Bacheloroppgave

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

Morten Bjerke
Hamse A. Hashi
Fredrik Røstad

Konteinertjenester for OpenStack

Bacheloroppgave i IT-drift og informasjonssikkerhet
Veileder: Erik Hjelmås
Mai 2019

NTNU
Kunnskap for en bedre verden

Morten Bjerke
Hamse A. Hashi
Fredrik Røstad

Konteinertjenester for OpenStack

Bacheloroppgave i IT-drift og informasjonssikkerhet
Veileder: Erik Hjelmås
Mai 2019

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

NTNU
Norwegian University of
Science and Technology

Konteinertjenester for OpenStack

Forfattere
Morten Bjerke
Hamse A. Hashi
Fredrik Røstad

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

20.05.2019

Veileder Erik Hjelmås

Sammendrag av Bacheloroppgaven

Tittel:	**Konteinertjenester for OpenStack**
Dato:	20.05.2019
Deltakere:	Morten Bjerke Hamse A. Hashi Fredrik Røstad
Veiledere:	Erik Hjelmås
Oppdragsgiver:	Eigil Obrestad and Lars Erik Pedersen from the deparment for information security and communication technology at the Norwegian University of Science and Technology
Kontaktperson:	Erik Hjelmås, erik.hjelmas@ntnu.no, 61135220
Nøkkelord:	Norway, Norsk
Antall sider:	64
Antall vedlegg:	7
Tilgjengelighet:	Åpen

Sammendrag:

NTNUs institutt for informasjonssikkerhet og kommunikasjonsteknologi tar i bruk OpenStack som privat skyløsning til studenter og ansatte. Tjenestene som tilbys kjører nå i virtuelle maskiner. NTNU ser nå mot å bruke konteinere for å kjøre sine OpenStack-tjenester. Dette fordi oppgraderinger av OpenStack er tidkrevende og det skaper unødvendig mye arbeid for administratorene. OpenStack slipper ut nye versjoner to ganger i året. Med disse versjonene kommer det også nye avhengigheter, blant annet til operativsystem. Konteinere kan forenkle denne prosessen ved hjelp av sin abstraksjon. Konteinere kan bygges og styres selv, eller bygges ved hjelp av images fra prosjekter som arbeider med å slippe oppdaterte OpenStack images og verktøy for disse. OpenStack-Kolla er et prosjekt som slipper produksjonsklare versjoner av OpenStack sine tjenester som Docker-images for å prøve å bidra til forenkling av arbeidsrutiner rundt OpenStack. Kolla-Ansible og OpenStack-Helm er verktøy som kan brukes for å konfigurere, rulle ut og administrere disse konteinerne.

Summary of Graduate Project

Title:	**Konteinertjenester for OpenStack**
Date:	20.05.2019
Authors:	Morten Bjerke Hamse A. Hashi Fredrik Røstad
Supervisor:	Erik Hjelmås
Employer:	Eigil Obrestad and Lars Erik Pedersen from the deparment for information security and communication technology at the Norwegian University of Science and Technology
Contact Person:	Erik Hjelmås, erik.hjelmas@ntnu.no, 61135220
Keywords:	OpenStack, Kolla, Konteinere, SkyHiGh
Pages:	64
Attachments:	7
Availability:	Open

Abstract:

NTNU's department for information security and communication technology is using OpenStack's private cloud solution for students and employees. Currently the services offered are all running on virtual machines. NTNU is looking to change these services from virtual machines into containers. This is because upgrading OpenStack is time-consuming, and that is causing unnecessary amounts of work for the administrators. OpenStack releases two updates each year. With each version comes updated requirements, including to the operating system. The abstraction of containers can help with this. Container images can be built and managed internally, or they can be acquired from projects releasing public OpenStack images. OpenStack-Kolla is a project whose mission is to provide production ready container images for OpenStack. Kolla-Ansible and OpenStack-Helm are tools that can be used to help configure, deploy and manage these containers.

Preface

Vi ønsker å takke vår veileder Erik Hjelmås for god hjelp og veiledning. Vi vil også takke våre oppdragsgivere Eigil Obrestad og Lars Erik Pedersen, som har vært åpen for hjelp og spørsmål, samt vært tilgjengelige og responsive gjennom hele prosjektet.

Innhold

Preface . . . iii
Innhold . . . iv
Figurer . . . viii
Tabeller . . . ix
List of Listings . . . x
Forkortelser . . . xi
Terminologi . . . xii
1 Innledning . . . 1
1.1 Bakgrunn . . . 1
1.2 Oppgaven . . . 1
1.2.1 Problemområdet . . . 1
1.2.2 Oppgavebeskrivelse . . . 2
1.2.3 Rammer . . . 3
1.2.4 Avgrensinger . . . 3
1.3 Formål . . . 3
1.4 Målgruppe . . . 4
1.5 Prosjektgruppens bakgrunn . . . 4
1.6 Roller . . . 5
1.7 Rapportens innhold . . . 6
1.7.1 Rapportstruktur . . . 6
2 Forarbeid, inndeling og kravspesifikasjon . . . 7
2.1 Oppstartsfase . . . 7
2.1.1 Hovedinndeling av prosjektoppgave . . . 7
2.2 Kravspesifikasjon . . . 8
2.2.1 Use cases . . . 10
3 Utviklingsprosess . . . 13
3.1 Utviklingsmetode . . . 13
3.2 Begrunnelse for valg av metodikk . . . 13
3.3 Gjennomføring av metodikk . . . 14
3.4 Detaljert bruk av Scrum . . . 14
3.4.1 Sprint planning . . . 14
3.4.2 Daily sprint . . . 14
3.4.3 Sprint review . . . 14
3.4.4 Møte med veileder og oppdragsgiver . . . 15
3.4.5 Scrum board . . . 15

4 Teori **16**
4.1 OpenStack 16
4.1.1 Arkitektur 16
4.1.2 Komponenter 17
4.1.3 Støtteverktøy 19
4.2 SkyHiGh 19
4.2.1 Arkitektur 19
4.3 OpenStack-Kolla 20
4.3.1 Images 20
4.3.2 Kolla-Ansible 20
4.4 OpenStack-Helm 21
4.5 Konteinerteknologi 22
5 Initielt utvalg av teknologier **24**
5.1 Kriterier 24
5.1.1 Utforskning og metoder 24
5.2 Kandidater 25
5.2.1 OpenStack Kolla-Ansible 25
5.2.2 OpenStack Helm 25
5.2.3 Selvbygde konteinere 26
5.2.4 Andre kandidater 27
5.2.5 Resultat for initielt utvalg 30
6 Implementering av utvalgte kandidater **31**
6.1 Oppsett av Kolla-Ansible 31
6.1.1 Installasjon og konfigurasjon 31
6.1.2 Driftsrutiner 33
6.1.3 Feilsøking 34
6.2 Oppsett av OpenStack-Helm 35
6.2.1 Installasjon og konfigurasjon 35
6.2.2 Driftsrutiner 38
6.2.3 Feilsøking 39
6.3 Oppsett av selvbygde konteinere 39
6.3.1 Installasjon og konfigurasjon 39
6.3.2 Driftsrutiner 40
6.3.3 Feilsøking 41
7 Detaljert vurdering av implementerte teknologier **42**
7.1 Vurderingskriterier 42
7.1.1 Effektivitet 43
7.1.2 Kompleksitet 43
7.1.3 Popularitet og trender 43
7.1.4 Sikkerhet 44

7.1.5 Oppsummering . . . 45
7.1.6 Sammