fra vanlige brukere, samt Trello sin opplevelse av de forskjellige teknologiene.

### Redis

- Høyeste brukerantall sammen med Elastic Stack ifølge Stackshare [66]
- Elastic Stack har en plugin for teknologien [67]
- Brukt av Trello tidligere, men byttet ut på grunn av manglende funksjonalitet for større og mer krevende infrastrukturer.
- Trello antyder at dersom manglende funksjonalitet blir implementert i Redis, så vil teknologien brukes igjen [65]
- Beskrivelser fra brukere fra Stackshare [66]
    - God ytelse
    - Rask
    - Enkel å bruke

### RabbitMQ

- Elastic Stack har en plugin for teknologien [68]

- Brukt av Trello, men byttet ut med Kafka på grunn av ustabilitet [65]
- Beskrivelser fra brukere på Stackshare [66]
    - Rask og fungerer bra med monitorering
    - Lett å konfigurere
    - Bra brukergrensesnitt

**Kafka**

- Ingen plugin fra Elastic Stack
- Brukes i dag av Trello på grunn av kapasitet for store infrastrukturer [65]
- Beskrivelser fra brukere på Stackshare [66]
    - Større kapasitet for mottak av data
    - Distribuert
    - Skalerbar

Redis er den mest brukte teknologien for Elastic Stack blant de tre utvalgte. Trello sin erfaring er at Redis ville vært det mest effektive verktøyet, hvis versjonen Redis Streams var sluppet på tidspunktet av sammenligningen [65].

Både Kafka og RabbitMQ fungerer bedre for større infrastrukturer enn Redis, men det er ikke et kriterie for dette prosjektet. Ellera har kjennskap til Redis fra tidligere, og det ikke er gode nok grunner til å bruke noe annet for dette prosjektet. Redis blir derfor tatt i bruk som verktøy for message queue for logger til Elastic Stack.

## 6.6 CI/CD Pipeline

Det finnes flere måter å strukturere en CI/CD pipeline. Utvalg av teknologier for infrastrukturen avgrenser hva som kan brukes. Docker skal brukes, så testing og bygging av image er en forutsetning. Dette gir to muligheter for CI/CD metode:

- Jenkins som en komponent i infrastrukturen, med testing av konfigurasjon fra repository på utsiden av infrastrukturen
- repository som sentralisert konfigurasjonslagring, med webhook mot Docker Hub som tester og bygger image

Begge metodene kan utføre automatisert testing og bygging av image. Den avgjørende forskjellen gjelder at Jenkins må være installert og konfigurert, samtidig som den krever ressurser fra infrastrukturen for å gjennomføre konfigurasjonsendringer. For å holde infrastrukturen så liten som mulig og redusere ressursbruk, velges metoden for webhooks mellom infrastrukturen, Bitbucket og Docker Hub.

# 7 Systemdesign

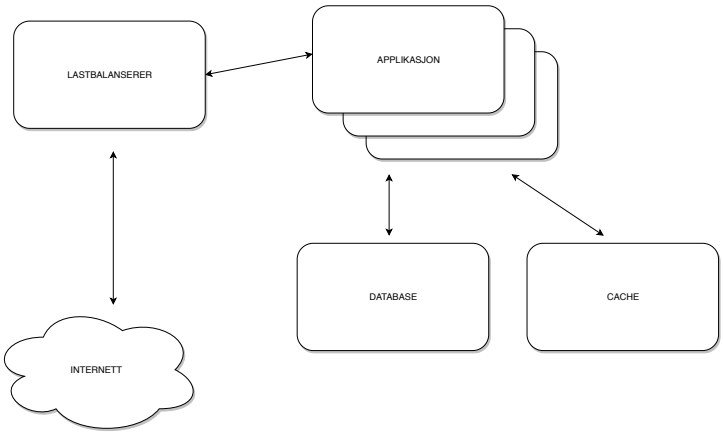
Infrastrukturen og komponentene som skal implementeres, tar utgangspunkt i hvordan infrastrukturen til Ellera var ved inngåelse av oppdragsavtale. Dette nye designet skal beskrive en infrastruktur som er mer automatisert og enklere å bruke. Designet skal også gi administrator god oversikt over komponentene i form av monitorering og logging, samtidig som endringer utført i konfigurasjon for Docker-miljø skal kunne gjøres raskere, enklere og tryggere i form av en CI/CD pipeline.

## 7.1 Arkitektur

Systemet moddeleres basert på en trelagsarkitektur (se figur 2), denne arkitekturen er en av de vanligste innenfor infrastruktur. Modellen deler infrastrukturen inn i flere lag, et offentlig og to private:

- Presentasjon (Offentlig)
- Applikasjon (Privat)
- Data (Privat)

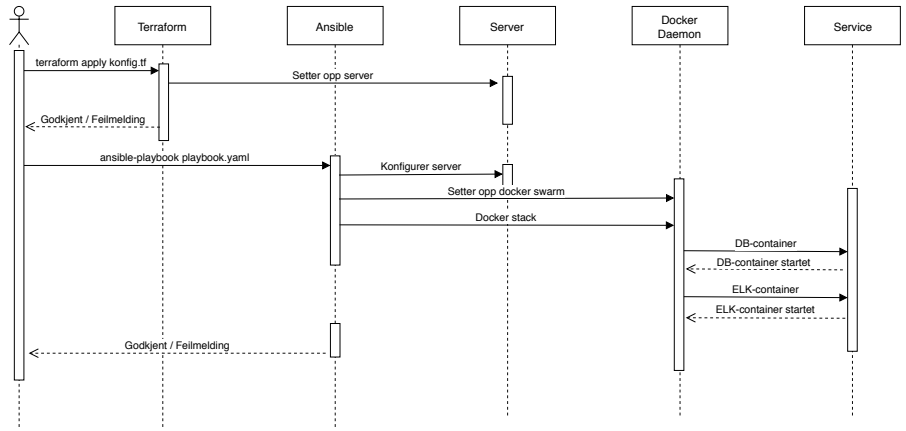
Ideen bak dette er at det er kun det offentlige laget som er tilgjengelig fra internett, og de private kun fra innsiden av nettverket. Presentasjonslaget er tilgjengelig via et brukergrensesnitt som ofte representeres av en nettleser. Applikasjonslaget representerer en applikasjon og funksjonaliteten denne tilbyr. Datalaget holder på informasjonen til tjenesten og gjør denne tilgjengelig for applikasjonen, ofte ved bruk av MySQL, PostgreSQL eller MongoDB [69].




Figur 2: Arkitektur

### 7.1.1 Sekvensdiagram

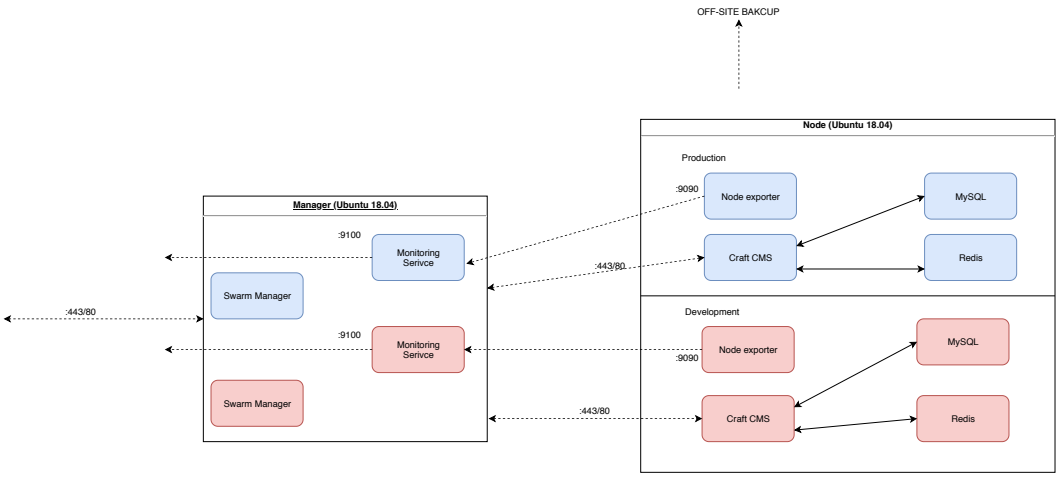
Overordnet flyt ved oppsett av infrastruktur kan en se i figur 3. Når administrator initierer Terraform blir komponenter som bygger opp en infrastruktur, som f.eks servere satt opp ved bruk av Application programming interface (API)-kall mot en skyleverandør. Når serverne er ferdig-provisjonert vil administrator starte Ansible som setter i gang med konfigurering og installasjon av programvare, i dette tilfellet Docker. Ansible vil med installasjon av Docker sette opp og koble provisjonerte servere sammen i et Docker Swarm cluster. Med et cluster konfigurert, vil Ansible trigge Docker til å sette opp en stack.




Figur 3: Flyt oppsett av infrastruktur

## 7.2 Provisjonering av containere

Kjernen i infrastrukturen vil kjøre i Docker Swarm. Manageren vil motta en tjenestedefinisjon og sende arbeidsoppgaver ut til workerne. Manageren har ansvaret for orkestrering, og administrering av clusteret for å opprettholde ønsket tilstand til swarmen.




Figur 4: Docker arkitektur

Ved å ta i bruk CI/CD pipeline som prosess for testing og bygging av image for workernoden(e), kan endringer i applikasjonen gjøres tryggere. Alle nye endringer må følge en

fastsatt prosess før den nye koden kan settes i produksjon. For at koden skal nå helt ut i produksjon, må den bestå alle testene i prosessen. Ved feil vil prosessen stoppe, og koden vil hindres fra å kunne implementeres.

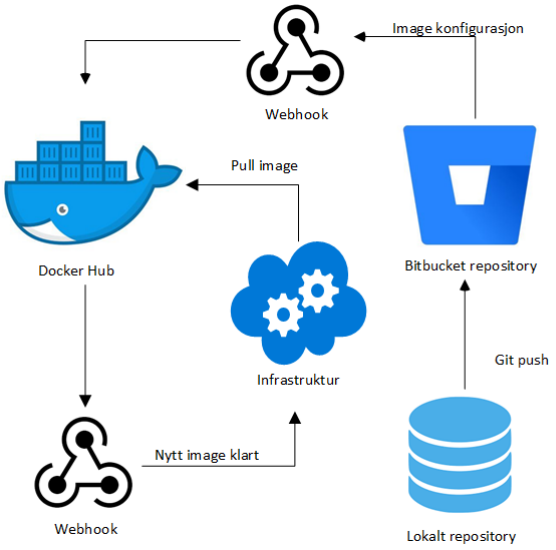
### 7.2.1 Utfordringer ved datalagring i Docker Swarm

**Database:** Siden nodene kjøres som swarm, vil det ved flere worker-noder føre til uoverenstemmelser for lagring av data i databasene for hver node.

**Off-site backup:** containere er praktiske både på grunn av optimalisering av ressursbruk, men også fordi containere som oftest vil lages på nytt dersom de feiler. Bruk-og-kast metoden fungerer ikke for en backup-løsning, og oppdragsgiver har derfor som krav at backup skal tas off-site.

## 7.3 CI/CD for Docker image

Hver komponent i infrastrukturen har sitt eget image. Endringer i konfigurasjon, og naturligvis lagring, foregår i Bitbucket. Dette gjør versjonskontroll og eventuell tilbakerulling til tidligere versjoner oversiktlig og enkelt. Repoene er igjen koblet til hvert sitt Docker Hub-repository ved hjelp av webhooks. For hver git-push til et repo, vil Docker Hub automatisk hente ny konfigurasjon og bygge et nytt image, så lenge den nye konfigurasjonen ikke inneholder helt grunnleggende feil. Dersom testing er implementert, vil tester også gjennomføres, og vil forhindre bygging av nytt image hvis testing feiler. Ved ferdig bygget image, er det også mulighet med enda en webhook fra Docker Hub, som gir beskjed om at imaget er klart til bruk.
Arbeidsflyten er illustrert nedenfor.




Figur 5: Flyt av CI/CD for Docker Image

### 7.3.1 Sikkerhetsmomenter

I forbindelse med Docker og Docker Hub finnes det risikoer med tanke på oppdateringer av images. På grunn av at et image ofte baserer seg på et eller flere andre images, kan det inkluderes sikkerhetshull uten at det oppdages. Dette kalles for Supply Chain Attacks. Hvis det gjøres endringer på et image, oppdateringen inneholder svakheter og applikasjonen oppdateres ved bruk av imaget vil de svakhetene også bli innført i applikasjonen. Ved en automatisert prosess for oppdatering av image i en tjeneste er det fullt mulig at dette oppstår, og det kan være vanskelig å oppdage. For å mitigere denne risikoen bør det kun tas utgangspunkt i gode kilder for images, på Docker Hub vil disse være merket som Certified eller Docker Official Images. For at et image skal bli Certified må det følge best practice og bestå tester. En annen måte er å ha et privat repository som inneholder *Dockerfile* og *.yaml* som imaget bygges på. Med tanke på alt dette vil det benyttes private repositories og for tjenester som MySQL vil det benyttes Docker Official Images fra Docker Hub [70].

## 7.4 Webserver & Craft CMS

Craft CMS er selve sluttproduktet i tjenesten og blir gjort tilgjengelig via web servere. Web serverene settes opp via containere som kjører i Docker. Docker og Docker Swarm bidrar til at antallet tilgjengelige webservere kan skaleres og distribueres på de tilgjengelige nodene i infrastrukturen. Manageren vil passe på at et bestemt antall webservere alltid skal være tilgjengelig, og sette opp nye containere om nødvendig. Docker Swarm sørger for intern lastbalansering mellom tilgjengelige containere i et cluster, disse vil tilhøre den samme tjenesten. Craft CMS er kun et rammeverk for å designe websider og er derfor avhengig av en programvare for webservere, som for eksempel Nginx eller Apache2. På grunn av dette vil Nginx benyttes og konfigureres slik at Craft CMS vil bli tilgjengelig over port 80.

### 7.4.1 HTTPS

For at systemet skal følge best practice er det nødvendig å kryptere trafikken som går til og fra webserverene. Implementering av HTTPS står for dette. Selv om systemet ikke nødvendigvis behandler sensitiv data betyr det ikke at kommunikasjonen ikke behøver kryptering. HTTPS hjelper med å opprettholde integriteten til datakommunikasjonen samtidig som den øker sikkerheten, og er et krav for mange nettlesere og funksjoner [71].

### 7.4.2 HSTS

HTTPS Strict Transport Security er et webserver direktiv som informerer klienter om hvordan de skal behandle tilkoblingen sin via en response header. Denne setter verdien for det som kalles for Strict-Transport-Security policy field parameter. Når denne er satt vil tilkoblingen bli tvunget til å foregå over HTTPS. Ved bruk av denne vil alle script som prøver å laste inn ressurser over HTTP bli forkastet og trafikken vil krypteres via HTTPS. HSTS vil også beskytte mot Man-In-The-Middle attacks [72].

## 7.5 Database

Et krav fra oppdragsgiver var at databasen skulle implementeres med en løsning for containere. Database som kjøres i container kan fungere godt nok i et utviklingsmiljø, men kan by på problemer om den benyttes i et produksjonsmiljø. Dette har å gjøre med at databaser er kritiske tjenester der feil og nedetid kan ha katastrofale følger for tjenesten [73].

Tilgjengelige metoder for å skalere en container-basert databaseløsning eksisterer, men de bygger på teknologier som er beregnet for cluster med flere hundre noder og derfor anses som urelevant for prosjektoppgaven [74].

Løsningen vil fungere, men vil begrense størrelsen på tjenesten og muligheten for videre skalering. For å løse problemer med skalering og imøtekomme nye krav for redundans og ytelse vil det være en fordel å flytte databasen ut fra containeren og heller benytte VM. Hvis virtuelle maskiner benyttes i stedet for containere vil det være mulig å benytte databasearkitekturer som replikering og database cluster. Disse arkitekturene består av to eller flere databaser og derfor også redundans. Ved flere databaser vil det også være mulig å lastbalansere mellom de tilgjengelige nodene for økt ytelse.

## 7.6 Logging og monitorering

Loggingen fungerer ved bruk av Elastic Stack. Filebeat installeres på Craft CMS og Nginx- og database-containerne, og har ansvaret for videresending av relevante logger [75]. Logger lagret lokalt på hver komponent blir videresendt av Filebeat til Logstash.

Ved store og krevende infrastrukturer med mange komponenter, burde det også vært en meldingskø (message-queue) som kontrollerer mengden med trafikk sendt videre til Logstash.

Logstash har ansvar for aggregering og prosessering av logger [76]. Aggregering er allerede i gang ved at Filebeat sender logger til et sted, nemlig Logstash. Logstash sender deretter behandlede logger videre til Elasticsearch for lagring. Prosessering gjøres ved at innkommende logger, i form av en streng av tekst, blir delt opp av Logstash til relevante deler.
Dette er en eksempel-linje fra en access-log til Nginx, som viser at leseligheten kan bli bedre:

```
10.22.206.242 - - [28/Mar/2019:07:29:52 +0000] "GET / HTTP/1.1" 200 241
"-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:64.0) Gecko/20100101
Firefox/64.0" "-"
```

Konfigurasjonen for behandling av logger blir definert i Logstash. Konfigurasjonen gir retningslinjer for hvordan Logstash skal prosessere data den får tilsendt. Dette gjøres for økt oversikt og søkbarhet. Dersom denne prosesseringen ikke ble gjort ville lagringen i Elasticsearch bare lagret hele logg-linjen uten noen form for struktur, som ville gjort logging til et nærmest ubrukelig verktøy. Logstash skaper denne strukturen utifra konfigurasjonsfilene sine, og gjør det mulig å kategorisere ved for eksempel alle logger som

gjelder et tidspunkt.

Data ferdig prosessert i Logstash sendes videre til Elasticsearch, som lagrer loggene. Data lagret her kan søkes opp, sammenlignes og analyseres. Om ønskelig kan alle logger for hele infrastrukturen lagres her, som oppnår logg aggregering med mulighet for oversiktlig og enkel analyse av samtlige logger.

For visualisering og brukergrensesnitt mot logg-analyse kan Kibana brukes i tillegg. Kibana henter data ved å søke i og hente data fra Elasticsearch, og kan i tillegg til god oversikt over loggene, brukes til visualisering av data. Kibana er monitoreringsverktøyet i Elastic-stacken, og kan gi sanntids oversikt over flere hendelser og dataflyt i infrastrukturen.

# 8 Implementasjon

Fra tidligere Begrunnelse for valgt teknologi (se kapittel 6), blir det lagt grunnlag for beskrivelse av denne delen av rapporten, som tar for seg implementasjon.

- Terraform (v0.11.13)
- Ansible (v2.7)
- Docker (v18.09.3)
- Elastic Stack (6.6.1)

## 8.1 Forutsetninger

Oppsett av infrastrukturen forutsetter kunnskap innenfor de valgte teknologiene nevnt ovenfor. Kunnskap ble tilegnet under selve arbeidet av prosjektet, og kildene for disse har i hovedsak vært offisiell dokumentasjon og tutorials.

I tillegg forutsetter implementasjonen tilgang til maskinressurser, og i dette tilfellet ble skolens (NTNU Gjøvik) egne skytjeneste – SkyHiGh – benyttet. Siden SkyHiGh bruker OpenStack, ble det også naturlig å komme med en løsning til favør for dette.

## 8.2 Terraform

Terraform har blitt brukt til å definerer og sette opp infrastrukturens komponenter, virtuelle maskiner, nettverk osv. Terraform konfigurasjonsfiler brukes til å administrere, og lage en beskrivelse av topologien til infrastrukturen. Dette gjøres ved hjelp av et høynivå domenespesifikt språk kjent som *Hashicorp Configuration Language* (HCL). Konfigurasjonsfilene identifiseres ved forlengelsen `.tf`.

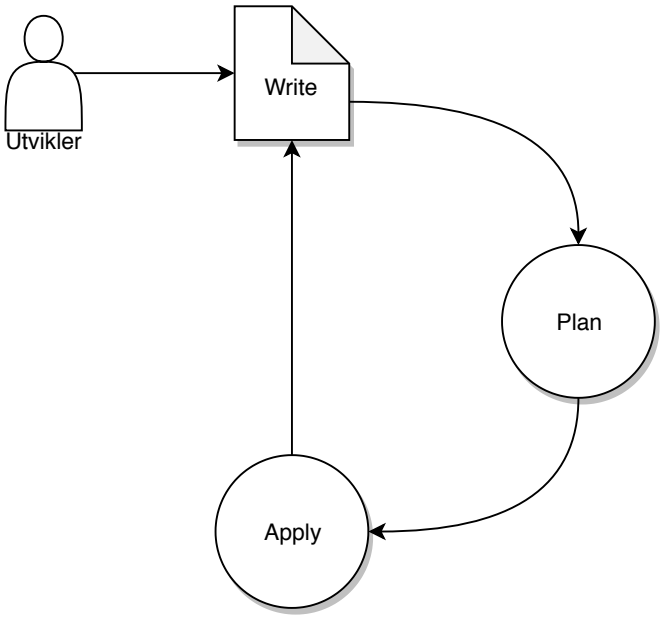
Terraforms oppgave er å generere en 'gjennomføringsplan', dette er en beskrivelse over hva som skal implementeres for å oppnå den ønskede tilstanden beskrevet i konfigurasjonsfilene.

### 8.2.1 Arbeidsflyt

Utvikling og deployering av en Terraform-infrastruktur kan beskrives med tre steg:

1. **Write** - Definere infrastrukturen i domenespesifikt språk (`HCL`).
2. **Plan** - Gjennomgang av endringer før de iverksettes.
3. **Apply** - Provisjonere endringer til en reproduserbar infrastruktur.

Figur 6 viser en kontinuerlig prosess hvor Terraform-konfigurasjon blir skrevet og endret (*write*) – på samme måte som med kode. Etter hver endring vil gjentatte sjekker (*plan*) sile ut syntaksfeil og sørge for at konfigurasjonen blir som forventet. Ved endelig bekreftelse (*apply*) vil endringer av infrastrukturen gjennomføres [77].




Figur 6: Arbeidsflyt

Selv om den initielle konfigurasjonen skrives i definisjonfiler, utføres resten ved hjelp av kommandoen *terraform* i kommandolinjen:

```
#Skrive innledende konfigurasjon
   vim example.tf
#Installere moduler & plugins
   terraform init
#Genererer en gjennomføringsplan
   terraform plan
#Bygger eller endrer infrastrukturen
   terraform apply
#Sletter Terraform-administrert infrastruktur
   terraform destroy
```

### 8.2.2 Tilstand

Resultatet fra en initiell syklus (se figur 6) er en tilstandsfil, *terraform.tfstate*. Terraform benytter denne filen til å kartlegge ressurser, holde rede på metadata og tilstanden til infrastrukturen. Gjøres det påfølgende endringer eller modifikasjoner på infrastrukturen vil Terraform refere til og oppdatere *terraform.tfstate*.

Tilstandsfilen lagres per definisjon lokalt på maskinene som utfører kommandoen `terraform plan`. Hvis det er flere personer på samme infrastruktur og alle benytter hver sin unike tilstandfil, så vil ikke filene inneholde lik informasjon som kan skape problemer. Grunnen til dette er fordi tilstandsfilen oppdateres ved endringer av infrastrukturen, og dersom

hver enkelt person benytter sin egne tilstandsfil så vil filen oppdateres kun for den som utfører endringen. Gjøres det påfølgende endringer med hvor filene er usynkronisert, så vil nye endringer fjerne/slette deler av infrastrukturen. For å gjøre informasjon om tilstanden til infrastrukturen tilgjengelig for flere personer og samtidig unngå inkonsistent informasjon om infrastrukturen, kan det være fordelaktig å lagre tilstandsfilen eksternt. En slik abstraksjon kalles *backend*. Backend bestemmer hvordan tilstanden skal lastes, og muliggjør eksekvering for alle i prosjektgruppen [78].

Listing 8.1: Eksempel av Azure Blob Backend

```
1 terraform {
2   backend "azurerm" {
3     storage_account_name = "abcd1234"
4     container_name       = "tfstate"
5     key                  = "prod.terraform.tfstate"
6   }
7 }
```

Eksempelet over bruker en backend til å lagre, bruke og oppdatere tilstanden til infrastrukturen i en Azure Blob. Det er en av mange alternativer Terraform tilbyr. De fleste backender støtter *tilstands-lås* som forhindrer *race condition* og sjekker konsistenthet av data. Mer om dette i *Sikkerhetsaspekter* (se seksjon 8.2.3)

For denne implementasjonen har tilstandfilen blitt lagret lokalt, men gruppen velger å nevne bruk av backend for å belyse muligheten, men også problematikken for oppdragsgiver.

Før man kan ta i bruk Terraform er det nødvendig å konfigurere en tilkobling opp mot en skyleverandør. Dette gjøres ved hjelp av en *Provider* block.

Listing 8.2: Eksempel OpenStack Provider

```
1 # Konfigurere OpenStack Provider
2 provider "openstack" {
3   user_name   = "admin"
4   tenant_name = "admin"
5   password    = "pwd"
6   auth_url    = "http://myauthurl:5000/v2.0"
7   region      = "RegionOne"
8 }
```

*OpenStack provider* benyttes for å kommunisere med ressursene som brukes av OpenStack. Provider må konfigureres med tilstrekkelig kredentialer før den kan benyttes. Dette kan gjøres enten (se listing 8.2) ved å legge kredentialer direkte i konfigurasjonsfilen, eller et mer hensiktsmessig alternativ, benytte miljøvariabler. Ved bruk av miljøvariabler åpner man blant annet opp for versjonskontroll av infrastrukturen offentlig uten å risikere eksponering av sensitive kredentialer.

Terraform har sin egen måte å håndtere miljøvariabler, og vil søke gjennom arbeidsmiljøet til sin egen prosess etter miljøvariabler kalt `TF_VAR_`, etterfulgt av navn på deklarert variabel.

Det viktigste elementet i Terraform-språket er *Resource*. En resource blokk beskriver et eller flere infrastrukturobjekter. I listing 8.3 er det eksempel på hvordan opprettelse av manager instansen gjøres. (se vedlegg B.2):

Listing 8.3: Manager instans definisjon

```
resource "openstack_compute_instance_v2" "manager" {
  name            = "Manager"
  image_name      = "${var.image}"
  flavor_name     = "${var.flavor}"
  key_pair        = "${var.kp}"
  security_groups = [
    "default",
    "${openstack_compute_secgroup_v2.Dockers.name}",
    "${openstack_compute_secgroup_v2.ssh.name}",
    "${openstack_compute_secgroup_v2.http.name}",
  ]

  network = {
    uuid = "${openstack_networking_network_v2.
       network_11.id}"
  }

  user_data = "${data.template_cloudinit_config.config.
     rendered}"
}
```

Resource blokken deklarerer en resource av type `"openstack_compute_instance_v2"`, som blir gitt et lokalt navn: "Manager". De fleste av ressursene har muligheter til å definere konfigurasjonsargumenter [79], som vist i listing 8.3 over blir konfigurasjonsargumentene som er tilgjengelige for OpenStack instansen brukt. Eksempler på dette er `name`, `image`, `security_groups` osv. Konfigurasjonsargumentene avhenger av ressurstypen.

Et annet viktig alternativ er bruk av *meta-argument*. Dette kan brukes på de fleste ressurstyper for å endre ressursens oppførsel. Eksempelet i listing 8.4 – opprettelse av Node-instansene (se vedlegg B.2) – bruker meta-argumentet *count* for å lage flere instanser av samme type.

Listing 8.4: Node instans definisjon

```
resource "openstack_compute_instance_v2" "node" {
  count           = "${var.server_count}"
  name            = "Node${count.index}"
  image_name      = "${var.image}"
                  ...
}
```

Eksempel på bruk av variabler er `"${var.image}"`. Slike variabler er definert et annet sted, typisk i en fil kalt `variables.tf`. Det er også mulig å benytte data fra en annen ressurs. `"${openstack_compute_secgroup_v2.dockers.name}"` er et eksempel på hvordan data fra en resource definert i listing 8.5 kan hentes og brukes i konfigurasjonsfilene (se vedlegg B.3):

Listing 8.5: Eksempel sikkerhetsgruppe

```
resource "openstack_compute_secgroup_v2" "dockers" {
  name        = "Docker_swarm"
  description = "Ports for Docker swarm"
              ...
}
```

Her brukes det lokale navnet `dockers` for å referere til resourcen innad i den lokale modulen. Det foretrekkes å gjøre det på denne måten, ettersom hardkoding ikke anses som best practice.

Typen *Data* gjør det mulig å hente og beregne data for senere bruk andre steder i Terraform-konfigurasjonen, enten eksternt eller lokalt i ressurser som *Cloud-init*.

Cloud-init brukes til å utføre oppgaver ved oppstart av virtuelle servere, eksempelvis installasjon av programvare som Ansible. Det ble i dette tilfellet brukt til å legge til SSH-nøkler, samt for å kjøre installeringsscript på provisjonerte servere.

Listing 8.6: CloudInit template

```
data "tls_public_key" "example" {
  private_key_pem = "${file("~/.ssh/test")}"
}

data "template_file" "script" {
  template = "${file("~/templates/add_keys.sh.tpl")}"

  vars = {
    pvt_key = "${data.tls_public_key.example.
       private_key_pem}"
    pub_key = "${data.tls_public_key.example.
       public_key_openssh}"
  }
}

data "template_cloudinit_config" "config" {
  gzip          = true
  base64_encode = true

  part {
    filename     = "add_keys.sh"
    content_type = "text/x-shellscript"
    content      = "${data.template_file.script.
       rendered}"
  }
}
```

I listing 8.6 benyttes *template*, dette er en annen type datakilde som gjengir en mal som lastes fra en ekstern fil. Vist i resourcen ovenfor refereres det til templaten ved bruk av `user_data` etterfulgt av datakilden. Her defineres det et cloud-init skript som laster ned nødvendig programvare og legger til privat nøkkel for å oppnå konnektivitet til de andre nodene.

Det er med metodene nevnt i denne seksjonen som har blitt benyttet til å definere hovedkomponentene som utgjør infrastrukturen. Dette gjelder server instanser, nettverk, subnettverk, sikkerhetsgrupper og flytende IP-er. Alt dette ved bruk av HashiCorp's domenespesifikke-språk, "HCL"[80]. Se vedlegg B.

### 8.2.3 Sikkerhetsaspekt

**Tilstandfil**

Tilstandsfilen Terraform brukes til å holde oversikt og kartlegge ressurser. Den kan potensielt inneholde sensitiv informasjon om infrastrukturen, og det er ikke ønskelig å gi uautoriserte aktører innsyn til denne. Skal denne filen deles mellom flere personer for å bevare konsistenhet, kan en løsning være å holde tilstandsfilen i en ekstern backend, som gjøres utilgjengelig for offentligheten. En videre forbedring på dette ville vært å helhetlig automatisert Terraform ved å benytte continuous deployment-miljø [81].

## 8.3 Ansible

Ansible fungerer ved å koble seg til noder og pushe ut små programmer, kalt *Ansible moduler*. Disse programmene fungerer som generaliserte maler for ressursene som skal brukes i systemet [82].

Det baseres på en predefinert liste over maskiner som skal administreres, kalt *inventory*. Hver maskin blir tildelt en *role*, et rammeverk for samlinger av moduler, variabler, oppgaver og filer. Hver role får en *task* som skal utføres. En task kan være installasjon av programvare, konfigurasjon eller sørger for at programmer kjøres idempotent.

Som best practice anbefaler Ansible å definere en role per tjeneste, samtidig ha en baserole som laster ned programvare som er nødvendig for alle maskinene. I en *playbook* tildeles maskinene en eller flere roles basert på forhåndsdefinerte grupper og *facts*. Facts er informasjon samlet inn fra maskinene, som gjør det mulig å installere programvare basert på operativsystemet, eller installere programvare for en bestemt gruppe, for eksempel programvare til webserverne.

### 8.3.1 Filstruktur

Alle filene Ansible benytter må være skrevet i YAML, med unntak av inventory-filen. Enhver definert role er en underkatalog av *roles* katalogen. Hver role har følgende underkataloger:

- *Tasks*: Definerer et sett med oppgaver som skal utføres av en role. En task er et kall på en ansible modul. Eksempel på dette kan innebære nedlasting pakken "Apache", hvor modulen som kalles er "apt"(en packet manager ansvarlig for nedlasting og sletting av programvare på Debian-baserte systemer).
- *Templates*: Inneholder konfigurasjonsparametre. Definert i et Jinja2 format, et templatespråk.
- *Files*: Statiske filer, dette kan være skript, binær eller en annen type som vil bli kopiert til serveren.

- *Vars*: Variabler som vil bli brukt til å fylle ut abstraksjoner i tasks og templates. Her kan det spesifiseres versjonsnummer for installasjon av en bestemt pakke, disse kan også overskrives.
- *Defaults*: Standard verdier for variabler som ikke defineres i en playbook.
- *Handlers*: Aktivere en task basert på gitte betingelser. Handlers er i bunn og grunn tasks som aktiverer *faktiske* tasks. Handlers kan brukes til å restarte en tjeneste etter det har blitt kopiert over konfigurasjonsfiler, slik at nye endringer vil bli tatt i bruk.
- *Meta*: Informasjon om en role, hvilke type platform den støtter, kompabilitet og avhengigheter. Hvis en role er avhengig av en annen vil det bli deklarert her.

**Prosjektstruktur**

Når det gjelder mappestruktur til prosjektet, så er dette basert på Ansibles best practice-eksempel [83]. Selve strukturen har blitt endret på for å tilpasse prosjektets behov, og selv sier Ansible at oppsettet er kun ment som et utgangspunkt.

«*Your usage of Ansible should fit your needs, however, not ours, so feel free to modify this approach and organize as you see fit* [83]».

```
Ansible/
├── ''playbook.yml''
├── ''inventory.py''
├── group_vars/
│   ├── all.yml
│   └── manager/
│       ├── vars
│       └── vault
└── roles/
    ├── base/
    │   └── tasks
    │       └── main.yml
    ├── docker/
    │   ├── handlers/
    │   │   └── ''main.yml''
    │   └── tasks/
    │       └── ''main.yml''
    └── stack/
        ├── library/
        │   └── ''docker_stack.py''
        └── tasks/
            └── ''main.yml''
```

### 8.3.2 Arbeidsflyt

Typisk arbeidsflyt ved en ny role er å planlegge hva en role skal gjøre, og bryte dette opp i mindre oppgaver. Nedenfor blir det gjennomgått oppbyggingen av Docker role (se vedlegg C.3):

**Role: Docker**

Før installering av applikasjonen blir GPG-nøkkel lagt til:

Listing 8.7: Legge til GPG-nøkkel

```
1    name: Add Docker GPG key
2    apt_key:
3      url: https://download.Docker.com/linux/ubuntu/gpg
4    become: yes
```

Feltet `name` behandles i dette tilfellet som en kommentar og blir skrevet ut til skjerm når Ansible kjører. `apt-key` beskriver modulen som brukes til å installere og legge til GPG-nøkkelen, samt parameteret `url` hvor nøkkelen skal hentes fra. Det blir brukt `become` til å forsikre om *privilegie-eskalering*, slik at oppgaven utføres med administratorrettigheter.

Deretter legges APT repository til:

Listing 8.8: Legge til APT-repo

```
1    - name: Add Docker APT repository
2    apt_repository:
3      repo: deb [arch=amd64] https://download.Docker.com/
           linux/ubuntu {{ansible_distribution_release}}
           stable
4      state: present
5    become: yes
```

I listing 8.8 ser man et lik oppsett som i listing 8.7, med unntak av `apt_repository` modulen som brukes til å legge til apt repo, og bruk av fakta –`{{ ansible_distribution_release }}` – som brukes til å hente passende apt-repository basert på kjørende operativsystem til maskinene i infrastrukturen.

Installasjon av Docker:

Listing 8.9: Installasjon av Docker

```
1  - name: Install Docker
2    apt:
3      name: Docker-ce
4      state: latest
5      update_cache: yes
6    become: yes
7    notify:
8      - restart_docker
```

Oppgaven over vil sørge for å installere nyeste, `state: latest` versjon av Docker, samt `update_cache: yes` for å oppdatere lokal cache til systemet. Alt utføres med – `become` – administratorretigheter. I tillegg benyttes `notify` til å kalle på en *handler* så lenge oppgaven resulterte i en endret tilstand (*changed state*).

Handler til å restarte Docker-tjeneste:

Listing 8.10: Restart handler

```
1  - name: restart_docker
2    service:
3      name: Docker
4      state: restarted
5    become: yes
```

I listing 8.10 brukes ikke `name` som en kommentar til hva som utføres, men heller som et funksjonsnavn. Ved et kall må verdien til `notify` i listing 8.9 være eksakt lik som navnet til handleren. Handleren sørger for å restarte Docker tjenesten som kjører på maskinen.

Tildeling av role:

Listing 8.11: Tildele role

```
1  - hosts: all
2    roles:
3      - base
4      - docker
```

Måten en role tildeles på i Ansible er ved å deklarere hvilke maskiner som skal endres med `hosts:`, for deretter å spesifisere hvilke role skal gjelde for maskinene ved bruk av `roles:`. I listing 8.11 ser man at alle maskinene ble tildelt `base` og `docker` role.

### Oppsett av Docker Swarm

En essensiell del av infrastrukturen er oppsett av Docker Swarm og orkestrering av containere. Ansible er ansvarlig for dette, og har allerede en eksisterende modul [84] som brukes til å initialisere og opprette et Docker Swarm cluster. Denne brukes også for å legge til/ fjerne noder eller managere fra et eksisterende cluster.

Et cluster vil initialiserers av en manager node, denne vil annonsere en IP-adresse som benytter standardporten for Docker Swarm, 2377. Dette clusteret gjøres tilgjengelig for andre noder i infrastrukturen, men for at nodene skal kunne ta del i clusteret må det benyttes en *join-token*. Token lages av manager ved opprettelse av clusteret. I listing 8.12 initialiseres Docker swarm clusteret og maskinene som legges til vil fungere som reserve-managere.

Listing 8.12: Initialisering av Docker swarm

```
1  - hosts: manager
2    tasks:
3      - name: Get Swarm information
4        docker_swarm:
5          state: inspect
6        register: swarm_info
7        become: yes
8
9      - name: Initialize a swarm
10       when: swarm_info.swarm_facts is not defined
11       docker_swarm:
```

```
12          state: present
13          advertise_addr: '{{ ansible_default_ipv4.
                address }}'
14        become: yes
15
16      - name: Get latest Swarm info
17      # https://github.com/ansible/ansible/issues/15710#
            issuecomment-216645922
18        docker_swarm:
19          state: inspect
20        register: swarm_info
21        become: yes
22
23      - name: Get join token
24        set_fact:
25          join_token: '{{ swarm_info.swarm_facts.
                JoinTokens.Manager }}'
26        become: yes
```

I listing 8.12 blir `register` brukt til å lagre Docker swarm informasjon i variabelen `swarm_info`. Dette blir gjort for å kunne sjekke med bruk av `when: swarm_info.swarm_facts is not defined`, og dermed forsikre seg at det ikke finnes et eksisterende cluster ved initialisering av en swarm.

I listing 8.13 blir maskinene i infrastrukturen lagt til i swarmen ved å bruke den IP-adressen til manageren-noden:
`{{ hostvars[groups['manager'][0]].ansible_default_ipv4.address }}:2377"`, og ved bruk av den annonserte join-token `join_token: "{{ hostvars[groups['manager'][0]].join_token }}` (se vedlegg C.1).

Listing 8.13: Join Docker swarm

```
1  - hosts: docker:!manager
2    tasks:
3      - name: Get current swarm state
4        docker_swarm:
5          state: inspect
6        register: swarm_info
7        become: yes
8
9      - name: Join swarm
10       when: swarm_info.swarm_facts is not defined
11       docker_swarm:
12         state: join
13         advertise_addr: "{{ ansible_default_ipv4.
               address }}"
14         remote_addrs:
15           - "{{ hostvars[groups['manager'][0]].
                 ansible_default_ipv4.address }}:2377"
16         join_token: "{{ hostvars[groups['manager'][0]].
               join_token }}"
17       become: yes
```

For å kjøre tjenester på tvers av Docker Swarm blir Docker Stack benyttet. Ettersom Ansible 2.7 ikke har offisiell støtte for Docker Stack, så har en modul for dette blitt manuelt inkludert fra GitHub. En oppdatering av Ansible til versjon 2.8 vil legge til denne modulen som standard funksjonalitet (se vedlegg **??**).

**Terraform-inventory**

Med en dynamisk infrastruktur er det vanskelig å oppdatere og vedlikeholde en statisk inventory. Dette skalere ikke, og vil derfor ikke lønne seg ved bruk av dette i lengden. *Terraform inventory* er et verktøy som ble brukt til å trekke ut IP-adressen til maskinen og grupperinger fra terraform tilstandsfilen, og genererte en dynamisk inventory som kunne brukes av Ansible til provisjonering [85]. Selv om verktøyet fungerte i sin helhet, skapte mindre feil et behov for å benytte et eget skript (se vedlegg D) [86] til å hente ut kun nødvendig informasjon.
Eksempel på slik feil: Terraform-inventory hentet ut lokal IP fra tilstandfilen, når flytende-IP var ønskelig.

(Alternativt finnes det flere ansible inventory skript for dynamisk inventory tilpasset de ulike skyleverandører fra forskjellige bidragsytere [87].)

### 8.3.3 Sikkerhetsaspekt

Sikkerhet er utvilsomt en viktig del å ta i betrakning i en infrastruktur, vilkårlig av hvilke teknologier og verktøy som blir tatt i bruk.

**SSH**

Som nevnt innledende i seksjon 5.2.2 bruker Ansible til vanlig SSH-protokollen til å autentisere tilkoblinger og kryptere kommunikasjonen. SSH som en "industristandard", anses å være generelt sikker, så lenge nøkler over 2048 bits eller mer brukes [88, 89].

Tilgang til Ansible kontroll-maskinen gir potensielt kontroll over hele infrastrukturen, og det er derfor viktig å sikre denne tilgangen.

For å sikre ekstern tilkobling så har maskinene blitt herdet til å tillate tilgang med bare bruk av offentlig nøkkel, men også fjernet ekstern tilgang til root bruker. Det er mulig å bruke flerfaktorautentisering ( *time-based one-time password*), slik at det legges til et ekstra lag med sikkerhet ved tilkobling opp mot serveren.

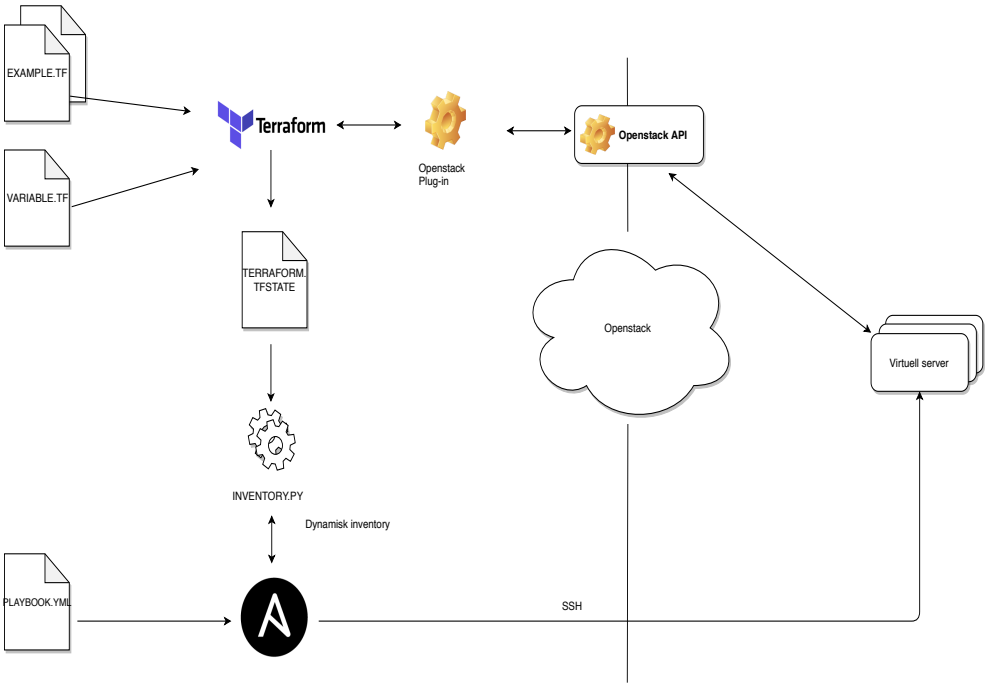
**Sensitive data**

Til å holde data skjult kommer Ansible med *Ansible Vault* som er blitt brukt til å kryptere sensitive data på filnivå. Ved å gjøre dette holder en data skjult, men muliggjør også bruk for versjonskontrollsystem.

## 8.4 Samarbeid mellom Terraform og Ansible

Følgende bilde viser hvordan komponentene i Ansible og Terraform samhandler ved oppsett av infrastrukturen. Terraform vil først (se seksjon 8.2.1) søke gjennom konfigurasjonen og sørge for installasjon av nødvendige moduler og plugins. Det vil så genereres en gjennomføringsplan (`terraform.tfstate`) med oversikt over hva som skal implementeres, for så å realisere dette. Deretter vil Ansible bruke inventarprosessen i skriptet

`inventory.py` til å lese tilstanden til provisjonert infrastruktur fra filen `terraform.tfstate`. Dette integreringspunktet lar dynamisk allokerte IP-adresser, vertsnavn og infrastrukturdetaljer bli sendt til Ansible som kan begynne å konfigurere maskinene.




Figur 7: Oppsett av infrastruktur

## 8.5 Craft CMS

Craft CMS installeres i containere, der hver enkelt container fungerer som en webserver. Containerene er basert på debian:stretch image fra Docker-Hub.

### 8.5.1 Installering

Nedlastningen av Craft 3 gjøres ved bruk av Composer, og det er gitt en guide til hvordan dette skal gjøres [90]. Når Composer har kjørt ferdig vil følgende mappestruktur være på plass under den angitte lokasjonen (se figur 8) . For at Craft skal kunne fungere som planlagt må rettighetene stemme slik at de nødvendige operasjonene kan utføres. Etter at Craft 3 er ferdig lastet ned må resten av avhengighetene settes opp slik at Craft kan installeres og settes i drift. Craft er avhengig av å kommunisere med en database, for eksempel PostgreSQL eller MySQL, samtidig som at det behøver en webserver som kan hoste tjenesten, for eksempel Nginx eller Apache2.

```
my-project.test/
├── config/
│   └── ...
├── storage/
│   └── ...
├── templates/
│   └── ...
├── vendor/
│   └── ...
├── web/
│   └── ...
├── .env
├── env.example
├── Composer.json
├── Craft
└── Craft.bat
```

Figur 8: Mappestruktur Craft CMS

### 8.5.2 Docker Image for Craft CMS

Selve oppsettet av containere for webserver med Craft CMS gjøres ved bruk av Docker og et Docker image (se vedlegg F.2) som legges på toppen av et eget image for Nginx-php-fpm. Innholdet i containerene bygges ut i fra flere ulike Dockerfiler. Som en base er det tatt utgangspunkt i eksisterende Docker image for Craft CMS og Nginx-php-fpm fra Docker Hub [91]. For at containerene som lages basert på disse imagene skal passe inn i infrastrukturen er det behov for justeringer av Dockerfilene.

### 8.5.3 Valg av Operativsystem

Docker imaget for php-fpm baseres på Debian:Stretch, som også kalles for Debian 9 [92]. Det finnes veldig mange muligheter når det kommer til valg av operativsystem for Docker containere, og det finnes ingen fasit på hva som er best. Hva som skal kjøres og bruksområde har mye å si [93]. Grunnlaget for å benytte Debian som operativsystem for containerene er mange. Debian støtter flere CPU arkitekturer som Intel og AMD. Leverandøren av Debian skryter av høy stabilitet, der maskiner vil være i drift i lang tid uten systemkrasj. Lavt bruk av systemressurser fører til at Debian har god ytelse. I motsetning til alt som er bra med Debian, er det ingen tydelige tegn på at Debian er et dårlig av operativsystem for containerene [94, 95].

### 8.5.4 PHP-FPM

Docker imaget for Nginx-php-fpm (se vedlegg F.1) bygger en base som inneholder avhengighetene som kreves av Craft CMS. I stedet for manuell installasjon er alle stegene for Craft CMS installasjonen bakt inn i et Docker image. Avhengighetene installeres gjennom apt install, og er for det meste pakker som kreves av komponentene slik at disse kan snakke sammen og utføre sine oppgaver. For at Nginx skal vite hva slags oppgave som skal utføres kopieres en ferdig oppsatt konfigurasjonsfil inn i mappen *conf.d* slik at den blir benyttet som standard. For endringer av konfigurasjonsfilene *php.ini* og *www.conf* benyttes *sed*, som er en stream editor der aktuelle linjer byttes ut med ønsket innhold uten behov for å kopiere inn filer utenfra. For nedlastning av Composer brukes komman-

doen curl, og installasjonsfilen hentes ned fra Composer sitt nettsted. Videre blir den lagt til i ønsket filplassering, slik at den kan refereres til senere. For å verifisere utgiveren av programvaren benyttes en PGP-nøkkel, om alt stemmer overens vil Composer installeres som planlagt.

### 8.5.5 Oppsett av MySQL

Databasen settes opp i en egen container. Craft 3 støttes av MySQL versjon 5.5 og oppover, noe som førte til at et Docker image for MySQL versjon 5.7 ble benyttet (se vedlegg F.4). For at Craft 3 skal kunne kobles til databasen må en databasebruker eksistere, samt en database. Gjennom konfigurasjonsfilene til Docker kan disse bestemmes på forhånd og sendes med som miljøvariabler, slik at alt settes opp automatisk. For sikker oppbevaring av brukernavn (`MYSQL_USER`) og passord (`MYSQL_ROOT_PASSWORD`), er disse blitt kryptert av Ansible Vault. Konfigurasjon av databasen er vist i listing 8.14. Innholdet i filen konfigurerer en container som blir kalt `db`. Databasen baserer seg på imaget mysql:5.7 og den skal ha `replicas: 1`. Miljøvariablene bestemmer navnet på selve databasen som skal benyttes, i dette tilfellet heter den Craft 3. Navn og passord på databasebrukeren ligger lagret i vault med tanke på sikkerhet rundt lagring av innloggingsinformasjon, som ikke bør være i klartekst. Porten som benyttes for kommunikasjon settes til 3306 som refereres til standardporten for MySQL. For at dataen som ligger lagret i containeren ikke skal blir borte om containeren går ned eller krasjer, konfigureres et volum. Volumet gjør at dataen for containeren ligger lagret utenfor slik at den ikke går tapt selv om containeren skulle forsvinne. Selve volumet ligger i Docker filplassering under den valgte mappen som bestemmes i konfigurasjonen.

Listing 8.14: Konfigurering av MySQL

```
1     db:
2         image: mysql:5.7
3         deploy:
4             replicas: 1
5             restart_policy:
6                 condition: on-failure
7         environment:
8             MYSQL_DATABASE: 'Craft3'
9         # So you don't have to use root, but you can if
              you like
10            MYSQL_USER: "{{ vault_db_user }}"
11        # You can use whatever password you like
12            MYSQL_PASSWORD: "{{ vault_db_password }}"
13        # Password for root access
14            MYSQL_ROOT_PASSWORD: "{{
                  vault_root_password }}"
15        # <Port exposed> : < MySQL Port running inside
              container>
16            ports: 3306:3306
17        # Where our data will be persisted
18        volumes:
19              - Craft3:/var/lib/mysql
```

### 8.5.6 Konfigurasjon av Nginx

Nginx konfigureres gjennom filen *default.conf* (se vedlegg G) der innholdet blir justert i forhold til ønsket funksjonalitet. Craft CMS og Nginx installeres i samme container. I det lokale testmiljøet settes porten som Nginx lytter på til port 80 med listen 80. Porten benyttes som standard for nettjenester for trafikk over HTTP, ved HTTPS brukes port 443.

Listing 8.15: Konfigurere http

```
1     listen   80; ## listen for ipv4; this line is
          default and implied
2     listen   [::]:80 default ipv6only=on; ## listen for
          ipv6
```

Root settes til å peke på destinasjonen for Craft CMS sine filer, som er applikasjonens dokumentrot. Slik vet Nginx hvilke filer som inngår i applikasjonen og hva som skal henvises til når det mottas forespørsler. For at indeksfiler som er av filtypen `.php` skal støttes legges dette til via `index` variabelen.

Listing 8.16: Konfigurere Root Folder

```
1     root /usr/share/Nginx/web;
2     index index.php index.html index.htm;
```

Med tanke på sikkerhet er det best practice at versjonen av Nginx ikke vises ved feilmeldinger. Ved standard konfigurasjon vises denne. Mulige angripere kan utnytte kjente feil i versjoner av Nginx, og derfor bør dette skrus av.

Listing 8.17: Konfigurere Server Token

```
1  server_tokens off;
```

Ettersom at Nginx ikke direkte støtter prosessering av PHP er det nødvendig å konfigurere en PHP-prosessor til dette formålet. Ved bruk av php-fpm er dette mulig. FastCGI prosessering i Nginx brukes for å oversette klientforespørsler for applikasjoner som ikke skal håndtere disse direkte [96].

Listing 8.18: Konfigurere FastCGI

```
1    location ~ \.php$ {
2          try_files $uri =404;
3          fastcgi_split_path_info ^(.+\.php)(/.+)$;
4          fastcgi_pass unix:/run/php/php7.3-fpm.sock;
5          fastcgi_index index.php;
6          include fastcgi_params;
7          fastcgi_param SCRIPT_FILENAME
               $document_root$fastcgi_script_name;
8          fastcgi_param PATH_INFO $fastcgi_path_info;
9      }
```

I mappene og roten til applikasjonen eksisterer det filer som ikke skal være tilgjengelige for å vises i nettleseren. For at disse skal gjøres utilgjengelig konfigureres det en adgangskontroll som stopper forespørslene for disse.

Listing 8.19: Konfigurere Adgangskontroll

```
1    location ~* (?:\.(?:bak|config|sql|fla|psd|ini|log|sh
        |inc|swp|dist)|~)$ {
2          deny all;
3          access_log off;
4          log_not_found off;
5  }
```

For å kunne kontrollere og overvåke flere prosesser inne i containeren konfigureres supervisor. Supervisor gir et sentralisert sted for start og stopp av prosesser,og gir samtidig mange muligheter for å bestemme lokasjoner for blant annet loggfiler. Filen *supervisord.conf* (se vedlegg H) legges med i containeren gjennom Dockerfilen.

## 8.6 Elastic Stack

Elastic Stack kjøres som en egen container i swarmen. Det vil si at Logstash, Kibana og Elasticsearch er installert på samme container. Grunnen til dette er at læringsversjoner av Elastic Stack ofte er samlet på en container, når det gjelder bruk i Docker. Det skal sies at det hovedsakelig er en stor ulempe med å samle tjenestene på en container. Dersom containeren går ned, vil alle tjenestene også naturlig nok gå ned. Derfor bør tjenestene i Elastic Stacken deles ut over flere containere.

For fordeler og ulemper ble det også undersøkt om kommunikasjon mellom containere ville forårsake forsinkeler for Elastic Stack. Denne forsinkelsen vil dersom den i det hele tatt eksisterer være så lav, at det ikke vil være en fordel for bruk av en container for hele Elastic Stack, som forsøket til Rothprimardho antyder [97].
Imaget brukt for Elastic Stack er basert på det populære imaget til Sebastien Pujadas [98]. Ubuntu er ikke laget for å kjøres i en container, og derfor er grunn-imaget til Pujadas basert på Phusion sin *baseimage-Docker*. Dette skal ifølge baseimage-Docker dokumentasjonen optimalisere ressursbruk og funksjonalitet for imaget [99].

Under samme container kjøres Elastic Stack under samme service ved navn `elk` (se listing 8.20). Det skreddersydde imaget hentes fra Docker Hub, og inneholder alle relevante konfigurasjoner for hvordan Elastic Stack skal fungere i infrastrukturen (se vedlegg F.3). Elastic Stack må linkes til andre komponenter i infrastrukturen for å ha mulighet til loggaggregering og monitorering. Docker swarm setter automatisk opp et swarm-nettverk for tjenester under samme compose-fil. Dette gir mulighet for å bruke Docker swarm sin integrerte service-discovery med hjelp av vertsnavn. Kommunikasjonen mellom nodene sammen med åpning av de relevante portene, gjør Elastic Stack klar for å motta data fra resten av swarmen.

I listing 8.20 vises hvordan dette settes opp i stack.yml (se vedlegg C.4).

Listing 8.20: Oppsett av Elastic Stack

```
1  services:
2      elk:
3        image: daniebru/ellera:latest
4        ports:
5          - 5601:5601
```

```
6           - 9200:9200
7           - 5044:5044
8         environment:
9         #WEB is linked
10          WEB_HOST: web
11        #DB is linked
12          DB_HOST: db
13        #REDIS is linked
14          REDIS_HOST: redis
```

Elastic Stack må ha tilgang til logger lagret lokalt på hver container i infrastrukturen. Konfigurasjon for fungerende logging har relativt lik fremgangsmåte for samtlige komponenter, og innebærer disse punktene:

- Konfigurasjon av logg-forvaltning for tjenesten eller tjenester i kontaineren.
- Modulen Filebeat installeres og konfigureres til å håndtere videresending av logger til Logstash.
- Konfigurering av logghåndtering for Logstash, derav filtrering, mottak fra Filebeat og segmentering av logg-strenger.

Denne gjennomgangen av implementeringen omhandler tjenesten Nginx med loggene av typen *access*. Det vil si loggene som gir innsikt i blant annet når og hvilken IP-addresse som besøker nettsiden gjennom Nginx. Filene følger ferdig oppsett fra Sebastien Pujadas [98].

### Konfigurering av logger for tjenesten

Logger konfigureres i *default.conf*. Her lagres logg av type access i valgt mappe ved navn *Nginx*, for videresending ved hjelp av Filebeat (se vedlegg G):

Listing 8.21: Plassering av logg

```
1  # Add logging to file
2      access_log /var/log/nginx/access.log;
```

### Konfigurering av Filebeat

Konfigurasjonen for hvordan Filebeat skal håndtere logger omhandler hvor Filebeat skal sende loggene(*Output*), og hvor loggene skal hentes fra (*Prospectors*). For autentisering mellom Filebeat og Logstash kan sertifikater brukes, som vist under(se vedlegg I.1):

Listing 8.22: Konfigurasjon av Filebeat

```
1  output:
2    logstash:
3      enabled: true
4      hosts:
5        - elk:5044
6      timeout: 15
7      ssl:
8        certificate_authorities:
9        - /etc/pki/tls/certs/logstash-beats.crt
10
11 filebeat:
12   prospectors:
```

```
13      -
14        paths:
15          - "/var/log/Nginx/access.log"
16        document_type: Nginx-access
```

### Konfigurering av Logstash

Logstash konfigurasjon omhandler filtrering av videresendte logger fra Filebeat. Om typen logg ikke samsvarer med det Logstash skal håndtere vil den ikke segmenteres slik Nginx-loggene skal segmenteres, (se vedlegg I.2).

Listing 8.23: Filtrering av logger

```
1  filter {
2    if [type] == "Nginx-access" {
3      grok {
4        match => { "message" => "%{NginxACCESS}" }
5      }
6    }
7  }
```

For loggene som er av samme type *Nginx-access* vil den segmenteres slik som vist i listing I.3 nedenfor:

Listing 8.24: Segmentering av Nginx-logg

```
1  NGUSERNAME [a-zA-Z\.\@\-\+_%]+
2  NGUSER %{NGUSERNAME}
3  NginxACCESS %{IPORHOST:clientip} %{NGUSER:ident} %{
       NGUSER:auth}
4      \[%{HTTPDATE:timestamp}\] "%{WORD:verb} %{
           URIPATHPARAM:request}
5      HTTP/%{NUMBER:httpversion}" %{NUMBER:response:int}
6      (?:%{NUMBER:bytes:int}|-) (?:"(?:%{URI:referrer}|-)
           "|%{QS:referrer})
7      %{QS:agent}
```

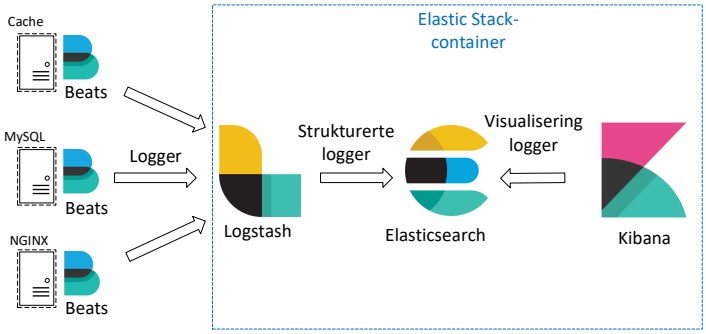
Med loggen segmentert i deler med navn, sendes dette til Elasticsearch for lagring. Loggen er søkbar, og kan søkes opp direkte fra Elasticsearch, eller ved hjelp av Kibana.

Arbeidsflyt for komponentene i Elastic Stack er vist i figur 9.

### 8.6.1 Sikkerhet

Elastic Stack er ikke klar til bruk i produksjon etter konfigurering av funksjonene den skal utføre. Den kommuniserer med andre komponenter som åpner flere angrepsvektorer i infrastrukturen, men har mulighet for sikkerhetsmekanismer som mitigerer disse. Sikkerhetsmomenter som autentisering, autorisering, sikker kommunikasjon, intern logging og integritets-mekanismer bør også konfigureres før infrastrukturen er produksjonsklar [100].

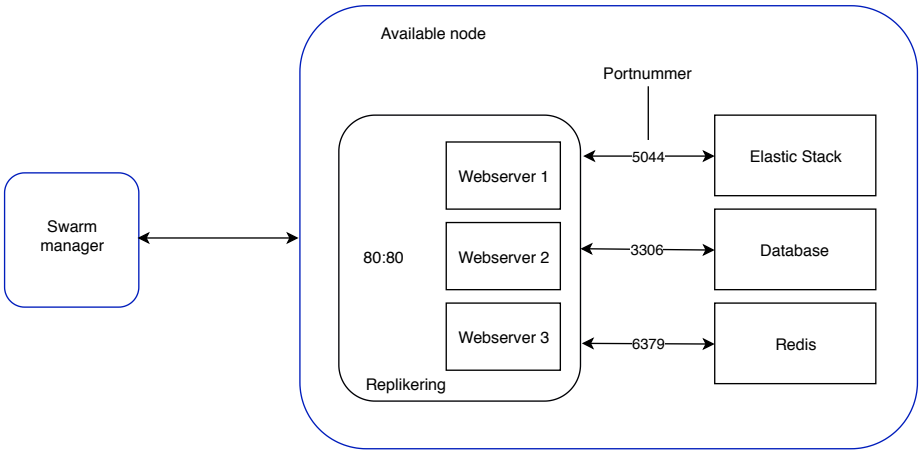
Sikkerhetsmomentene er beskrevet nærmere i kapittel 9.3.2




Figur 9: Elastic Stack oversikt

## 8.7 Resultat av implementering

Ved endt implementering av verktøy og teknologier, vil det være mulig å sette opp infrastrukturen ut i fra en automatisert prosess. Når denne prosessen kjører ferdig vil resultatet være som vist i figur 10.




Figur 10: Implementert Infrastruktur

Gruppen er klar over at infrastrukturen ikke vil være produksjonsklar, og har avklart mange av disse manglene (se seksjon 9.3).

# 9 Diskusjon

## 9.1 Hva kunne blitt gjort annerledes

Til å begynne med var vi raske med å legge en plan for hvor vi ville med oppgaven. I stedet for å fremskynde en slik planlegging burde mer tid blitt brukt sammen med veileder og oppdragsgiver, for å definere og konkretisere oppgaven. Resultatet av en slik forhastet beslutning var at definisjonen av oppgaven ble et sidespor, som førte til at mye arbeid ble utført uten at det førte til relevante resultater. Hadde vi kommet frem til en tydelig oppgave med klare målsettinger, ville det mest sannsynlig vært lettere for prosessen, planleggingen og veien videre.

I forhold til møter med veileder og oppdragsgiver burde vi hatt en tydeligere agenda for å utnytte mest mulig av tiden som var til rådighet. Det ble brukt mye tid på å utarbeide et formelt rammeverk som skulle ligge til grunn for vurdering, drøfting og sammenligning av diverse teknologier. Mye av denne tiden burde ha blitt rettet mot utvikling for å komme i mål med spesifiserte krav, slik at mer tid kunne prioriteres til den praktiske delen av oppgaven.

I løpet av prosjektperioden hadde vi en tendens til å delegere større oppgaver i stedet for å dele opp i mindre konkrete arbeidsoppgaver. Dette kunne forbedret motivasjon og arbeidsflyt underveis.

### 9.1.1 Bruk av Utviklingsmodell

Det var ikke et krav at Scrum skulle brukes som metode for prosjektet, men det virket som en passende modell. Til å begynne med ble det utviklet en fremdriftsplan og sprintene ble planlagt med tanke på denne. I startfasen viste det seg å være problematisk å utføre de planlagte arbeidsoppgavene, som en følge av vår vage oppgavedefinisjon.

Definisjonsproblemene med oppgaven ble etterhvert et problem for sprintene våre. Det ble foretatt kontinuerlig endringer av planen for gjennomføring av oppgaven, og vi ble nødt til å gå tilbake for å endre arbeidsoppgavene underveis. Dette resulterte i at vi ikke klarte å tydeliggjøre veien videre.

Etter at oppgaven endelig fikk tydligere fokus, ble det satt opp en ny sprint på to uker der vi hadde som mål å ferdigstille en fungerende prototype som skulle vises frem til veileder. Ved endt sprint var målet nådd. Den neste tiden var avgjørende for rapporten, der alt fokus ble rettet mot skriving.

Ved ettertanke mener vi at ingen andre utviklingsmodeller ville hatt noe avgjørende innvirkning på utviklingsprosessen sammenlignet med Scrum.

## 9.2 Manglende krav

Ved endt implementeringsfase ble ikke alle kravene fra prosjektets kravspesifikasjon oppfylt (se kapittel 2).

- Ekstern backup er ikke implementert
- Kryptering av trafikk gjøres internt i Docker swarm, men trafikk mot internett benytter HTTP.
- Caching settes opp via Redis i en egen container, men det er ikke konfigurert hva som caches
- Logging og monitorering håndterer kun logger for Nginx.
- Logger skulle lagres eksternt, men disse lagres lokalt i Elasticsearch.

## 9.3 Videre arbeid

Selv om den utviklede infrastrukturen fungerer og driftes uten kjente problemer er det fortsatt muligheter for forbedring og endringer som bør gjøres før den kan benyttes i produksjon. Dette gjelder krav som ikke er implementert, sikkerhetsmekanismer, og hvilke muligheter som eksisterer hos andre leverandører.

### 9.3.1 Ansible

Under oppsettet av infrastrukturen ble Ansible brukt til initiell konfigurasjon av serverne. For å oppnå en helautomatisert bruk av Ansible i en produksjonsetting burde verktøy som *Ansible AWX/Ansible Tower* undersøkes videre for bruk med tanke på tidsplanlegging av arbeidsoppgaver, og opprettholde ønsket tilstand [101, 102].

### 9.3.2 Elastic Stack

Elastic Stack har en modul tilgjengelig ved navn Elastic Stack Features(ESF). Denne pakken er til nytte for nødvendige sikkerhetspunkter:

#### Autentisering, autorisering og sertifikater

Autentisering bør implementeres siden Elasticsearch og Kibana sitt grensesnitt har tilgang fra nettleseren. Uten autentisering kan hvem som helst se loggene lagret i Elasticsearch. Autorisering gjøres ved bruk av roller, for å forvalte privilegier hver bruker har. I ESF kalles disse realms [103]. I løsningen er et forhåndslaget læring-sertifikat i bruk for kommunikasjon mellom Filebeat og Logstash. Dette sertifikatet bør byttes ut med egne sertifikater.

#### Sikker kommunikasjon

Logger kan inneholde private eller konfidensielle data. For å beholde integriteten ved sending av loggdata, bør kommunikasjon krypteres. EFS har funksjonalitet for dette, i form av kommunikasjon ved SSL [103]. Løsningen gjelder både for et enkelt Dockermiljø [104], og for en infrastruktur med flere noder [105].

#### Versjoner

Elastic Stack har flere fungerende versjoner. I løsningen er versjon 6.6.1 brukt, på grunn av mer praktisk læringsmateriale for bruk av Elastic Stack. På tidspunktet av gjennomføring av denne rapporten, er nyeste versjon 7.0. Så langt det er mulig bør det strebes etter å ha nyeste versjon av teknologien i bruk. Oppdagede svakheter mitigeres i nyere

versjoner i forhold til sikkerhet, og generelt forbedringer til teknologien, er gode grunner for å holde den oppdatert.

#### Intern logging

Mennesker gjør feil, og selv om brukeren er autentisert bør det også logges for hva ansatte gjør. Funksjonalitet i EFS muliggjør logging av hvem som aksesserer Elastic Stack komponenter, og hva som blir gjort [103].

### 9.3.3 Craft CMS og Nginx

For at webapplikasjonen skal være fullverdig for bruk i produksjon må det gjøres endringer i forbindelse med kryptering av trafikk. I det interne miljøet er CraftCMS satt opp ved bruk av HTTP, men best practice er å benytte HTTPS [106, 107]. Dette er en sikker versjon av samme protokoll som krypterer dataoverføringen. For Nginx finnes det ulike verktøy som setter opp støtte for HTTPS nesten helt automatisk, samtidig som det eksisterer en rekke løsninger for signering av sertifikater.

#### Reverse proxy

For å kjøre flere applikasjoner i vår infrastruktur vil manager lastbalansere mellom tilgjengelige workers, og håndtere forespørsler på vegne av worker-nodene både utad og innad i swarmen. I tillegg til å fungere som en lastbalanserer kunne manager settes opp som en reverse-proxy. Dette ville muliggjort bruk av flere applikasjoner på samme maskin.

### 9.3.4 Database

Måten databasen er implementert er ikke helt optimal for et fullverdig produksjonsmiljø. For videre utvikling bør det vurderes å sette opp egne servere for databaser for å kunne innføre redundans, bedre ytelse og bedre løsninger for backup. Skalering av containere blir fort problematisk med tanke på databaser, ettersom at alle containerene skal kunne tilby den samme informasjonen.

### 9.3.5 CI/CD

I vårt miljø på SkyHigh brukte vi en flytende-IP for å nå de virtuelle maskinene på det interne nettet. For å automatisere prosessen med oppdatering av Docker images fullstendig, måtte VMet også være tilgjengelig utenfra og ikke bare lokalt. For at et VM skal være tilgjengelig for omverdenen i SkyHigh må en global IP-addresse allokeres. Hvis dette er på plass skal det være mulig å motta webhooks fra Docker Hub for automatisk oppdatering av images ut til tjenesten.

### 9.3.6 Vurdere alternative leverandører

For at oppdragsgiver skal kunne oppnå en programmerbar infrastruktur er det essensielt å foreta en vurdering av hvilken leverandør som passer til deres behov, og som støtter opp under kravene i seksjon 3.2.

### 9.3.7 Daglig drift

Når det gjelder daglig drift av infrastrukturens bør rutiner for oppdatering av komponenter utarbeides. Disse burde baseres på best practice.

# 10 Konklusjon

## 10.1 Resultat

Oppdragsgiver ønsket en mer effektiv, oversiktlig og organisert versjon av sin eksisterende infrastruktur, med utvidet funksjonalitet og som i større grad baserer seg på automatisering.

De spesifikke ønskene til oppdragsgiver (se seksjon 1.3) har blitt implementert og oppnådd.

Det har blitt foretatt en vurdering for valg av ulike teknologier som kunne være passende i forhold til oppdragsgivers behov, samtidig som de utvalgte teknologiene har blitt brukt i implementasjonen av en infrastruktur.
Følgende funksjonalitet og systemer er implementert i infrastrukturen:

- CraftCMS med sine avhengigheter
- Redis som cache
- MySQL-database
- Loggaggregering, søkbarhet for logger og monitorering ved bruk av Elastic Stack
- Orkestrering av containere er implementert ved bruk av Docker Swarm
- CI/CD pipeline er implementert ved bruk av Bitbucket og Docker Hub
- Et alternativ til provisjonering av servere og konfigurasjon er blitt implementert ved bruk av Terraform og Ansible

Sammen med denne rapporten vil infrastrukturen som er blitt implementert fungere som et utgangspunkt for Ellera. Dette vil være et steg videre i overgangen til en containerbasert infrastruktur, som baseres på aspekter fra programmerbar infrastruktur.

## 10.2 Alternative muligheter og valg underveis

Andre planer for hvordan oppgaven skulle gjennomføres:

- Utvikle flere infrastrukturer ved bruk av ulike konsepter, og sammenligne disse
- Utvikle en infrastruktur ved bruk av Kubernetes

En av våre originale planer var å vise til forskjellige typer løsninger. Disse løsningene skulle belyse fordeler ved automatisering og bruk av forskjellige teknologier. Mer spesifikt skulle en løsning orkestreres og driftes uten bruk av CMS og orkestreringsverktøy, og hovedsakelig bestå av skript. Den andre løsningen skulle være så automatisert som mulig, og i stedet benytte verktøy til å abstraktere manuell konfigurasjon, med et høy-nivå språk. Deretter ville det blitt foretatt en sammenligning av disse, for å vurdere hva som ville passet best for Ellera sitt bruksområde.

Bruk av Kubernetes ble forespurt av oppdragsgiver og satt som et ønske, men ble tidlig vurdert som uaktuelt. Sammen med veileder kom vi frem til at Kubernetes ville være

for omfattende å sette opp fra bunnen av, samt å være uhensiktsmessig for oppgavens omfang.

## 10.3 Evaluering av gruppens arbeid

Gruppen har hatt et godt samarbeid gjennom hele prosessen med prosjektarbeidet. For det meste har gruppen vært samlet stort sett fire dager i uken, men også oftere når behovet var tilstede. Som verktøy for prosjektarbeid har Scrum og Trello blitt brukt. Trello ble brukt for å ha en oversikt over arbeidsoppgaver og hvem som arbeidet med hva. Dette har gitt et overblikk over mengden arbeid som har blitt utført av hver enkelt. Til å begynne med var det vanskelig å få til en realistisk estimering over hva som skulle utføres i løpet av en sprint, på grunn av definisjonsproblemer rundt oppgaven. Dette ble enklere etterhvert som definisjon av oppgaven ble tydeligere.

Ved å jobbe sammen som en gruppe har det vært lettere å holde motivasjonen oppe, og samtidig enklere å be om hjelp om oppgaver skulle by på problemer eller utfordringer. For å opprettholde kvalitet i rapporten har det vært vanlig å se over arbeidet som er utført av andre, samtidig som at det har bidratt til å få et innblikk i hva andre har gjort. I forhold til mengden arbeid som er lagt inn i rapporten er alle fornøyd med hverandres innsats, og føler at alle har gjort det de kan for at prosjektet skulle oppnå best mulig resultat.

### 10.3.1 Hva har vi lært

Arbeidet med bacheloroppgaven har bidratt med både oppturer og nedturer. Gjennom prosjektet har vi økt kompetansenivået innenfor flere felt med tanke på drift og programmerbar infrastruktur. Vi har fått innsyn i ulike teknologier, utfordringer ved å automatisere, og omfanget rundt mer komplekse løsninger. Det har vært mye glede i å lykkes med utfordringer, men også en del skuffelse rundt alt som ikke gikk etter planen og uforutsette problemer. Ikke minst har vi fått øynene opp for hvor mye arbeid som skal til for å utvikle noe som er bunnsolid, og hvor mye tid som kreves med tanke på det vi ikke fikk gjennomført. Oppgaven har vært krevende, kunnskapsrik, skapt frustrasjon og glede, der tiden investert føles riktig.

# Bibliografi

[1] Beal, V. Devops - development and operations. (Visited May. 2019). URL: https://www.webopedia.com/TERM/D/devops_development_operations.html.

[2] OpenStack. Heat. (Visited May. 2019). URL: https://wiki.openstack.org/wiki/Heat.

[3] Wikipedia. Repository (version control). (Visited May. 2019). URL: https://en.wikipedia.org/wiki/Repository_(version_control).

[4] Wikipedia. Scrum (software development). (Visited May. 2019). URL: https://en.wikipedia.org/wiki/Scrum_(software_development).

[5] Wikipedia. Sprint (software development). (Visited May. 2019). URL: https://en.wikipedia.org/wiki/Sprint_(software_development).

[6] Wikipedia. Unified modeling language. (Visited May. 2019). URL: https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[7] Wikipedia. Use case. (Visited May. 2019). URL: https://en.wikipedia.org/wiki/Use_case.

[8] Amazon. (Visited Apr. 2019). URL: https://aws.amazon.com/compute/sla/.

[9] Morris, K. 2016. Infrastructure as Code. 1(1), Side 5, 6 – 11, 50.

[10] EnterpriseProject. It-automation. (Visited Feb. 2019). URL: https://enterprisersproject.com/it-automation.

[11] Schmidt, D. C. Model-drivenengineering. (Visited May. 2019). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1597083.

[12] Audun Huseby, Stein Ove Jernberg, A. O. Nytt konfigurasjonssystem for linux-servere. (Visited May. 2019). URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2562144/Huseby_Jernberg_Osborg[1].pdf?sequence=1&isAllowed=y#section.2.3.

[13] for Standardisation, I. O. So 10007:2017(en) quality management — guidelines for configuration management. (Visited May. 2019). URL: https://www.iso.org/obp/ui/#iso:std:iso:10007:ed-3:v1:en.

[14] Kaiser, A. Role of configuration management in devops. (Visited Feb. 2019). URL: https://www.pluralsight.com/guides/role-of-configuration-management-in-devops.

[15] Power, V. What is a continuous integration and delivery pipeline, and why is it important? (Visited Feb. 2019). URL: https://codefresh.io/continuous-integration/continuous-integration-delivery-pipeline-important/.

[16] Amazon. What is continuous delivery? (Visited Feb. 2019). URL: https://aws.amazon.com/devops/continuous-delivery/.

[17] Power, V. Hva er kontinuerlig integrasjon (ci): Raskere og bedre software development. (Visited Feb. 2019).

[18] Puppet. (Visited Mar. 2019). URL: https://puppet.com/blog/how-to-sell-configuration-management-to-your-boss.

[19] Tawara, A. An introduction to caching: How and why we do it. (Visited May. 2019). URL: https://dzone.com/articles/introducing-amp-assimilating-caching-quick-read-fo.

[20] AWS. Benefits of message queues. (Visited May. 2019). URL: https://aws.amazon.com/message-queue/benefits/.

[21] Watson, M. Message queues and you - 12 reasons to use message queuing. (Visited May. 2019). URL: https://stackify.com/message-queues-12-reasons/.

[22] Hashicorp. Providers. (Visited May. 2019). URL: https://www.terraform.io/docs/providers/.

[23] Hashicorp. Community providers. (Visited May. 2019). URL: https://www.terraform.io/docs/providers/type/community-index.html.

[24] Puppet. (Visited Mar. 2019). URL: https://puppet.com/docs/puppetserver/5.1/services_master_puppetserver.html.

[25] Opensource. What is docker? (Visited Feb. 2019). URL: https://opensource.com/resources/what-docker.

[26] Kaewkasi, C. Docker swarm - an analysis of a very-large-scale container system. (Visited May. 2019). URL: https://blog.scaleway.com/2016/docker-swarm-an-analysis-of-a-very-large-scale-container-system/.

[27] Hub, D. Build and ship any application anywhere. (Visited May. 2019). URL: https://hub.docker.com/.

[28] Opsview. Why monitoring it infrastructure is so important. (Visited May. 2019). URL: https://www.opsview.com/resources/infrastructure/blog/why-monitoring-it-infrastructure-so-important.

[29] Elastic. (Visited May. 2019). URL: https://www.elastic.co/elk-stack.

[30] Prometheus. (Visited May. 2019). URL: https://prometheus.io/docs/introduction/overview/.

[31] AWS. What is caching and how it works. (Visited May. 2019). URL: https://aws.amazon.com/caching/.

[32] AWS. What is a message queue. (Visited May. 2019). URL: https://aws.amazon.com/message-queue/.

[33] RabbitMQ. What can rabbitmq do for you? (Visited May. 2019). URL: https://www.rabbitmq.com/features.html.

[34] Redis. Redis explained in 5 minutes or less. (Visited Feb. 2019). URL: https://www.credera.com/blog/technology-insights/java/redis-explained-5-minutes-less/.

[35] Kafka, A. Introduction. (Visited May. 2019). URL: https://kafka.apache.org/intro.

[36] Jenkins. Jenkins user documentation. (Visited May. 2019). URL: https://jenkins.io/doc/.

[37] Atlassian. What is bitbucket? (Visited May. 2019). URL: https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket.

[38] Guru99. What is a database? what is sql? (Visited Feb. 2019). URL: https://www.guru99.com/introduction-to-database-sql.html.

[39] Gurjar, N. Think before using configuration management tools for infrastructure provisioning. (Visited May. 2019). URL: http://neeleshgurjar.co.in/techidnyan/think-before-using-configuration-management-tools-for-infrastructure-provisioning/.

[40] Nuñez, C. Why configuration management and provisioning are different. (Visited May. 2019). URL: https://www.thoughtworks.com/insights/blog/why-configuration-management-and-provisioning-are-different.

[41] slikk66. (Visited May. 2019). URL: https://www.reddit.com/r/aws/comments/8iltiz/which_do_you_use_terraform_or_cloudformation/.

[42] Tarry, A. (Visited May. 2019). URL: https://andrewtarry.com/thoughts_on_terraform/.

[43] Gheorghiu, G. (Visited May. 2019). URL: http://agiletesting.blogspot.com/2010/03/automated-deployment-systems-push-vs.html.

[44] Bootcamp, O. D. (Visited May. 2019). URL: https://devopsbootcamp.osuosl.org/configuration-management.html#pull-vs-push-models.

[45] Branigan, J. (Visited May. 2019). URL: https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/The-pros-and-cons-of-pull-and-push-models-for-processing-IoT-data.

[46] D. Aksoy Comput. Sci. Dept., California Univ., D. C. U. L. C. S. D. C. U. D. C. U. (Visited Apr. 2019). URL: https://ieeexplore.ieee.org/abstract/document/1378225.

[47] Michael DeHaan, James Cammarata, J. K. (Visited Apr. 2019). URL: http://cdn2.hubspot.net/hub/330046/file-476015569-pdf/pdf_content/Scaling_and_Performance_of_the_Ansible_Management_Toolchain.pdf.

[48] Puppet. (Visited May. 2019). URL: https://puppet.com/products/bolt.

[49] Puppet. Docker. (Visited May. 2019). URL: https://forge.puppet.com/puppetlabs/docker.

[50] Ravindra, S. Kubernetes vs docker swarm whats the difference. (Visited May. 2019). URL: https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/.

[51] India, S. Kubernetes vs. docker. (Visited May. 2019). URL: https://hackernoon.com/kubernetes-vs-docker-swarm-a-complete-comparison-guide-15ba3ac6f750.

[52] Kidd, C. (Visited May. 2019). URL: https://www.bmc.com/blogs/monitoring-logging-tracing/.

[53] Research, W. & Development. (Visited May. 2019). URL: https://winderresearch.com/logging-vs-tracing-vs-monitoring/.

[54] Prometheus. (Visited May. 2019). URL: https://prometheus.io/docs/prometheus/latest/installation/.

[55] Research, W. & Development. (Visited May. 2019). URL: https://www.elastic.co/guide/en/elastic-stack-get-started/master/get-started-docker.html.

[56] Prometheus. Prometheus. (Visited May. 2019). URL: https://github.com/prometheus/prometheus.

[57] Stackshare. Prometheus. (Visited May. 2019). URL: https://stackshare.io/prometheus.

[58] Stackshare. Prometheus. (Visited May. 2019). URL: https://github.com/elastic/elasticsearch.

[59] Elastic. Kibana. (Visited May. 2019). URL: https://github.com/elastic/kibana.

[60] Elastic. Logstash. (Visited May. 2019). URL: https://github.com/elastic/logstash.

[61] Stackshare. Elasticsearch vs kibana vs logstash. (Visited May. 2019). URL: https://stackshare.io/stackups/elasticsearch-vs-kibana-vs-logstash.

[62] Prometheus. (Visited May. 2019). URL: https://prometheus.io/docs/introduction/faq/.

[63] Newnman, A. How to prevent log data loss when using elasticsearch in production. (Visited May. 2019). URL: https://logdna.com/blog/how-to-prevent-log-data-loss-when-using-elastic-search-in-a-production-environment/.

[64] Francini, G. Usefulness of an mq layer in an elk stack. (Visited May. 2019). URL: https://logdna.com/blog/how-to-prevent-log-data-loss-when-using-elastic-search-in-a-production-environment/.

[65] Mayr, S. Why we chose kafka for the trello socket architecture. (Visited May. 2019). URL: https://tech.trello.com/why-we-chose-kafka/.

[66] Stackshare. Redis vs rabbitmq vs kafka. (Visited May. 2019). URL: https://stackshare.io/stackups/kafka-vs-rabbitmq-vs-redis.

[67] Stack, E. Redis plugin. (Visited May. 2019). URL: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-redis.html.

[68] Stack, E. Rabbitmq plugin. (Visited May. 2019). URL: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-rabbitmq.html.

[69] JReport. 3-tier architecture. (Visited May. 2019). URL: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/.

[70] Clemenko, A. Building a docker secure supply chain. (Visited May. 2019). URL: https://success.docker.com/article/secure-supply-chain.

[71] Basques, K. (Visited Mar. 2019). URL: https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https.

[72] Fuglseth, B. (Visited Mar. 2019). URL: https://www.makeuseof.com/tag/what-is-hsts/.

[73] Supalov, V. (Visited Mar. 2019). URL: https://vsupalov.com/database-in-docker/.

[74] Kubernetes. (Visited Mar. 2019). URL: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/.

[75] Search, E. (Visited Mar. 2019). URL: https://www.elastic.co/products/beats/filebeat.

[76] Search, E. (Visited Mar. 2019). URL: https://www.elastic.co/products/logstash.

[77] Terraform. Core workflow. (Visited May. 2019). URL: https://www.terraform.io/guides/core-workflow.html.

[78] Terraform. Backend. (Visited May. 2019). URL: https://www.terraform.io/docs/backends/.

[79] Terraform. Resources. (Visited May. 2019). URL: https://www.terraform.io/docs/configuration/resources.html.

[80] Adrian Lund, V. M. terraform-consul. (Visited May. 2019). URL: https://github.com/adrialu/terraform-consul/blob/production/paper/rendered.pdf.

[81] HashiCorp. (Visited May. 2019). URL: https://learn.hashicorp.com/terraform/development/running-terraform-in-automation.

[82] Ansible. Overview how ansible works. (Visited May. 2019). URL: https://www.ansible.com/overview/how-ansible-works.

[83] Ansible. Best practices. (Visited May. 2019). URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html.

[84] Ansible. docker_swarm – manage swarm cluster. (Visited May. 2019). URL: https://docs.ansible.com/ansible/latest/modules/docker_swarm_module.html.

[85] (adammck), A. M. (Visited Mar. 2019). URL: https://github.com/adammck/terraform-inventory.

[86] Adrian Lund, V. M. hacking-portal. (Visited May. 2019). URL: https://github.com/vetletm/hacking-portal/blob/master/build/inventory.

[87] Ansible. (Visited Mar. 2019). URL: https://github.com/ansible/ansible/blob/devel/contrib/inventory/.

[88] Elaine Barker, Q. D. (Visited May. 2019). URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf.

[89] OpenSSH. SSH security. Waltham, MA, U. S. C. S. I. (Visited May. 2019). URL: https://man.openbsd.org/ssh-keygen.

[90] CMS, C. (Visited Mar. 2019). URL: https://docs.craftcms.com/v3/installation.html.

[91] Wyveo. (Visited Apr. 2019). URL: https://hub.docker.com/u/wyveo.

[92] debian. (Visited Apr. 2019). URL: https://wiki.debian.org/DebianIntroduction.

[93] Chavali, P. (Visited Apr. 2019). URL: https://www.ca.com/en/blog-developers/docker-containers-os-base-image.html.

[94] debian. (Visited Apr. 2019). URL: https://www.debian.org/intro/why_debian.

[95] Williams, G. (Visited Apr. 2019). URL: https://www.togaware.com/linux/survivor/Advantages_Debian.html.

[96] Ocean, D. (Visited Apr. 2019). URL: https://www.digitalocean.com/community/tutorials/understanding-and-implementing-fastcgi-proxying-in-nginx.

[97] Rohprimardo. (Visited Apr. 2019). URL: https://www.delaat.net/rp/2014-2015/p92/report.pdf.

[98] Pujadas, S. (Visited Mar. 2019). URL: https://github.com/spujadas/elk-docker.

[99] Theaxiom. (Visited Apr. 2019). URL: https://github.com/phusion/baseimage-docker.

[100] Elastic. (Visited Apr. 2019). URL: https://www.elastic.co/guide/en/elastic-stack-overview/current/elasticsearch-security.html.

[101] Ansible. Red hat ansible tower. (Visited May. 2019). URL: https://www.ansible.com/products/tower.

[102] Ansible. The awx project. (Visited May. 2019). URL: https://www.ansible.com/products/awx-project/faq.

[103] Elastic. (Visited Apr. 2019). URL: https://www.elastic.co/guide/en/elastic-stack-overview/current/elasticsearch-security.html.

[104] Elastic. (Visited Apr. 2019). URL: https://www.elastic.co/guide/en/elasticsearch/reference/7.0/configuring-tls-docker.html.

[105] Elastic. (Visited Apr. 2019). URL: https://www.elastic.co/guide/en/elasticsearch/reference/7.0/configuring-tls.html#node-certificates.

[106] Gite, V. Top 25 nginx web server best security practices. (Visited May. 2019). URL: https://www.cyberciti.biz/tips/linux-unix-bsd-nginx-webserver-security.html.

[107] Hernandez, M. Http vs. https for seo: What you need to know to stay in google's good graces. (Visited May. 2019). URL: https://ahrefs.com/blog/http-vs-https-for-seo/.

[108] Bravo, M. Top 5 configuration management tools. (Visited Mar. 2019). URL: https://opensource.com/article/18/12/configuration-management-tools.

[109] SoftwaretestingHelp. 10 best software configuration management tools (scm tools in 2019). (Visited Mar. 2019). URL: https://www.softwaretestinghelp.com/top-5-software-configuration-management-tools/.

[110] UpGuard. 7 configuration management (cm) tools you need to know about. (Visited Mar. 2019). URL: https://www.upguard.com/articles/the-7-configuration-management-tools-you-need-to-know.

[111] (Visited Mar. 2019). URL: https://github.com/ansible/ansible.

[112] (Visited Mar. 2019). URL: https://github.com/cfengine/core.

[113] (Visited Mar. 2019). URL: https://github.com/chef/chef.

[114] (Visited Mar. 2019). URL: https://github.com/puppetlabs/puppet.

[115] (Visited Mar. 2019). URL: https://github.com/saltstack/salt.

[116] Hat, R. (Visited Mar. 2019). URL: https://www.redhat.com/en/technologies/all-products.

[117] Gjøvik, N. (Visited Mar. 2019). URL: https://www.ntnu.no/studier/emner/IMT3005#tab=omEmnet.

# A Vurderingsgrunnlag for kandidater

## A.1 Innledning

Viktige punkter for valg og vurdering bør være utifra en kravspesifikasjon og attributter for kandidatene som samsvarer med infrastrukturen de skal brukes i. Ingen kravspesifikasjon foreligger for prosjektet, som vil si at valgene kan tas utelukkende fra et synspunkt for nøyaktig det infrastrukturen passer best til. På grunn av begrenset tid, vil også gruppens tidligere kjennskap til teknologier spille inn på valgene som blir tatt.

Kriteriene for valg av kandidater vil være popularitet og attributter.

- Popularitet, begrunnes som et kriterie med at dersom teknologien er populær, vil den sannsynligvis fortsette å utbedres. Spesielt relevant er dette for open-source prosjekter der alle brukerne kan komme med forbedringer, og skaperne vil få stadig tilbakemelding på produktet. For å kunne fastlå en verdi vil resultater fra sider som Google Trends og GitHub stars brukes for å måle populariteten til produktet.
- Attributter
    - Brukervennlighet
    - Open source eller ikke
    - Begrensninger på abonnement, eller ikke (Community vs Enterprise)
    - Fokusområde (feks skalerbarhet o.l)
    - Tilgjengelighet og Kompleksitet
    - Støtte for flere skyer? Vendor uavhengig? Vendor Lock-In
    - Sikkerhet og Autentisering (Lagring, Nøkler, Tilgang osv)
    - Dokumentasjon og tilgjengelige Ressurser
    - Systemets Utvikling: Blir det jobbet med, er det under utvikling, er det fortsatt støtte for gamle OS, vil det fortsatt utvikles nye oppdateringer osv.

## A.2 Kravspesifikasjon

### A.2.1 Krav til sikkerhet

Sikkerhetsaspektene ved de aktuelle teknologiene er veldig viktig da disse vil være en del av en infrastruktur som vil kunne behandle sensitiv data. Enhver sårbarhet vil medføre risiko i ulik grad som kan ha stor innvirkning på systemet. Aspekter for hvordan data behandles i form av overføring og lagring vil bli vurdert, og hva slags funksjonalitet som er implementert for å håndtere dette. Hva slags porter benyttes av tjenesten og om dataoverføringen over disse portene benytter noen form for kryptering. Vil konfigurasjonsdata ligge i klartekst, finnes det mekanismer for å beskytte dette.

### A.2.2 Krav til dokumentasjon

Dokumentasjon bør være lett tilgjengelig fra produsenten sin side, denne skal omhandle funksjonalitet og bruk av systemet. Dokumentasjonen bør inneholde versjonsnummer

slik at det lett kan skilles mellom funksjonalitet for de ulike versjonene av systemet.

### A.2.3 Krav til opplæring

Selv om dokumentasjonen er god, er det ikke alltid nok for at aspirerende brukere skal bli kjent med en teknologi. Det bør være muligheter for annen læring enn bare dokumentasjonen, som for eksempel læringsvideoer, kursing eller fungerende maler på enkel bruk av teknologien.

## A.3 Vurderingskriterier

Vurderingskriteriene vil fungere som et rammeverk og mal for vurderingsprosessen av de ulike kandidatene. De kandidatene som oppfyller kriteriene vil ta del i prosessen for å finne de mest ideelle teknologiene for sluttproduktet av infrastrukturen.

### A.3.1 Organisasjon

Organisasjonens rykte og historie kan påvirke vurderingen av teknologien. Undersøkelse av organisasjonen kan gi svar på spørsmål som:

- Har organisasjonen utviklet gode produkter tidligere
- Finnes det eksempler på aktører som bruker teknologien

### A.3.2 Dokumentasjon

Dersom dokumentasjonen ikke er komplett, kan det føre til frustrasjon for brukeren. Dårlig dokumentasjon kan føre til liten forståelse av systemet, og får heller ikke se nytten teknologien kan ha. Vurdering av dokumentasjon kan gi svar på følgende spørsmål:

- Er dokumentasjon lett tilgjengelig
- Til hvilken grad er systemet dokumentert

### A.3.3 Teknologiens utvikling

For en infrastruktur som skal brukes over lengre tid, bør også de teknologiske komponentene i den kunne brukes i lengre tid. Dette gjør videre drift enklere siden komponenter ikke må byttes ut. I tillegg kan det føre til økt sikkerhet for driften, siden eventuelle feil og svakheter i komponenten vil rettes via tilgjengelige oppdateringer. Vurderingen av teknologiens utvikling vil gi svar på følgende spørsmål.

- Tid mellom releases
- Langtids støtte for versjoner

### A.3.4 Opplæring og hjelpefunksjoner

Opplæringsmateriell hjelper brukerens opplevelse av brukervennlighet, og med gode opplevelser følger flere trofaste brukere. Ved å vurdere alt fra bøker, forum og andre hjelpefunksjoner kan vurdering av opplæring og hjelpefunksjoner gi svar på følgende spørsmål:

- Tilgjengeligheten av opplæringsmateriell som opplæringsvideoer
- Eksempler på bruk av funksjonalitet

## A.4 Konfigurasjonssystem

Det finnes mange forskjellige typer konfigurasjonssystem, men for en ung bedrift uten tidligere erfaring med slike verktøy, vil ikke de små fordelene eller ulempene ha noe spesielt fokus. Det viktigste er hvor tilgjengelig kompetansen er for verktøyet. Dette kan delvis måles ut ifra populariteten til verktøyet. Dersom verktøyet er mye brukt, vil det også naturlig være flere som har kompetanse for verktøyet. Ved å bruke Google-trends, kan det illustreres en oversikt over antall søk og etterspørsel etter verktøyet. Det er blitt tatt utgangspunkt i flere lister og rangeringer[108, 109, 110], som viser til de kandidatene som er mest brukt og omtalt. Begrunnet med prosjektets varighet skal det velges 2 kandidater for grundigere vurdering.
Tall fra Google-trends er hentet fra følgende kilde.

| Kandidat | Siste versjon | Dato | Google Trends score | GitHub stars |
|---|---|---|---|---|
| Ansible | 2.7.8 | Feb 2019 | 78 | 35742 [111] |
| CFEngine | Enterprise 3.12.1 | 30. Nov 2018 | 2 | 317 [112] |
| Chef | 0.2.52 | 1. Mars 2019 | 6 | 5677 [113] |
| Puppet | 6.0.5 | 15. Jan 2019 | 16 | 5211 [114] |
| Saltstack | Salt 2019.2.0 Fluorine | 25. Feb 2019 | 5 | 9712 [115] |

Tabell 8: Configuration

Tabell 2 viser at Ansible er en klar vinner på popularitet, både på Google-søk og GitHub stars. Chef, Puppet og Saltstack er på omtrent samme nivå, og CFEngine er det minst omtalte konfigurasjonssystemet.

Neste steg er en vurdering i forhold til krav og kriterier, som beskrevet i kravspesifikasjon og vurderingskriterier.

| | Ansible | CFEngine | Chef | Puppet | Saltstack |
|---|---|---|---|---|---|
| Sikker kommunikasjon | SSH | HTTPS/TLS | Kryptert | HTTPS/TLS | Kryptert |
| Autentisering | Nøkler | Nøkler | Sertifikater | Nøkler | Nøkler |
| Dokumentasjon | Ja | Ja | Ja | Ja | Ja |
| Opplæringsmateriell | Ja | Ja | Ja | Ja | Ja |
| Tidligere gode produkter | Ja(Red Hat) | Nei | Nei | Nei | Nei |
| Lett tilgjengelig dokumentasjon | Ja | Ja | Ja | Ja | Ja |
| Release siste år | Ja | Ja | Ja | Ja | Ja |
| Betalt løsning | Ja | Ja | Ja | Ja | Ja |

Tabell 9: Vurdering av konfigurasjonsstyring

Resultatet fra tabell 9, viser at utvalget stiller relativt likt i forhold til tilgjengelige midler og tekniske spesifikasjoner. Med kriteriene for attributter nesten oppfylt for samtlige kandidater, er alle passende kandidater, men prosjektets varighet er ikke lang nok til å ta i bruk alle. Ansible skiller seg ut med å ha tidligere gode produkter på markedet.

Red Hat som nå er eier av Ansible, har utviklet mange produkter som Satellite, Jboss, Fuse, etc.[116]. Med et større utviklingsmiljø og høyere popularitet enn andre konfigurasjonssystemer, velges Ansible som et passende valg.

Uten noen store forskjeller fra de resterende kandidatene, bortsett fra CFEngines popularitet, velges konfigurasjonssystem nr.2 utifra praktiske grunner. Medlemmene i gruppen har tidligere erfaring med Puppet fra faget Programmerbar Infrastruktur[117] i Gjøvik. Tidligere erfaring vil fremskynde en tidkrevende prosess med læring av et helt nytt konfigurasjonssystem. Derfor blir kandidat nr. 2 Puppet.

# B Terraform

## B.1 terraform/main.tf

Listing B.1: main.tf

```
1  #https://github.com/hashicorp/terraform/issues/2844
2  resource "openstack_compute_floatingip_associate_v2" "
       docker" {
3    count       = "${var.server_count}"
4    floating_ip = "${element(
       openstack_networking_floatingip_v2.float1.*.
       address, count.index)}"
5    instance_id = "${element(
       openstack_compute_instance_v2.node.*.id, count.
       index)}"
6  }
7
8  resource "openstack_compute_floatingip_associate_v2" "
       manager" {
9    instance_id = "${openstack_compute_instance_v2.
       manager.id}"
10   floating_ip = "${openstack_networking_floatingip_v2.
       float1man.address}"
11 }
12
13 data "tls_public_key" "example" {
14   private_key_pem = "${file("~/.ssh/test")}"
15 }
16
17 data "template_file" "script" {
18   template = "${file("~/templates/add_keys.sh.tpl")}"
19
20   vars = {
21     pvt_key = "${data.tls_public_key.example.
         private_key_pem}"
22     pub_key = "${data.tls_public_key.example.
         public_key_openssh}"
23   }
24 }
25
26 data "template_cloudinit_config" "config" {
27   gzip          = true
28   base64_encode = true
29
30   part {
31     filename     = "add_keys.sh"
32     content_type = "text/x-shellscript"
33     content      = "${data.template_file.script.
         rendered}"
```

```
34    }
35  }
36
37  resource "openstack_compute_keypair_v2" "kp" {
38    name       = "manager_kp"
39    public_key = "${data.tls_public_key.example.
          public_key_openssh}"
40  }
```

## B.2 terraform/node.tf

Listing B.2: node.tf

```
1   resource "openstack_compute_instance_v2" "manager" {
2     name            = "Manager"
3     image_name      = "${var.image}"
4     flavor_name     = "${var.flavor}"
5     key_pair        = "${var.kp}"
6     security_groups = [
7       "${openstack_compute_secgroup_v2.dockers.name}",
8       "${openstack_compute_secgroup_v2.ssh.name}",
9       "${openstack_compute_secgroup_v2.http.name}",
10    ]
11
12    network = {
13      uuid = "${openstack_networking_network_v2.
            network_11.id}"
14    }
15
16    user_data = "${data.template_cloudinit_config.config.
          rendered}"
17  }
18
19  resource "openstack_compute_instance_v2" "node" {
20    count           = "${var.server_count}"
21    name            = "Node${count.index}"
22    image_name      = "${var.image}"
23    flavor_name     = "${var.flavor}"
24    key_pair        = "${var.kp}"
25    security_groups = [
26      "${openstack_compute_secgroup_v2.dockers.name}",
27      "${openstack_compute_secgroup_v2.ssh.name}",
28      "${openstack_compute_secgroup_v2.http.name}",
29    ]
30
31    network = {
32      uuid = "${openstack_networking_network_v2.
            network_11.id}"
33    }
34  }
```

## B.3 terraform/secgroup.tf

Listing B.3: secgroup.tf

```
resource "openstack_compute_secgroup_v2" "ssh" {
  name        = "ssh"
  description = "SSH access"

  rule {
    from_port   = 22
    to_port     = 22
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }
}

resource "openstack_compute_secgroup_v2" "http" {
  name        = "http"
  description = "HTTP access"

  rule {
    from_port   = 443
    to_port     = 443
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }

  rule {
    from_port   = 80
    to_port     = 80
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }
}

resource "openstack_compute_secgroup_v2" "dockers" {
  name        = "dockers"
  description = "Swarm access"

  #https://www.digitalocean.com/community/tutorials/how-to-configure-the-linux-firewall-for-docker-swarm-on-ubuntu-16-04
  rule {
    from_port   = 2376
    to_port     = 2376
    ip_protocol = "tcp"
    cidr        = "${var.docker_net}"
  }

  rule {
    from_port   = 2377
    to_port     = 2377
    ip_protocol = "tcp"
    cidr        = "${var.docker_net}"
  }

```

```
51    rule {
52      from_port   = 7946
53      to_port     = 7946
54      ip_protocol = "tcp"
55      cidr        = "${var.docker_net}"
56    }
57
58    rule {
59      from_port   = 4789
60      to_port     = 4789
61      ip_protocol = "tcp"
62      cidr        = "${var.docker_net}"
63    }
64  }
```

## B.4 terraform/network.tf

Listing B.4: network.tf

```
1  resource "openstack_networking_network_v2" "network_11"
       {
2    name           = "docker-net"
3    admin_state_up = "true"
4  }
5
6  resource "openstack_networking_subnet_v2" "subnet_11" {
7    name        = "subnet_11"
8    network_id  = "${openstack_networking_network_v2.
       network_11.id}"
9    cidr        = "${var.docker_net}"
10   ip_version  = 4
11   enable_dhcp = "true"
12 }
13
14 resource "openstack_networking_router_v2" "router" {
15   name                = "router"
16   admin_state_up      = "true"
17   external_network_id = "730cb16e-a460-4a87-8c73-50
       a2cb2293f9" #"6a4d7272-0b9e-4a94-9e39-758f3941f091
       " #"730cb16e-a460-4a87-8c73-50a2cb2293f9"
18 }
19
20 resource "openstack_networking_router_interface_v2" "
      routerint" {
21   router_id = "${openstack_networking_router_v2.router.
       id}"
22   subnet_id = "${openstack_networking_subnet_v2.
       subnet_11.id}"
23 }
24
25 resource "openstack_networking_floatingip_v2" "float1"
      {
26   count      = "${var.server_count}"
27   pool       = "${var.floating_pool}"
```

```
28    depends_on = ["
          openstack_networking_router_interface_v2.routerint
          "]
29  }
30
31  resource "openstack_networking_floatingip_v2" "
        float1man" {
32    pool       = "${var.floating_pool}"
33    depends_on = ["
          openstack_networking_router_interface_v2.routerint
          "]
34  }
```

## B.5 terraform/cred.tf

Listing B.5: cred.tf

```
1  provider "openstack" {
2    user_domain_name = "${var.user_domain}"
3    tenant_id        = "${var.tenant_id}"
4    tenant_name      = "${var.tenant_name}"
5    user_name        = "${var.username}"
6    password         = "${var.password}"
7    auth_url         = "${var.auth_url}"
8    region           = "${var.region}"
9  }
```

# C Ansible

## C.1 playbook.yml

Listing C.1: Ansible playbook

```
---
- hosts: all
  roles:
    - base
    - docker

- hosts: manager
  tasks:
    - name: Get current Swarm state
      docker_swarm:
        state: inspect
      register: swarm_info
      become: yes

    - name: Initialize a swarm
      when: swarm_info.swarm_facts is not defined
      docker_swarm:
        state: present
        advertise_addr: '{{ ansible_default_ipv4.
            address }}'
      become: yes

    - name: Get latest Swarm state
    # https://github.com/ansible/ansible/issues/15710#
       issuecomment-216645922
      docker_swarm:
        state: inspect
      register: swarm_info
      become: yes

    - name: Get join token
      set_fact:
        join_token: '{{ swarm_info.swarm_facts.
            JoinTokens.Manager }}'
      become: yes

- hosts: docker:!manager
  tasks:
    - name: Get current swarm state
      docker_swarm:
        state: inspect
      register: swarm_info
      become: yes
```

```
41
42      - name: Join swarm
43        when: swarm_info.swarm_facts is not defined
44        docker_swarm:
45          state: join
46          advertise_addr: "{{ ansible_default_ipv4.
                address }}"
47          remote_addrs:
48            - "{{ hostvars[groups['manager'][0]].
                  ansible_default_ipv4.address }}:2377"
49          join_token: "{{ hostvars[groups['manager'][0]].
                join_token }}"
50        become: yes
51
52      - name: Get current swarm state
53        docker_swarm:
54          state: inspect
55        register: swarm_info
56        become: yes
57
58      - name: Join swarm
59        when: swarm_info.swarm_facts is not defined
60        docker_swarm:
61          state: join
62          advertise_addr: "{{ ansible_default_ipv4.
                address }}"
63          remote_addrs:
64            - "{{ hostvars[groups['manager'][0]].
                  ansible_default_ipv4.address }}:2377"
65          join_token: "{{ hostvars[groups['manager'][0]].
                join_token }}"
66      - debug: msg="value is {{ hostvars[groups['manager
            '][0]].ansible_default_ipv4.address }}:2377"
67        become: yes
68
69  - hosts: manager
70    roles:
71      - stack
```

## C.2 docker/handlers/main.yml

Listing C.2: Docker restart handler

```
1  - name: restart_docker
2    service:
3      name: docker
4      state: restarted
5    become: yes
```

## C.3 docker/tasks/main.yml

Listing C.3: Docker installasjon

```
  - name: Install dependencies
    apt:
      name: '{{ packages }}'
      state: latest
      update_cache: yes
    vars:
      packages:
        - apt-transport-https
        - ca-certificates
        - curl
        - software-properties-common
        - python-pip
        - python3
        - python3-pip
        - python3-setuptools
    become: yes


- name: Install Python dependencies
  pip:
    name: '{{ packages }}'
  vars:
    packages:
      - docker
      - jsondiff
  become: yes


- name: Add Docker GPG key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
  become: yes

- name: Add Docker APT repository
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/
        linux/ubuntu {{ansible_distribution_release}}
        stable
    state: present
  become: yes

- name: Install Docker
  apt:
    name: docker-ce
    state: latest
    update_cache: yes
  become: yes
  notify:
    - restart_docker

- name: Enable service
  service:
    name: docker
```

```
52      state: started
53      enabled: yes
54    become: yes
```

## C.4 stack/tasks/main.yml

Listing C.4: Stack deploy

```
1   - name: Deploy
2   become: yes
3   docker_stack:
4     state: present
5     name: test
6     prune: yes
7     compose:
8       - version: '3'
9
10        volumes:
11          craftcms-data:
12          craftcms-logs:
13          craft3:
14          data:
15
16        services:
17          elk:
18            image: daniebru/elastic:latest
19            ports:
20              - 5601:5601
21              - 9200:9200
22              - 5044:5044
23            environment:
24             #WEB is linked
25               WEB_HOST: web
26              #DB is linked
27                DB_HOST: db
28              #REDIS is linked
29                REDIS_HOST: redis
30              deploy:
31                replicas: 1
32                restart_policy:
33                  condition: on-failure
34          web:
35            image: birgerm/docker-craftcms:latest
36            ports:
37              - 80:80
38            volumes:
39              - craftcms-logs:/var/log
40              - craftcms-data:/usr/share/nginx
41            deploy:
42              replicas: 1
43              restart_policy:
44                condition: on-failure
45
46            # env vars are replaced in .env
```

```
47              environment:
48                # Set locale to UTF-8 (https://oncletom.
                     io/2015/docker-encoding/)
49                LANG: C.UTF-8
50                # REDIS is linked
51                REDIS_HOST: redis
52                # DB is linked
53                DB_DRIVER: mysql
54                DB_SERVER: db
55                DB_DATABASE: craft3
56                DB_PASSWORD: "{{ vault_db_password }}"
57                DB_SCHEMA: public
58                DB_PORT: '3306'
59                DB_USER: "{{ vault_db_user }}"
60
61            db:
62              image: mysql:5.7
63              deploy:
64                replicas: 1
65                restart_policy:
66                  condition: on-failure
67              environment:
68                MYSQL_DATABASE: 'craft3'
69              # So you don't have to use root, but you
                   can if you like
70                MYSQL_USER: "{{ vault_db_user }}"
71              # You can use whatever password you like
72                MYSQL_PASSWORD: "{{ vault_db_password }}"
73              # Password for root access
74                MYSQL_ROOT_PASSWORD: "{{
                     vault_root_password }}"
75              # <Port exposed> : < MySQL Port running
                   inside container>
76                ports: 3306:3306
77              # Where our data will be persisted
78              volumes:
79                - craft3:/var/lib/mysql
80
81            redis:
82              image: redis:5.0.3-alpine
83              volumes:
84                - data:/data
85              deploy:
86                replicas: 1
87                restart_policy:
88                  condition: on-failure
```

# D inventory.py

Listing D.1: Ansible dynamic inventory

```
#!/usr/bin/env python3

#
# the inventory from Terraform Inventory does not give the hosts their floating IPs,
# so we wrap the script and create our own inventory
#

import re
import json
import subprocess

# get json inventory from local terraform state file
# (by default: .terraform/terraform.tfstate)
p = subprocess.Popen(
        ['~/terraform-inventory', '--list'],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
out, err = p.communicate()

# get a inventory json list from the output
inventory = json.loads(out.strip().decode())

# prepare a new inventory
new_inventory = {
        'all': {'hosts': []},
        'docker': [],
        'manager': [],
}

# iterate through the IPs and add them to the inventory
for host in ('docker', 'manager'):
  i = 1
  for ip in inventory[host]:
    if ip.startswith('10.212.'):
          #print(ip)
          new_inventory[host].append(ip)
          new_inventory['all']['hosts'].append(ip)
          new_inventory['{}{}'.format(host, i)] = [ip]
          i += 1

print(json.dumps(new_inventory, indent=4))
```

# E docker_stack.py

Listing E.1: Ansible Docker Stack module

```
#Copyright (c) 2018 Dario Zanzico (git@dariozanzico.com)
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.txt)

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type


ANSIBLE_METADATA = {'status': ['preview'],
                    'supported_by': 'community',
                    'metadata_version': '1.1'}

DOCUMENTATION = '''
---
module: docker_stack
author: "Dario Zanzico (@dariko)"
short_description: docker stack module
description:
-   Manage docker stacks using the 'docker stack' command
    on the target node
    (see examples)
version_added: "2.8"
options:
    name:
        required: true
        description:
        -   Stack name
    state:
        description:
        -   Service state.
        default: "present"
        choices:
        -   present
        -   absent
    compose:
        required: true
        default: []
        description:
        -   List of compose definitions. Any element may be a string
            referring to the path of the compose file on the target host
```

```
40              or the YAML contents of a compose file
                    nested as dictionary.
41      prune:
42          required: false
43          default: false
44          description:
45          -   If true will add the C(--prune) option to
                the C(docker stack deploy) command.
46              This will have docker remove the services
                    not present in the
47              current stack definition.
48          type: bool
49      with_registry_auth:
50          required: false
51          default: false
52          description:
53          -   If true will add the C(--with-registry-auth
                ) option to the C(docker stack deploy)
                command.
54              This will have docker send registry
                    authentication details to Swarm agents.
55          type: bool
56      resolve_image:
57          required: false
58          choices: ["always", "changed", "never"]
59          description:
60          -   If set will add the C(--resolve-image)
                option to the C(docker stack deploy) command
                .
61              This will have docker query the registry to
                     resolve image digest and
62              supported platforms. If not set, docker use
                     "always" by default.
63      absent_retries:
64          required: false
65          default: 0
66          description:
67          -   If C(>0) and C(state==absent) the module
                will retry up to
68              C(absent_retries) times to delete the stack
                     until all the
69              resources have been effectively deleted.
70              If the last try still reports the stack as
                    not completely
71              removed the module will fail.
72      absent_retries_interval:
73          required: false
74          default: 1
75          description:
76          -   Interval in seconds between C(
                absent_retries)
77
78  requirements:
```

```
79  -   jsondiff
80  -   pyyaml
81  '''
82
83  RETURN = '''
84  docker_stack_spec_diff:
85      description: |
86          dictionary containing the differences between
               the 'Spec' field
87          of the stack services before and after applying
                the new stack
88          definition.
89      sample: >
90          "docker_stack_specs_diff":
91          {'test_stack_test_service': {u'TaskTemplate': {
               u'ContainerSpec': {delete: [u'Env']}}}}
92      returned: on change
93      type: dict
94  '''
95
96  EXAMPLES = '''
97  -   name: deploy 'stack1' stack from file
98      docker_stack:
99          state: present
100         name: stack1
101         compose:
102         -   /opt/stack.compose
103
104 -   name: deploy 'stack2' from base file and yaml
       overrides
105     docker_stack:
106         state: present
107         name: stack2
108         compose:
109         -   /opt/stack.compose
110         -   version: '3'
111             services:
112                 web:
113                     image: nginx:latest
114                     environment:
115                         ENVVAR: envvar
116
117 -   name: deprovision 'stack1'
118     docker_stack:
119         state: absent
120 '''
121
122
123 import json
124 import tempfile
125 from ansible.module_utils.six import string_types
126 from time import sleep
127
```

```
128 try:
129     from jsondiff import diff as json_diff
130     HAS_JSONDIFF = True
131 except ImportError:
132     HAS_JSONDIFF = False
133
134 try:
135     from yaml import dump as yaml_dump
136     HAS_YAML = True
137 except ImportError:
138     HAS_YAML = False
139
140 from ansible.module_utils.basic import AnsibleModule,
        os
141
142
143 def docker_stack_services(module, stack_name):
144     docker_bin = module.get_bin_path('docker', required
            =True)
145     rc, out, err = module.run_command([docker_bin,
146                                        "stack",
147                                        "services",
148                                        stack_name,
149                                        "--format",
150                                        "{{.Name}}"])
151     if err == "Nothing found in stack: %s\n" %
            stack_name:
152         return []
153     return out.strip().split('\n')
154
155
156 def docker_service_inspect(module, service_name):
157     docker_bin = module.get_bin_path('docker', required
            =True)
158     rc, out, err = module.run_command([docker_bin,
159                                        "service",
160                                        "inspect",
161                                        service_name])
162     if rc != 0:
163         return None
164     else:
165         ret = json.loads(out)[0]['Spec']
166         return ret
167
168
169 def docker_stack_deploy(module, stack_name,
        compose_files):
170     docker_bin = module.get_bin_path('docker', required
            =True)
171     command = [docker_bin, "stack", "deploy"]
172     if module.params["prune"]:
173         command += ["--prune"]
174     if module.params["with_registry_auth"]:
```

```
175             command += ["--with-registry-auth"]
176         if module.params["resolve_image"]:
177             command += ["--resolve-image",
178                         module.params["resolve_image"]]
179         for compose_file in compose_files:
180             command += ["--compose-file",
181                         compose_file]
182         command += [stack_name]
183         return module.run_command(command)
184
185
186     def docker_stack_inspect(module, stack_name):
187         ret = {}
188         for service_name in docker_stack_services(module,
                stack_name):
189             ret[service_name] = docker_service_inspect(
                    module, service_name)
190         return ret
191
192
193     def docker_stack_rm(module, stack_name, retries,
            interval):
194         docker_bin = module.get_bin_path('docker', required
                =True)
195         command = [docker_bin, "stack", "rm", stack_name]
196
197         rc, out, err = module.run_command(command)
198
199         while err != "Nothing found in stack: %s\n" %
                stack_name and retries > 0:
200             sleep(interval)
201             retries = retries - 1
202             rc, out, err = module.run_command(command)
203         return rc, out, err
204
205
206     def main():
207         module = AnsibleModule(
208             argument_spec={
209                 'name': dict(required=True, type='str'),
210                 'compose': dict(required=False, type='list'
                        , default=[]),
211                 'prune': dict(default=False, type='bool'),
212                 'with_registry_auth': dict(default=False,
                        type='bool'),
213                 'resolve_image': dict(type='str', choices=[
                        'always', 'changed', 'never']),
214                 'state': dict(default='present', choices=['
                        present', 'absent']),
215                 'absent_retries': dict(type='int', default
                        =0),
216                 'absent_retries_interval': dict(type='int',
                         default=1)
```

```
217             },
218             supports_check_mode=False
219         )
220
221         if not HAS_JSONDIFF:
222             return module.fail_json(msg="jsondiff is not
                    installed, try 'pip install jsondiff'")
223
224         if not HAS_YAML:
225             return module.fail_json(msg="yaml is not
                    installed, try 'pip install pyyaml'")
226
227         state = module.params['state']
228         compose = module.params['compose']
229         name = module.params['name']
230         absent_retries = module.params['absent_retries']
231         absent_retries_interval = module.params['
                absent_retries_interval']
232
233         if state == 'present':
234             if not compose:
235                 module.fail_json(msg=("compose parameter
                        must be a list "
236                                       "containing at least
                                           one element"))
237
238             compose_files = []
239             for i, compose_def in enumerate(compose):
240                 if isinstance(compose_def, dict):
241                     compose_file_fd, compose_file =
                            tempfile.mkstemp()
242                     module.add_cleanup_file(compose_file)
243                     with os.fdopen(compose_file_fd, 'w') as
                            stack_file:
244                         compose_files.append(compose_file)
245                         stack_file.write(yaml_dump(
                                compose_def))
246                 elif isinstance(compose_def, string_types):
247                     compose_files.append(compose_def)
248                 else:
249                     module.fail_json(msg="compose element
                            '%s' must be a " +
250                                          "string or a
                                             dictionary" %
                                             compose_def)
251
252             before_stack_services = docker_stack_inspect(
                    module, name)
253
254             rc, out, err = docker_stack_deploy(module, name
                    , compose_files)
255
```

```
256             after_stack_services = docker_stack_inspect(
                    module, name)
257
258             if rc != 0:
259                 module.fail_json(msg="docker stack up
                        deploy command failed",
260                                  out=out,
261                                  rc=rc, err=err)
262
263             before_after_differences = json_diff(
                    before_stack_services,
264                                                  after_stack_services
                                                        )
265             for k in before_after_differences.keys():
266                 if isinstance(before_after_differences[k],
                        dict):
267                     before_after_differences[k].pop('
                            UpdatedAt', None)
268                     before_after_differences[k].pop('
                            Version', None)
269                     if not list(before_after_differences[k
                            ].keys()):
270                         before_after_differences.pop(k)
271
272             if not before_after_differences:
273                 module.exit_json(changed=False)
274             else:
275                 module.exit_json(
276                     changed=True,
277                     docker_stack_spec_diff=json_diff(
                            before_stack_services,
278                                                      after_stack_services
                                                            ,
279                                                      dump=
                                                            True
                                                            ))
280
281         else:
282             if docker_stack_services(module, name):
283                 rc, out, err = docker_stack_rm(module, name
                        , absent_retries,
                        absent_retries_interval)
284                 if rc != 0:
285                     module.fail_json(msg="'docker stack
                            down' command failed",
286                                      out=out,
287                                      rc=rc,
288                                      err=err)
289                 else:
290                     module.exit_json(changed=True, msg=out,
                            err=err, rc=rc)
291             module.exit_json(changed=False)
292
```

```
293
294 if __name__ == "__main__":
295     main()
```

# F Docker

## F.1 Nginx-php-fpm

```
FROM debian:stretch
MAINTAINER Birger "birgermy@stud.ntnu.no"

# Let the container know that there is no tty
ENV DEBIAN_FRONTEND noninteractive
ENV NGINX_VERSION 1.15.9-1~stretch
ENV php_conf /etc/php/7.3/fpm/php.ini
ENV fpm_conf /etc/php/7.3/fpm/pool.d/www.conf
ENV COMPOSER_VERSION 1.8.4
#FILEBEAT
ENV FILEBEAT_VERSION 6.6.1


# Install Basic Requirements
RUN apt-get update \
    && apt-get install --no-install-recommends --no-install-
       suggests -q -y gnupg2 dirmngr wget apt-transport-
       https lsb-release ca-certificates \
    && \
    NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
          found=''; \
          for server in \
                  ha.pool.sks-keyservers.net \
                  hkp://keyserver.ubuntu.com:80 \
                  hkp://p80.pool.sks-keyservers.net:80 \
                  pgp.mit.edu \
          ; do \
                  echo "Fetching GPG key $NGINX_GPGKEY from
                     $server"; \
                  apt-key adv --batch --keyserver "$server"
                     --keyserver-options timeout=10 --recv-
                     keys "$NGINX_GPGKEY" && found=yes &&
                     break; \
          done; \
    test -z "$found" && echo >&2 "error: failed to fetch GPG
        key $NGINX_GPGKEY" && exit 1; \
    echo "deb http://nginx.org/packages/mainline/debian/
       stretch nginx" >> /etc/apt/sources.list \
    && wget -O /etc/apt/trusted.gpg.d/php.gpg https://
       packages.sury.org/php/apt.gpg \
    && echo "deb https://packages.sury.org/php/ $(
       lsb_release -sc) main" > /etc/apt/sources.list.d/php.
       list \
    && apt-get update \
```

```
    && apt-get install --no-install-recommends --no-install-
       suggests -q -y \
            apt-utils \
            curl \
            nano \
            zip \
            unzip \
            python-pip \
            python-setuptools \
            git \
            nginx=${NGINX_VERSION} \
            php7.3-fpm \
            php7.3-cli \
            php7.3-bcmath \
            php7.3-dev \
            php7.3-common \
            php7.3-json \
            php7.3-opcache \
            php7.3-readline \
            php7.3-mbstring \
            php7.3-curl \
            php7.3-imagick \
            php7.3-gd \
            php7.3-mysql \
            php7.3-zip \
            php7.3-pgsql \
            php7.3-intl \
            php7.3-xml \
            php7.3-redis \
    && mkdir -p /run/php \
    && pip install wheel \
    && pip install supervisor supervisor-stdout \
    && echo "#!/bin/sh\nexit 0" > /usr/sbin/policy-rc.d \
    && rm -rf /etc/nginx/conf.d/default.conf \
    && sed -i -e "s/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/g
       " ${php_conf} \
    && sed -i -e "s/memory_limit\s*=\s*.*/memory_limit = 256
       M/g" ${php_conf} \
    && sed -i -e "s/upload_max_filesize\s*=\s*2M/
       upload_max_filesize = 100M/g" ${php_conf} \
    && sed -i -e "s/post_max_size\s*=\s*8M/post_max_size =
       100M/g" ${php_conf} \
    && sed -i -e "s/variables_order = \"GPCS\"/
       variables_order = \"EGPCS\"/g" ${php_conf} \
    && sed -i -e "s/;daemonize\s*=\s*yes/daemonize = no/g" /
       etc/php/7.3/fpm/php-fpm.conf \
    && sed -i -e "s/;catch_workers_output\s*=\s*yes/
       catch_workers_output = yes/g" ${fpm_conf} \
    && sed -i -e "s/pm.max_children = 5/pm.max_children = 4/
       g" ${fpm_conf} \
    && sed -i -e "s/pm.start_servers = 2/pm.start_servers =
       3/g" ${fpm_conf} \
```

```
    && sed -i -e "s/pm.min_spare_servers = 1/pm.
       min_spare_servers = 2/g" ${fpm_conf} \
    && sed -i -e "s/pm.max_spare_servers = 3/pm.
       max_spare_servers = 4/g" ${fpm_conf} \
    && sed -i -e "s/pm.max_requests = 500/pm.max_requests =
       200/g" ${fpm_conf} \
    && sed -i -e "s/www-data/nginx/g" ${fpm_conf} \
    && sed -i -e "s/^;clear_env = no$/clear_env = no/" ${
       fpm_conf} \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

RUN curl -o /tmp/composer-setup.php https://getcomposer.org/
   installer \
  && curl -o /tmp/composer-setup.sig https://composer.github
     .io/installer.sig \
  && php -r "if (hash('SHA384', file_get_contents('/tmp/
     composer-setup.php')) !== trim(file_get_contents('/tmp/
     composer-setup.sig'))) { unlink('/tmp/composer-setup.
     php'); echo 'Invalid installer' . PHP_EOL; exit(1); }"
     \
  && php /tmp/composer-setup.php --no-ansi --install-dir=/
     usr/local/bin --filename=composer --version=${
     COMPOSER_VERSION} && rm -rf /tmp/composer-setup.php

#FILEBEAT - INSTALL
RUN curl -L -O https://artifacts.elastic.co/downloads/beats/
   filebeat/filebeat-${FILEBEAT_VERSION}-amd64.deb \
 && dpkg -i filebeat-${FILEBEAT_VERSION}-amd64.deb \
 && rm filebeat-${FILEBEAT_VERSION}-amd64.deb

#FILEBEAT - CONFIGURATION
ADD filebeat/filebeat.yml /etc/filebeat/filebeat.yml
RUN chmod 644 /etc/filebeat/filebeat.yml

#FILEBEAT-  CA cert
RUN mkdir -p /etc/pki/tls/certs
ADD filebeat/logstash-beats.crt /etc/pki/tls/certs/logstash-
   beats.crt

# FILEBEAT- create template based on filebeat version (
   assumption: it is the same version as elasticsearch
   version)
RUN filebeat export template --es.version ${FILEBEAT_VERSION
   } > /etc/filebeat/filebeat.template.json


# Supervisor config
ADD ./supervisord.conf /etc/supervisord.conf

# Override nginx's default config
ADD ./default.conf /etc/nginx/conf.d/default.conf

# Override default nginx welcome page
```

```
COPY html /usr/share/nginx/html



# Add Scripts
ADD ./start.sh /start.sh

EXPOSE 80

CMD ["/start.sh"]
```

## F.2 Craft CMS

```
FROM basewebtest:latest

MAINTAINER  birgermy@ntnu.stud.no

# Remove existing webroot, configure PHP session handler for
    Redis, install postgresql-client (pg_dump)
RUN rm -rf /usr/share/nginx/* && \
sed -i -e "s/memory_limit\s*=\s*.*/memory_limit = 256M/g" ${
   php_conf} && \
sed -i -e "s/session.save_handler\s*=\s*.*/session.
   save_handler = redis/g" ${php_conf} && \
sed -i -e "s/;session.save_path\s*=\s*.*/session.save_path =
    \"\${REDIS_PORT_6379_TCP}\"/g" ${php_conf} && \
apt-get update

# Create Craft project
RUN composer create-project craftcms/craft /usr/share/nginx/

# Install the yii2-redis library
RUN composer require --prefer-dist yiisoft/yii2-redis -d /
   usr/share/nginx/

# Add default craft cms nginx config
ADD ./default.conf /etc/nginx/conf.d/default.conf

# Add database environment
ADD .env.sample /usr/share/nginx/.env

# Add default config
ADD ./config /usr/share/nginx/config

RUN chown -Rf nginx:nginx /usr/share/nginx/

EXPOSE 80
```

## F.3 Elastic Stack

```
 # Dockerfile for ELK stack
# Elasticsearch, Logstash, Kibana 6.6.1
```

```
# Build with:
# docker build -t <repo-user>/elk .

# Run with:
# docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it --
    name elk <repo-user>/elk

FROM phusion/baseimage
MAINTAINER Ellera Project
ENV REFRESHED_AT 2017-02-28


############################################################
                    INSTALLATION
############################################################

### install prerequisites (cURL, gosu, JDK, tzdata)

ENV GOSU_VERSION 1.10

ARG DEBIAN_FRONTEND=noninteractive
RUN set -x \
 && apt-get update -qq \
 && apt-get install -qqy --no-install-recommends ca-
    certificates curl \
 && rm -rf /var/lib/apt/lists/* \
 && curl -L -o /usr/local/bin/gosu "https://github.com/
    tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg
    --print-architecture)" \
 && curl -L -o /usr/local/bin/gosu.asc "https://github.com/
    tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg
    --print-architecture).asc" \
 && export GNUPGHOME="$(mktemp -d)" \
 && gpg --keyserver hkp://ha.pool.sks-keyservers.net:80 --
    recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \
 && gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/
    bin/gosu \
 && rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc \
 && chmod +x /usr/local/bin/gosu \
 && gosu nobody true \
 && apt-get update -qq \
 && apt-get install -qqy openjdk-8-jdk tzdata \
 && apt-get clean \
 && set +x


ENV ELK_VERSION 6.6.1

### install Elasticsearch

ENV ES_VERSION ${ELK_VERSION}
ENV ES_HOME /opt/elasticsearch
```

```
ENV ES_PACKAGE elasticsearch-${ES_VERSION}.tar.gz
ENV ES_GID 991
ENV ES_UID 991
ENV ES_PATH_CONF /etc/elasticsearch
ENV ES_PATH_BACKUP /var/backups

RUN mkdir ${ES_HOME} \
 && curl -O https://artifacts.elastic.co/downloads/
    elasticsearch/${ES_PACKAGE} \
 && tar xzf ${ES_PACKAGE} -C ${ES_HOME} --strip-components=1
     \
 && rm -f ${ES_PACKAGE} \
 && groupadd -r elasticsearch -g ${ES_GID} \
 && useradd -r -s /usr/sbin/nologin -M -c "Elasticsearch
    service user" -u ${ES_UID} -g elasticsearch
    elasticsearch \
 && mkdir -p /var/log/elasticsearch ${ES_PATH_CONF} ${
    ES_PATH_CONF}/scripts /var/lib/elasticsearch ${
    ES_PATH_BACKUP} \
 && chown -R elasticsearch:elasticsearch ${ES_HOME} /var/log
    /elasticsearch /var/lib/elasticsearch ${ES_PATH_CONF} ${
    ES_PATH_BACKUP}

ADD ./elasticsearch-init /etc/init.d/elasticsearch
RUN sed -i -e 's#^ES_HOME=$#ES_HOME='$ES_HOME'#' /etc/init.d
   /elasticsearch \
 && chmod +x /etc/init.d/elasticsearch


### install Logstash

ENV LOGSTASH_VERSION ${ELK_VERSION}
ENV LOGSTASH_HOME /opt/logstash
ENV LOGSTASH_PACKAGE logstash-${LOGSTASH_VERSION}.tar.gz
ENV LOGSTASH_GID 992
ENV LOGSTASH_UID 992
ENV LOGSTASH_PATH_CONF /etc/logstash
ENV LOGSTASH_PATH_SETTINGS ${LOGSTASH_HOME}/config

RUN mkdir ${LOGSTASH_HOME} \
 && curl -O https://artifacts.elastic.co/downloads/logstash/
    ${LOGSTASH_PACKAGE} \
 && tar xzf ${LOGSTASH_PACKAGE} -C ${LOGSTASH_HOME} --strip-
    components=1 \
 && rm -f ${LOGSTASH_PACKAGE} \
 && groupadd -r logstash -g ${LOGSTASH_GID} \
 && useradd -r -s /usr/sbin/nologin -d ${LOGSTASH_HOME} -c "
    Logstash service user" -u ${LOGSTASH_UID} -g logstash
    logstash \
 && mkdir -p /var/log/logstash ${LOGSTASH_PATH_CONF}/conf.d
    \
 && chown -R logstash:logstash ${LOGSTASH_HOME} /var/log/
    logstash ${LOGSTASH_PATH_CONF}
```

```
ADD ./logstash-init /etc/init.d/logstash
RUN sed -i -e 's#^LS_HOME=$#LS_HOME='$LOGSTASH_HOME'#' /etc/
    init.d/logstash \
 && chmod +x /etc/init.d/logstash


### install Kibana

ENV KIBANA_VERSION ${ELK_VERSION}
ENV KIBANA_HOME /opt/kibana
ENV KIBANA_PACKAGE kibana-${KIBANA_VERSION}-linux-x86_64.tar
    .gz
ENV KIBANA_GID 993
ENV KIBANA_UID 993

RUN mkdir ${KIBANA_HOME} \
 && curl -O https://artifacts.elastic.co/downloads/kibana/${
     KIBANA_PACKAGE} \
 && tar xzf ${KIBANA_PACKAGE} -C ${KIBANA_HOME} --strip-
     components=1 \
 && rm -f ${KIBANA_PACKAGE} \
 && groupadd -r kibana -g ${KIBANA_GID} \
 && useradd -r -s /usr/sbin/nologin -d ${KIBANA_HOME} -c "
     Kibana service user" -u ${KIBANA_UID} -g kibana kibana \
 && mkdir -p /var/log/kibana \
 && chown -R kibana:kibana ${KIBANA_HOME} /var/log/kibana

ADD ./kibana-init /etc/init.d/kibana
RUN sed -i -e 's#^KIBANA_HOME=$#KIBANA_HOME='$KIBANA_HOME'#'
     /etc/init.d/kibana \
 && chmod +x /etc/init.d/kibana


################################################

   CONFIGURATION
################################################

### configure Elasticsearch

ADD ./elasticsearch.yml ${ES_PATH_CONF}/elasticsearch.yml
ADD ./elasticsearch-default /etc/default/elasticsearch
RUN cp ${ES_HOME}/config/log4j2.properties ${ES_HOME}/config
    /jvm.options \
     ${ES_PATH_CONF} \
 && chown -R elasticsearch:elasticsearch ${ES_PATH_CONF} \
 && chmod -R +r ${ES_PATH_CONF}

### configure Logstash

# certs/keys for Beats and Lumberjack input
```

```
RUN mkdir -p /etc/pki/tls/certs && mkdir /etc/pki/tls/
    private
ADD ./logstash-beats.crt /etc/pki/tls/certs/logstash-beats.
    crt
ADD ./logstash-beats.key /etc/pki/tls/private/logstash-beats
    .key

# pipelines
ADD pipelines.yml ${LOGSTASH_PATH_SETTINGS}/pipelines.yml

# filters
ADD ./02-beats-input.conf ${LOGSTASH_PATH_CONF}/conf.d/02-
    beats-input.conf
ADD ./10-syslog.conf ${LOGSTASH_PATH_CONF}/conf.d/10-syslog.
    conf
ADD ./11-nginx.conf ${LOGSTASH_PATH_CONF}/conf.d/11-nginx.
    conf
ADD ./30-output.conf ${LOGSTASH_PATH_CONF}/conf.d/30-output.
    conf

# patterns
ADD ./nginx.pattern ${LOGSTASH_HOME}/patterns/nginx
RUN chown -R logstash:logstash ${LOGSTASH_HOME}/patterns

# Fix permissions
RUN chmod -R +r ${LOGSTASH_PATH_CONF} ${
    LOGSTASH_PATH_SETTINGS} \
 && chown -R logstash:logstash ${LOGSTASH_PATH_SETTINGS}

### configure logrotate

ADD ./elasticsearch-logrotate /etc/logrotate.d/elasticsearch
ADD ./logstash-logrotate /etc/logrotate.d/logstash
ADD ./kibana-logrotate /etc/logrotate.d/kibana
RUN chmod 644 /etc/logrotate.d/elasticsearch \
 && chmod 644 /etc/logrotate.d/logstash \
 && chmod 644 /etc/logrotate.d/kibana


### configure Kibana

ADD ./kibana.yml ${KIBANA_HOME}/config/kibana.yml


################################################
                    START
################################################

ADD ./start.sh /usr/local/bin/start.sh
RUN chmod +x /usr/local/bin/start.sh

EXPOSE 5601 9200 9300 5044
VOLUME /var/lib/elasticsearch
```

```
CMD [ "/usr/local/bin/start.sh" ]
```

## F.4 MySQL

```
 FROM debian:stretch-slim

# add our user and group first to make sure their IDs get
   assigned consistently, regardless of whatever
   dependencies get added
RUN groupadd -r mysql && useradd -r -g mysql mysql

RUN apt-get update && apt-get install -y --no-install-
   recommends gnupg dirmngr && rm -rf /var/lib/apt/lists/*

# add gosu for easy step-down from root
ENV GOSU_VERSION 1.7
RUN set -x \
        && apt-get update && apt-get install -y --no-install
           -recommends ca-certificates wget && rm -rf /var/
           lib/apt/lists/* \
        && wget -O /usr/local/bin/gosu "https://github.com/
           tianon/gosu/releases/download/$GOSU_VERSION/gosu-
           $(dpkg --print-architecture)" \
        && wget -O /usr/local/bin/gosu.asc "https://github.
           com/tianon/gosu/releases/download/$GOSU_VERSION/
           gosu-$(dpkg --print-architecture).asc" \
        && export GNUPGHOME="$(mktemp -d)" \
        && gpg --batch --keyserver ha.pool.sks-keyservers.
           net --recv-keys
           B42F6819007F00F88E364FD4036A9C25BF357DD4 \
        && gpg --batch --verify /usr/local/bin/gosu.asc /usr
           /local/bin/gosu \
        && gpgconf --kill all \
        && rm -rf "$GNUPGHOME" /usr/local/bin/gosu.asc \
        && chmod +x /usr/local/bin/gosu \
        && gosu nobody true \
        && apt-get purge -y --auto-remove ca-certificates
           wget

RUN mkdir /docker-entrypoint-initdb.d

RUN apt-get update && apt-get install -y --no-install-
   recommends \
# for MYSQL_RANDOM_ROOT_PASSWORD
                pwgen \
# for mysql_ssl_rsa_setup
                openssl \
# FATAL ERROR: please install the following Perl modules
   before executing /usr/local/mysql/scripts/
   mysql_install_db:
# File::Basename
# File::Copy
```

```
# Sys::Hostname
# Data::Dumper
                    perl \
        && rm -rf /var/lib/apt/lists/*

RUN set -ex; \
# gpg: key 5072E1F5: public key "MySQL Release Engineering <
    mysql-build@oss.oracle.com>" imported
        key='A4A9406876FCBD3C456770C88C718D3B5072E1F5'; \
        export GNUPGHOME="$(mktemp -d)"; \
        gpg --batch --keyserver ha.pool.sks-keyservers.net
            --recv-keys "$key"; \
        gpg --batch --export "$key" > /etc/apt/trusted.gpg.d
            /mysql.gpg; \
        gpgconf --kill all; \
        rm -rf "$GNUPGHOME"; \
        apt-key list > /dev/null

ENV MYSQL_MAJOR 5.7
ENV MYSQL_VERSION 5.7.25-1debian9

RUN echo "deb http://repo.mysql.com/apt/debian/ stretch
    mysql-${MYSQL_MAJOR}" > /etc/apt/sources.list.d/mysql.
    list

# the "/var/lib/mysql" stuff here is because the mysql-
    server postinst doesn't have an explicit way to disable
    the mysql_install_db codepath besides having a database
    already "configured" (ie, stuff in /var/lib/mysql/mysql)
# also, we set debconf keys to make APT a little quieter
RUN { \
                echo mysql-community-server mysql-community-
                    server/data-dir select ''; \
                echo mysql-community-server mysql-community-
                    server/root-pass password ''; \
                echo mysql-community-server mysql-community-
                    server/re-root-pass password ''; \
                echo mysql-community-server mysql-community-
                    server/remove-test-db select false; \
        } | debconf-set-selections \
        && apt-get update && apt-get install -y mysql-server
            ="${MYSQL_VERSION}" && rm -rf /var/lib/apt/lists
            /* \
        && rm -rf /var/lib/mysql && mkdir -p /var/lib/mysql
            /var/run/mysqld \
        && chown -R mysql:mysql /var/lib/mysql /var/run/
            mysqld \
# ensure that /var/run/mysqld (used for socket and lock
    files) is writable regardless of the UID our mysqld
    instance ends up having at runtime
        && chmod 777 /var/run/mysqld \
# comment out a few problematic configuration values
        && find /etc/mysql/ -name '*.cnf' -print0 \
```

```
                | xargs -0 grep -lZE '^(bind-address|log)' \
                | xargs -rt -0 sed -Ei 's/^(bind-address|log
                    )/#&/' \
# don't reverse lookup hostnames, they are usually another
    container
        && echo '[mysqld]\nskip-host-cache\nskip-name-
            resolve' > /etc/mysql/conf.d/docker.cnf

VOLUME /var/lib/mysql

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh /entrypoint.sh
    # backwards compat
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 3306 33060
CMD ["mysqld"]
```

# G Nginx Config

```
server {
    listen   80; ## listen for ipv4; this line is default
       and implied
    listen   [::]:80 default ipv6only=on; ## listen for ipv6

    root /usr/share/nginx/html;
    index index.php index.html index.htm;

    # Make site accessible from http://localhost/
    server_name _;

    # Disable sendfile as per https://docs.vagrantup.com/v2/
       synced-folders/virtualbox.html
    sendfile off;

    # Security - Hide nginx version number in error pages
       and Server header
    server_tokens off;

    # Add logging to file for Filebeat
    error_log /var/log/nginx/error.log info;
    access_log /var/log/nginx/access.log;

    # reduce the data that needs to be sent over network
    gzip on;
    gzip_min_length 10240;
    gzip_proxied expired no-cache no-store private auth;
    gzip_types text/plain text/css text/xml application/json
        text/javascript application/x-javascript application
       /xml;
    gzip_disable "MSIE [1-6]\.";

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to index.html
        try_files $uri $uri/ =404;
    }

    # redirect server error pages to the static page /50x.
       html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
```

```
    # pass the PHP scripts to FastCGI server listening on
       socket
    #
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass unix:/run/php/php7.3-fpm.sock;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
           $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }

        location ~* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ {
                expires           5d;
        }

    # deny access to . files, for security
    #
    location ~ /\. {
            log_not_found off;
            deny all;
    }

}
```

# H Supervisord.conf

```
[unix_http_server]
file=/tmp/supervisor.sock   ; (the path to the socket file)
username=nobody
password=nobody

[supervisord]
logfile=/tmp/supervisord.log ; (main log file;default $CWD/
   supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4
   rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation
    backups;default 10)
loglevel=info                ; (log level;default info;
   others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default
   supervisord.pid)
nodaemon=false               ; (start in foreground if true;
   default false)
minfds=1024                  ; (min. avail startup file
   descriptors;default 1024)
minprocs=200                 ; (min. avail process
   descriptors;default 200)
user=root                    ; (default is current user,
   required if root)

; the below section must remain in the config file for RPC
; (supervisorctl/web interface) to work, additional
   interfaces may be
; added by defining them in separate rpcinterface: sections
[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:
   make_main_rpcinterface

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL
   for a unix socket

[program:php-fpm7]
command=/usr/sbin/php-fpm7.3 --nodaemonize --fpm-config=/etc
   /php/7.3/fpm/pool.d/www.conf
autostart=true
autorestart=true
priority=5
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

```
[program:nginx]
command=/usr/sbin/nginx -g "daemon off;"
autostart=true
autorestart=true
priority=10
stdout_events_enabled=true
stderr_events_enabled=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[eventlistener:stdout]
command = supervisor_stdout
buffer_size = 100
events = PROCESS_LOG
result_handler = supervisor_stdout:event_handler
```

# I Elastic Stack

## I.1 Filebeat.yml

Listing I.1: filebeat.yml

```
output:
  logstash:
    enabled: true
    hosts:
      - elk:5044
    timeout: 15
    ssl:
      certificate_authorities:
      - /etc/pki/tls/certs/logstash-beats.crt

filebeat:
  prospectors:
    -
      paths:
        - "/var/log/nginx/access.log"
      document_type: nginx-access
    -
      paths:
        - "/var/log/nginx/error.log"
      document_type: nginx-error
```

## I.2 11-nginx.conf

Listing I.2: 11-nginx.conf

```
filter {
  if [type] == "nginx-access" {
    grok {
      match => { "message" => "%{NGINXACCESS}" }
    }
  }
}
```

## I.3 Nginx pattern

Listing I.3: nginx.pattern

```
NGUSERNAME [a-zA-Z\.\@\-\+_%]+
NGUSER %{NGUSERNAME}
NGINXACCESS %{IPORHOST:clientip} %{NGUSER:ident} %{
    NGUSER:auth} \[%{HTTPDATE:timestamp}\] "%{WORD:verb}
     %{URIPATHPARAM:request} HTTP/%{NUMBER:httpversion}"
     %{NUMBER:response:int} (?:%{NUMBER:bytes:int}|-)
```

```
        (?:"(?:%{URI:referrer}|-)"|%{QS:referrer}) %{QS:
        agent}
```

# J Prosjekt Plan

# Prosjektplan

15. mai 2019

**Bacheloroppgave for Ellera**

Birger Myhrvold (997829)
Daniel Bruun (473117)
Jostein Vole Hage (999551)

# Innhold

# 1 Mål og rammer

## 1.1 Bakgrunn

Ellera ble stiftet i 14.01.2018 og registrert som et aksjeselskap 24.01.2018 av Simen Andreas Eira og Jørgen Ellingsen. Ellera utfører tjenester ved rådgivning, konsulentarbeid og produktutvikling innenfor informasjonsteknologi. Som et ungt firma i utvikling, ser Ellera etter alternativer for å håndtere drift optimalt. Project Brisbane er selve kjernen i firmaet, og skal gjøre drifting for kundenes behov effektiv og sikker.

## 1.2 Prosjektmål

Denne delen inneholder hvilke mål som skal oppnås med prosjektet.

### 1.2.1 Resultatmål

Hovedmålet med oppgaven er å utarbeide en fullverdig"/fungerende infrastruktur som forenkler drift og videreutvikling ved hjelp av konfigurasjonsstyringssystemer. Eksempler på dette er:

- Continous Integration / Continous Deployment.
- Backup: Automatisk backup.
- Logging og monitorering.
- Automatisk deployering og testing.

Det skal også produseres en rapport som inneholder oversikt over valg, faglige vurderinger og anbefalinger som en følge av diskusjoner og vurdering av tidligere forskning og oppslagsverk.

Gruppen har som mål å tilegne seg større grad av forståelse og kunnskap innenfor et større fagområde, samt teknologier som vil være passende ved oppsett av infrastruktur og CI/CD pipeline.

### 1.2.2 Effektmål

Prosjektets rapport skal gi Ellera grunnlag for valg av teknologier som best vil komplementere firmaets skreddersydde infrastruktur. Kildekode fra gruppens infrastruktur skal være til hjelp for Ellera sin videre utvikling av drifting innad i selskapet.

## 1.3 Rammer

Selv om oppdragsgiver har gjort det åpent for gruppen når det gjelder utførelse av oppgave, er det allikevel blitt satt rammer og begrensninger som må følges.

### 1.3.1 Tidsramme

Prosjektplan skal leveres den 1.Februar (dette dokumentet). Prosjektet har en tidsramme hvor hoveddokumentet, rapporten, skal være ferdig og levert 20.mai 2019. Arbeidsperioden for oppgaven skjer mellom oppstart av skoleår (07.januar) frem til leveringsdato.

### 1.3.2 Økonomiske rammer

Prosjektet i sin helhet vil ikke ha noen store økonomiske kostnader, hverken innkjøp av programlisenser eller av materiell er en forutsetning. Eventuelle reisekostnader, om nødvendig, vil bli fordelt mellom gruppemedlemmene.

### 1.3.3 Kravspesifikke rammer

Oppdragsgiver har følgende ønsker til funksjonalitet i infrastrukturen:

- Miljøet for applikasjonsutvikling skal ha "one-click"installasjon på localhost, som klargjør systemet fullstendig for utvikling
- Deplyment pipeline skal ha automatisk testing basert på commits i BitBucket
- Docker Swarm skal brukes for container-forvaltning og redundans
- Backup skal foregå automatisk og off-site

### 1.3.4 Teknologisk

Til utforming og skriving av rapporten benyttes Overleaf, Overleaf er tidligere ShareLaTeX. Som testmiljø for oppsett av infrastruktur benyttes SkyHigh, som er NTNUs egne skyløsning som benytter OpenStack. For å utarbeide en løsning til problemstillingen vil gruppen ta hensyn til flere aktuelle teknologier som Nomad, Kubernetes etc. Delegering av oppgaver og status føres via Trello. Gruppen har en egen informasjonskanal som er tilgjengelig via Discord hvor kontaktperson også har tilgang. Logging av timer gjøres i Google Docs. Gruppens medlemmer har egne regneark hvor timene per deltager registreres manuelt, og hvordan disse er benyttet.

#### 1.3.5 Utstyr

Utover eget utstyr, er gruppen blitt tildelt ressurser og gitt tilgang til NTNU Gjøvik sin egne skytjeneste SkyHigh. Denne platformen vil fungere som utviklings- og testmiljø gjennom prosjektperioden.

# 2 Omfang

## 2.1 Problemområde

Ellera AS er i vekst, og stadig flere kunder vil at deres nettsider og/eller nettbaserte programvareløsninger skal driftes hos Ellera. For å møte kundenes forventninger til infrastrukturen blir skalering, automatisering og brukervennlighet nøkkelpunkter under videre utvikling. Ellera ønsker at infrastrukturen skal følge konseptene som en CI/CD pipeline medfører, og at teknologiene som velges for bruk i infrastrukturen skal komplementere bruksområde.

## 2.2 Problemavgrensning

Ettersom driftingen for Ellera er skreddersydd til deres behov, vil det kreve at bruksområdene og funksjonaliteten også stemmer overens med prosjektets infrastruktur. Det vil si at funksjonalitet og bruksområde vil avgrenses til å samsvare med Ellera's infrastruktur slik som det var ved prosjektets oppstart. Allikevel kan det ikke påregnes at gruppens infrastruktur kan direkte tas i bruk for kommersielt bruk. Ellera bør selv studere prosjektets kildekode, og vurdere det som inspirasjon for egen bruk etter endt oppdrag.

## 2.3 Oppgavebeskrivelse

På vegne av Ellera skal gruppen skape en infrastruktur, som støtter driften selskapet har i dag på en mer effektiv, oversiktlig og organisert måte. Under prosessen skal det legges vekt på ulike teknologier, og gjøres en anbefalning av disse, der aktuelle kandidater vil kunne benyttes i et eksempeloppsett. Prosjektgruppen skal komme frem til et eksempeloppsett på en infrastruktur, som også muliggjør for automatisert provisjonering av servere. Formålet er å sette opp en CI/CD pipeline som kan forenkle utrullingen av oppdateringer for eksisterende applikasjoner eller tjenester. Skalering av containere skal kunne utføres automatisk, og tilføyes den eksisterende infrastrukturen uten manuell innblanding.

Med automatisering følger naturlig gode løsninger på logging og automatisk testing. Ellera har ingen gode løsninger på logg-aggregering, error-reporting eller automatisk

testing. Det vil derfor være nødvendig for gruppen å finne gode løsninger på disse elementene, og implementere de.

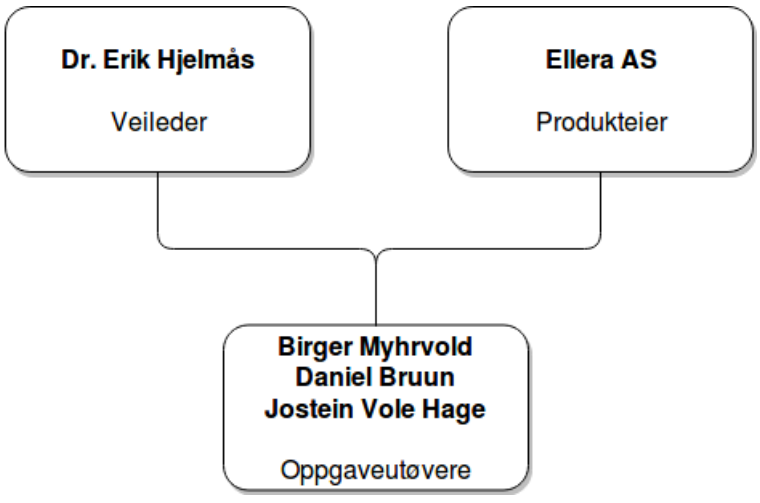
Gruppen må også vurdere hvorvidt skalering av database- og redis-containerne vil være nødvendig.

Utvalg av teknologier vil baseres på ønsker fra oppdragsgiver, omfang av infrastrukturen, samt tjenester og applikasjoner som skal driftes. En vurdering skal utføres for å skape et inntrykk av de aktuelle kandidatene, for å kunne velge ut den som er best egnet til et eksempeloppsett. Det er opp til prosjektgruppen å sammenligne og tydeliggjøre de vesentlige forskjellene på en slik måte at valget virker klart for en hver leser av rapporten.

Planlagt oppsett på infrastrukturen er som følger:

- Konfigurasjonssystem som styrer installasjon og konfigurasjon for nettside-platform. Konfigurasjonsdata hentes fra BitBucket
- Containere som kjører Craft CMS med Composer
- Logging, samt konkrete feilmeldinger ved driftsproblemer
- Automatisk testing av kode som skal implementeres
- Database enten innenfor eller utenfor swarm

# 3 Prosjektorganisering




Figur 1: Organisasjonskart for prosjektet

## 3.1 Gruppemedlemmer

Gruppen består av tre medlemmer, Jostein Vole Hage, Daniel Bruun og Birger Myhrvold. Alle studerer bachelor i IT-drift og Informasjonssikkerhet med oppstart i 2016. Fra tidligere har Daniel en bachelor i eiendomsmegling fra Høgskolen i Telemark. Alle har hatt valgfag i Infrastructure as Code og Software Security, men Jostein har i tillegg hatt Cloud Technologies. Parallelt med bacheloroppgaven har alle gruppemedlemmene valgfag i Campus nettverk og internettarkitektur. Gjennom fagene drift av tjenestearkitektur og infrastructure as code, har gruppen opparbeidet kunnskap om continuous integration, ulike arkitekturer og infrastrukturer, som kan bidra til arbeidet med bacheloroppgaven. Med utdanningen så langt, har gruppen generelt dannet seg et godt grunnlag for å takle oppgaven og eventuelt videre arbeidsliv.

## 3.2 Ansvarsforhold og roller

Daniel Bruun er gruppens leder og har ansvar for å forberede intervju og planlegge møter med oppdragsgiver og kontaktperson. Som gruppeleder har Daniel ansvar for å følge opp arbeidsoppgaver som er utdelt og sikre at disse blir utført i henhold til prosjektplanen.

Jostein Vole Hage har ansvar for smidig utvikling, og er utpekt til scrum master. Som scrum master har han ansvar for sprinter, og at fremdriften går som forventet.

Birger Myhrvold har ansvar for kodekvalitet og effektivisering innen infrastrukturen, samt oversikt over utviklingen av de forskjellige delene i infrastrukturen.

## 3.3 Rutiner og regler i gruppen

Medlemmene i gruppen tar utgangspunkt i å kunne møtes hver hverdag, med unntak av tirsdager, da denne dagen er forbeholdt forelesning og lab i IMT3691 (campusnettverk og internettarkitektur). Ordinær arbeidstid for felles arbeid er fra kl 0800 til 1600, med mindre annet er avklart på forhånd. Hvert medlem skal ukentlig jobbe 30 timer, og er selv ansvarlig for at disse logges. Ved unntak og spesielle tilfeller der gruppen ikke er samlet vil hvert enkelt medlem selv ha ansvar for å utføre den ukentlige arbeidsmegden og fordeling av denne.

Arbeidsoppgaver vil bli diskutert og delt ut i plenum - hvert gruppemedlem vil være ansvarlig for å utføre disse til avtalt tidspunkt. Skulle det oppstå usikkerhet med arbeidsoppgave eller andre problemer i løpet av denne tidsperioden, er de aktuelle medlemmene pliktige til å ta dette opp med resten av gruppen slik at en felles løsning kan utarbeides.
Grupperom bestilles to uker frem i tid, der rom og klokkeslett deles via felles informasjonskanal.

Gruppen har som mål i å skrive minimum 2-sider hver uke.

### 3.3.1 Grupperegler

1. Forventet Arbeidsinnsats
   (a) Hvert gruppemedlem skal jobbe 30 timer med prosjektet hver uke.
   (b) Arbeidet skal utføres mellom 08:00 og 16:00 Mandag til Fredag, med unntak av Tirsdager da denne dagen er forbeholdt arbeid med andre fag.
   (c) Arbeid som utføres utenfor dette tidsrommet vil ikke telle mot det ukentlige kravet. Hver enkelt har selv ansvar for logging av timer og utførte oppgaver.
   (d) Ved tilfeller der deadlines må holdes, vil overtidsarbeid kunne oppstå og det forventes at gruppemedlemmer deltar så langt det gjør seg mulig.
2. Aktiv Deltagelse
   (a) Ved møter og arbeid i fellesskap forventes det at gruppemedlemene deltar aktivt og viser initiativ for oppgavene som skal gjøres.
3. Møteplikt
   (a) Alle gruppemedlemmer er pliktige til å møte på avtalte møter og arbeidsdager.
   (b) Fravær vil bli notert, der antall timer og grunnlag blir oppført.
   (c) Ved fravær må dette meldes i god tid slik at eventuelle møter kan flyttes og arbeidsoppgaver kan omdelegeres.
4. Referat
   (a) Ved hvert møte skal det skrives referat. Dette skal inneholde fattede avgjørelser, hva som er utført, plan for fremdrift. Oppgaven om å skrive referat vil fordeles likt på gruppemedlemmene.
5. Arbeidsoppgaver
   (a) Hvert medlem er pliktig til å fullføre tildelte arbeidsoppgaver innen avtalt tidspunkt.
   (b) Om arbeidet ligger etter skjema skal dette meldes om i god tid slik at beslutninger om hvordan dette skal håndteres kan fattes.
6. Uenigheter
   (a) Ved uenigheter om prosjektets gjennomføring skal gruppen komme til enighet ved at det fattes en flertallsavgjørelse. Alle gruppemedlemmer er nødt til å avgi stemme.

(b) Ved tilfeller der gruppen ikke kommer til enighet vil uenigheten tas opp med veileder, der veileder vil ha rett til å avgjøre.

7. Innsats

   (a) Gruppen plikter å informere dersom de ikke er fornøyd med innsatsen til enkelte gruppemedlemmer.

8. Rutiner ved brudd på reglene

   (a) Samtale mellom alle gruppemedlemmer om problemet.

   (b) Ved gjentatte regelbrudd vil skriftlig advarsel mottas der:

      i. regelbruddene blir påpekt
      ii. konkret om hva som trengs for å bøte skaden
      iii. konkret om konsekvensene av manglende kravoppfyllelse

   (c) Samtale med hele gruppen og veileder

   (d) Ved alvorlige brudd på reglene vil skriftlig ekskludering fra gruppen mottas

# 4 Planlegging, oppfølging og rapportering

## 4.1 Hovedinndeling av prosjektet

### Valg av SU-modell med argumentasjon

Etter nøye undersøkelser og mye arbeid har gruppen kommet fram til at scrum er best egnet som utviklingsmodell. Mye av arbeidet som skal utføres er ikke helt bestemt og endringer kan oppstå underveis noe som gjør at en smidig modell vil være bedre, da den er bedre rustet for å tilpasse seg slike endringer. Det ville vært mulig å benytte en mer plandrevet modell, men dette vil påvirke endringsmulighetene negativt. Scrum vil føre til at prosjektarbeidet kommer i mål.

### Valg av metode og tilnærming (Avklare teori- og metodebruk)

Ettersom gruppen består av kun tre medlemmer er det nødvendig å justere omfanget av modellen deretter, slik at den passer bedre i forhold til oppgaven.

### Sprint

Sprintene er satt til å vare to uker, da gruppen har erfart fra tidligere prosjekter og oppgaver at en uke som oftest blir for hyppig. Disse vil kjøres fra mandager kl 10:00 og to uker frem i tid.

**Sprint Review Meeting  Sprint Planning Meeting**

Sprint review og sprint planning meeting vil gjøres sekvensielt. Det er ikke planlagt at oppdragsgiver og veileder skal delta på disse møtene, men resultatene fra møtene vil formidles videre.

**Product Backlog**

Trello vil bli benyttet for arbeidsoppgaver og deres status slik at det er lett å skaffe oversikt over hva som blir gjort og er blitt utført av oppgaver. wr

**Daily Scrum**

Innledningsvis vil hver møtedag ha et kort møte, der dagens agenda, og hvilke oppgaver som skal jobbes med blir diskutert.

**Scrum Master**

Jostein er satt til å være Scrum Master. Ansvaret vil være å spørre gruppemedlemmene om hva som er gjort dagen i forveien, hva som skal gjøres i dag og om det er noe som hindrer oppgaven i å bli utført.

## 4.2 Plan for statusmøter og beslutningspunkter i perioden

### 4.2.1 Interne møter

Interne møter vil følge valgt utviklingsmodell, der gruppen samles for å diskutere arbeidet og vise til resultater i forhold til prosjektplanen og målsetninger.

### 4.2.2 Møte med veileder

Møter med veileder er blitt avtalt og planlagt til å være ukentlig, hver Onsdag kl 09:00 - 09:30, så lenge ikke noe annet er avtalt.

### 4.2.3 Møte med kontaktperson

Når det gjelder kontaktpersonen er det en mer åpen dialog, der møter blir avtalt fortløpende. Dette fungerer utmerket da kontaktpersonen er lett tilgjengelig og åpen for møter via skype, discord etc og har sett seg villig til å reise til Gjøvik ved spesielle anledninger eller ved behov.

# 5 Organisering av kvalitetssikring

## 5.1 Dokumentasjon, standardbruk, konfigurasjonsstyring og kildekode

Ved praktisk testing og bruk av verktøy i gruppens infrastruktur, vil det være sterkt preg av hyppige notater og dokumentasjon på det som gjøres. Dette har høy prioritet, siden vår erfaring med infrastrukturen skal kunne forstås av leseren av rapporten.

For å sikre at testing av forskjellige verktøy blir gjort så effektivt som mulig, vil kildekoden brukt for konfigurasjon og eventuelle script, lagres med versjonskontroll på BitBucket. Dette gjøres ved bruk av Git, samtidig for å skape en oversiktlig struktur så langt det er mulig. Bruk av branches"vil tas i bruk for å holde testing og utviklingsmiljø adskilt.

I tillegg til adskilte miljøer ved hjelp av branches vil endringer og versjonshåndtering bli håndtert ved bruk av Git. Ved versjonskontroll vil de ulike versjonene av filer lagres slik at det er lett å gjenopprette tidligere konfigurasjoner hvis feil skulle oppstå ved oppdateringer.

## 5.2 Risikoanalyse

For å sikre seg best mulig mot potensielle problemer har gruppen gjennomført en risikoanalyse. Denne seksjonen tar for seg risikoer som kan oppstå under prosjektet, samt preventive og reaktive mottiltak.

Hver risiko vil bli vurdert i forhold til dens sannsynlighet for å skje, samt konsekvensen dersom den oppstår.

### Risikomatrise

Ved beregning av risikoverdi (S*K) vil det tas utgangspunkt i tabellen under. I risikomatrisen er sannsynlighet og konsekvens definert med en skala fra èn (1) til fire (4). Hvor èn er "Ubetydelig"og fire er "Alvorlig". 1

| Sannsynlighet | | | | | |
|---|---|---|---|---|---|
| | **4-Svært sannsynlig** | 4 | 8 | 12 | 16 |
| | **3-Sannsynlig** | 3 | 6 | 9 | 12 |
| | **2-Mindre sannsynlig** | 2 | 4 | 6 | 8 |
| | **1-Usannsynlig** | 1 | 2 | 3 | 4 |
| | | **1-Ubetydlig** | **2-Liten** | **3-Moderat** | **4-Alvorlig** |
| | | Konsekvens | | | |

Tabell 1: Risikomatrise

### 5.3 Vurdering av risikoer med tiltak

Den beregnede risikoverdien vil vurderes ut i fra tabellen under 2. Her gir hver fargesone et visuelt bilde på alvorlighetsgraden til den samlede risikoverdien. Fargen gul, oransje og rød innebærer at tiltak kreves eller er nødvendig. Grønn er en indikasjon hvor tiltak ikke er nødvendig, men kan være fordelaktig.

| **Farge** | **Alvorlighetsgrad** |
|---|---|
| 12-16 | Svært høy. Tiltak kreves |
| 8-9 | Høy. Grundige tiltak kreves |
| 4-6 | Moderat risiko. Enkelte tiltak kreves |
| 1-3 | Lav risiko. Akseptabel ingen tiltak kreves |

Tabell 2: Oversikt riskoverdi

| Risiko | Konsekvens | Sannsynlighet | K x S | Tiltak |
|---|---|---|---|---|
| 1. Langvarig sykdom | Moderat | Mindre sannsynlig | 3x2=6 | Ha god struktur og oversikt slik at andre enkelt kan overta eventuelle arbeidsoppgaver. |
| 2. Tap av data ved personalfeil, eller systemfeil | Alvorlig | Mindre sannsynlig | 4x2=8 | Sikre data ved hjelp av gode backup rutiner, både lokal og ekstern backup. |
| 3. Overskride tidsfrister | Alvorlig | Mindre sannsynlig | 4x2=8 | Struktur og gode rutiner slik at frister blir holdt. |
| 4. Medlem forlater eller sparkes ut av gruppen | Alvorlig | Mindre sannsynlig | 4x2=8 | God kommunikasjon, sørge for at alle medlemmer har innsikt i arbeidet som er blitt utført. |
| 5. Ikke få tilgang til NTNU's SkyHigh og nok ressurser | Moderat | Mindre sannsynlig | 3x2=6 | Be om mer ressurser eller benytte ekstern løsning. |
| 6. Ikke finne teknologi som er egnet for å løse oppgaven | Alvorlig | Mindre sannsynlig | 4x2=8 | Gjøre god research og avklare med oppdragsgiver om alternative løsninger. |
| 7. Store endringer i krav og forventninger fra oppdragsgiver | Moderat | Sannsynlig | 3x3=9 | Være smidig og ha god kommunikasjon med oppdragsgiver slik at endringer og nye krav kan oppdages tidlig slik at skadeomfanget blir minst mulig. |
| 8. Gruppemedlemmer motarbeider eller sabotere prosjektfremgang | Moderat | Mindre sannsynlig | 3x2=6 | Hvert enkelt gruppemedlem har forpliktet seg til prosjektet. Eventuelle problemer må løses internt, eventuelt vha. veileder. |
| 9. Sluttresultatet tilfredsstiller ikke kravene til oppdragsgiver | Moderat | Sannsynlig | 3x3=9 | God kommunikasjon med oppdragsgiver. Begge parter blir enige om hva som skal gjøres, og holde oppdragsgiver oppdatert underveis. |

### 5.4 Resterende risiko

Tabellen under viser resterende risikoverdi for og etter tiltak er blitt innført.

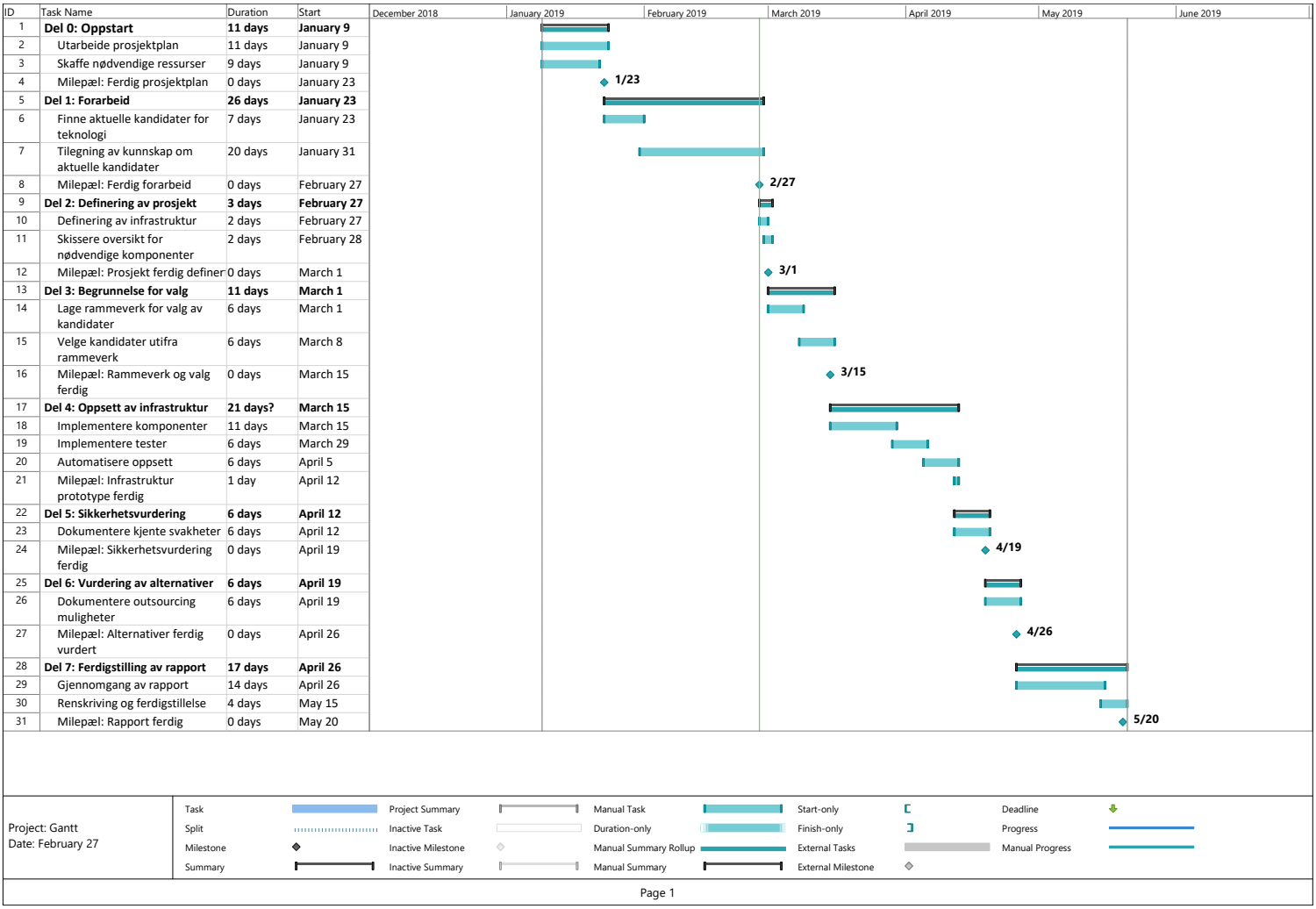
| Risiko # | K x S før tiltak | K x S etter tiltak |
|---|---|---|
| 1 | Moderat - 6 | 4 |
| 2 | Høy - 8 | 2 |
| 3 | Høy - 8 | 4 |
| 4 | Høy - 8 | 4 |
| 5 | Moderat - 6 | 2 |
| 6 | Høy - 8 | 4 |
| 7 | Høy - 9 | 4 |
| 8 | Moderat - 6 | 3 |
| 9 | Høy - 9 | 4 |

Tabell 3: Før og etter tiltak

# 6 Plan for gjennomføring - Gantt

Gantt er en type søylediagram som benyttes for å illustrere aktivitetene i et prosjekt, samt start- og sluttdato for disse. Den er med på å holde oversikt over frister og gir et estimat på hvor godt gruppen ligger ann underveis / til enhver tid.

Figur 2: Gantt-diagram



| ID | Task Name | Duration | Start |
|---|---|---|---|
| 1 | **Del 0: Oppstart** | **11 days** | **January 9** |
| 2 | Utarbeide prosjektplan | 11 days | January 9 |
| 3 | Skaffe nødvendige ressurser | 9 days | January 9 |
| 4 | Milepæl: Ferdig prosjektplan | 0 days | January 23 |
| 5 | **Del 1: Forarbeid** | **26 days** | **January 23** |
| 6 | Finne aktuelle kandidater for teknologi | 7 days | January 23 |
| 7 | Tilegning av kunnskap om aktuelle kandidater | 20 days | January 31 |
| 8 | Milepæl: Ferdig forarbeid | 0 days | February 27 |
| 9 | **Del 2: Definering av prosjekt** | **3 days** | **February 27** |
| 10 | Definering av infrastruktur | 2 days | February 27 |
| 11 | Skissere oversikt for nødvendige komponenter | 2 days | February 28 |
| 12 | Milepæl: Prosjekt ferdig definer | 0 days | March 1 |
| 13 | **Del 3: Begrunnelse for valg** | **11 days** | **March 1** |
| 14 | Lage rammeverk for valg av kandidater | 6 days | March 1 |
| 15 | Velge kandidater utifra rammeverk | 6 days | March 8 |
| 16 | Milepæl: Rammeverk og valg ferdig | 0 days | March 15 |
| 17 | **Del 4: Oppsett av infrastruktur** | **21 days?** | **March 15** |
| 18 | Implementere komponenter | 11 days | March 15 |
| 19 | Implementere tester | 6 days | March 29 |
| 20 | Automatisere oppsett | 6 days | April 5 |
| 21 | Milepæl: Infrastruktur prototype ferdig | 1 day | April 12 |
| 22 | **Del 5: Sikkerhetsvurdering** | **6 days** | **April 12** |
| 23 | Dokumentere kjente svakheter | 6 days | April 12 |
| 24 | Milepæl: Sikkerhetsvurdering ferdig | 0 days | April 19 |
| 25 | **Del 6: Vurdering av alternativer** | **6 days** | **April 19** |
| 26 | Dokumentere outsourcing muligheter | 6 days | April 19 |
| 27 | Milepæl: Alternativer ferdig vurdert | 0 days | April 26 |
| 28 | **Del 7: Ferdigstilling av rapport** | **17 days** | **April 26** |
| 29 | Gjennomgang av rapport | 14 days | April 26 |
| 30 | Renskriving og ferdigstillelse | 4 days | May 15 |
| 31 | Milepæl: Rapport ferdig | 0 days | May 20 |

# K Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

# Prosjektavtale

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

ELLERA AS

(oppdragsgiver), og

JOSTEIN VOLE HAGE

BIRGER MYHRVOLD

DANIEL BRUUN (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 14.01.19 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.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): ERIK HJELMÅS

Oppdragsgivers kontaktperson (navn): JØRGEN ELLINGSEN

Student(er) (signatur): _____ dato 14.01.19

_____ dato 14.01.19

_____ dato 14.01.19

_____ dato _____

Oppdragsgiver (signatur): _____ dato 14.01.19

*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 _____

# L Møtereferater

Møtereferat fra gruppemøter, og møter med oppdragsgiver og veileder

| **Møtereferat** | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 2 | 16.1.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Forespørsel om tidligere relevant bacheloroppgaver?
  Det skal vi få tilsendt på mail.
- Klargjøre miljø på Openstack?
  Send mail til Eigil med estimert databehov, gruppe, navn og oppgave.
- Hva slags muligheter har vi for språk på rapporten?
  Anbefaler engelsk, men vi kan skrive på norsk. Vi får hjelp med rettskriving.
- Tips fra Hjelmås:
  Kan Cloud Native Application Bundle være relevant for oss?
- Informasjon om forprosjekt, og hva vi bør ha med.
  Unngå personlig pronomen. Rapporten skal være en objektiv anbefaling.

| **Møtereferat** | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 6 | 23.1.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Vi trenger veiledning til veier vi kan gå, og om det er ok som case for en bacheloroppgave.
  - Vi tenker å bruke kubernetes for infrastrukturen?
    Ikke bruk Kubernetes. For stort i skala for oppdragsgivers infrastruktur, og det er MYE å sette seg inn i. Spesielt med tanke på høy-nivå konfigurasjon.
  - Hva er største mangelen vår så langt? Hva skal til for å ha en god prosjektplan/rapport?
    Få til en ordentlig case for hva som skal gjøres. Det er uklart hva dere skal gjøre.
- Vi må snakke med oppdragsgiver og få ny inspirasjon/retning om case for oppgaven vår.

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 9 | 26.1.2019 | Møte med Ellera | Alle tilstede |

**Innhold:**

- Info om infrastrukturen til oppdragsgiver, og tips til hva vår løsning bør innehold, og kan inneholde
  - Virtuell server med nginx, php og CraftCMS
  - Tenk på skalering og redundans, samt plass i minnet
  - Tenk på stateless vs stateful, og lastbalansering ved flere servere
  - Bruker Bitbucket og er kjent med det. Bruk det for versjonskontroll, og mulige webhooks.
  - Bruk docker og begrunn valg mot configuration drift
    - Applikasjonener som har behov for en god infrastruktur er en billettjeneste og en nettsideplatformtjeneste. For applikasjon bruk CraftCMS.
    - Wordpress er et alternativ, men composer burde brukes, og wordpress bruker det ikke.
    - For database er det nok lurt at den er extern, ved bruk av Docker.
    - For praktisk start: Begynn med en Craft-server og en database. Bygg derfra. Eksempel på ekstra funksjonalitet kan være en worker som fungerer som en logg-aggregator.

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 10 | 6.2.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Status for gruppen:
  - Prøver ut teknologier vi trenger for å starte en grunnleggende infrastruktur
  - Docker containere satt opp for webserver og Redis, ved bruk av Puppet og med Docker Compose. Også prøvd ut Docker Swarm.
  - Pågående: Ansible med Docker
  - Prometheus er også blitt satt opp, og fungerer
- Spørsmål: Hvordan gjøres rapportering av kode i rapporten?
  Vi bør ha deler av kode/konfig i rapporten. For å vise oppsett bør vi også ha hele filer i rapporten.
- For å forbedre"infrastruktur tenk arbeidsflyt Ellera har idag.
- Rulle ut applikasjon på en samling VMer.
- Hør med arbeidsgiver om hva de ønsker.
- Hva er nytten for Ellera, hva gjør de idag og hvordan kan vi gjøre ting bedre?
- Valg av CMS. Henvisning til relevant tidligere rapport.
- Bortkastet tid å lære seg alt mulig forskjellig. Velg noe.

| **Møtereferat** | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 11 | 20.2.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Status:
  - Timeplan siden forrige gang halvvert på grunn av Campus nettverk deleksamen
  - Akkurat nå skrives rapport. Grunnleggende teori vi har bruk for.
- For inspirasjon til oppsett og innhold se på tidligere rapporter.
- Spesifikke rapporter: Mobilautentiseringsfaktor, Sikkerhetsanalyse av Jenseg og Evry oppgaven. Tre sikkerhetsoppgaver som også kan ses på.
- Rapporten må være to-the-point, gode begrunnelser og nøyaktig.
- Skriv godt og være tydelig på hva vi skal få til. Gjør vurderinger underveis.
- Førsteutkast sendes inn til påske.
- Vis hvordan dere tar ansvar innenfor åpne rammer i oppgaven

| **Møtereferat** | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 12 | 27.2.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Gruppestatus:
  - Fra 10. mars og utover blir det testing og videreutvikling av infrastruktur
  - Vi bør spesifisere Gantt-diagram med mer konkrete oppgaver
  - Ta en titt tilbake på oppgavebeskrivelsen og avgrensning når en lager Gantt-diagram
  - Pass på samsvar i oppgaven. Det er viktig.
  - Figurer og font. Lær dere et program for figurer og bruk det.
  - Oppgavebeskrivelse er ok, men kan beskrives mer mtp komponenter
  - Tenk på avgrensning til Craft og andre avgrensninger
  - Kan være en fordel å bruke Gitlab for windows-server for eksempel

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 13 | 6.3.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

- Gruppestatus:
  - Fortsatt usikkerhet rundt valg av teknologier, og hvordan dette gjøres på en god måte i forhold til rapport
  - Blir vår vurdering av CMS litt for nøye? Bør vi ha den som vedlegg?
- Inspirasjon for valg av CMS er tatt fra tidligere rapport, men hvordan bruke dette på en god måte?
  Dersom noe skal måles, definer hva som menes og henvis til standard uten egne meninger.
- Rapporten bør referere til metodikk som benyttes, men ikke overdriv metodikken. Forankre vitenskapelig etterpå.
- Med tanke på sikkerhet. Overfladisk og enkelt holder ikke. Finn en standard og sjekkliste i ISO, som kan referes til ved måling av sikkerhet.
- Sjekk SNYK for vurdering av sikkerheten i infrastruktur og utrulling av dette.
- Hva vektlegger Ellera? Forenkling og standardisering?
- Finn en standardisert måte å gjøre ting på.
- Ved vurdering av teknologi, tenk tilgang på kompetanse. Henvis til firmaer hvor en kan hente kompetanse

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 15 | 18.3.2019 | Møte med Ellera | Alle tilstede |

**Innhold:**

- Forklaring av tenkt design fra Ellera:
  - To Virtual private servers i docker swarm
  - Labels i forhold til production og development
  - Portainer.io for GUI ved sjekk og administrering av images, samt webhooks
  - Ellera gleder seg til å høre tilbakemeldinger og nye tenker

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 16 | 13.3.2019 | Møte med veileder | Daniel Bruun |

**Innhold:**

- Gruppestatus
  - Problemer med hva rapporten skal inneholde
  - Daniel er på jobbintervju i dag
- Spørsmål:
  - Oppdragsgiver ønsker at vi skal bruke MySQL og Redis og derfor velger vi det?
  - Skal vi skrive om selve implementeringen?
  - Bør vi demonstrere hvordan Ansible/Puppet fungerer med Kodesnutter o.l
  - Er det nok at vi kun gjør vurdering av CMS, Hvilket CMS vi skal benytte og/eller om vi bør benytte CMS i det heletatt?
  - Er det nødvendig å vurdere Pipeline tool eller er det nok av vi gjør en vurdering av hvilket tool vi bør bruke med tanke på Caset vårt
  - Bør vi ha med en vurdering av Cloud Vendors som f.eks AWS osv og vurdering av hvorfor en overgang til en slik tjeneste burde gjøres
- Direkte mot avgrensning av oppgaven. Avklar med Ellera
- Fokuser på kjernen i prosjektet. Det blir fort overfladisk om det ikke gjøres
- Vurder om utrulling av servere kan automatisere
- Avklar handlingsrom
- Hva er best practice for Elleras tjenester
- Skriv om design
- Bruk tidligere rapport for inspirasjon til implementering
- Om kostnadsvurdering skal gjøres, sjekk Troy Hunt
- Få på plass en løsning av et produkt, deretter skriv mer rapport.

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 17 | 18.3.2019 | Møte med Ellera | Alle tilstede |

**Innhold:**

- Ny diskusjon om mulige ønsker fra Ellera
  - Containerization
  - Redis, MySQL og eventuelt en Queuer
  - CI/CD pipeline med git-repository
  - Applikasjonscontainer med Nginx og PHP
    - Til info; Network latency mellom containere er grunnen til at de skal være i samme container. Best practice er separat
  - Off-site backup, med lagring hver natt
  - Logg aggregering hadde vært flott
  - Mulige utfordringer:
    - Databasemigrasjoner i forhold til endringer ved for eksempel oppdateringer av webapplikasjon under drift.

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 18 | 3.4.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

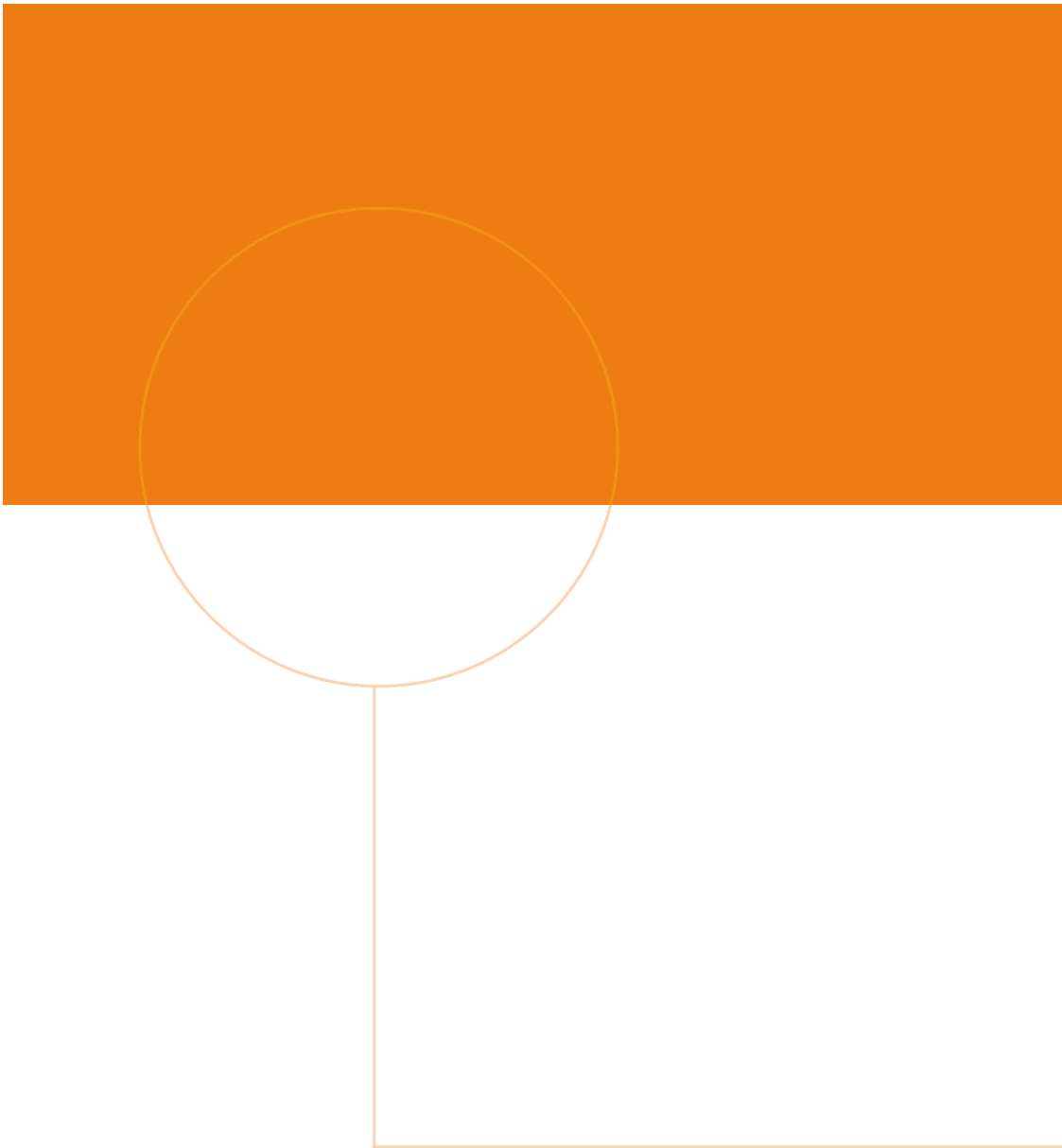
- Tips til rapportinnhold:
  - Patching, best practice. Holde konfigurasjon og oppdatering til dagens versjon
  - Firewall, Kan være lurt å ha flere lag
  - Sikkerhetsvurdering for logging o.l og finne løsninger
  - Produksjonsmiljøer
  - HTTPS for sikker kommunikasjon
  - Finne og refere til kilder brukt
  - Push vs pull
  - Service Discovery

| Møtereferat | | | |
|---|---|---|---|
| **Møtenummer** | **Dato** | **Type** | **Fravær** |
| 19 | 24.4.2019 | Møte med veileder | Alle tilstede |

**Innhold:**

Gjenstående tid og prioriteringer:

- Lag en leverbar rapport
- Språk i nåværende rapport er for muntlig
- Faglige vurderinger er overfladiske. Gå i dybden
- Hjelmås er borte 14.-15. mai pga samling
- Tidsfrist: Levere 2. utkast for rapport før 10. mai



NTNU

Kunnskap for en bedre verden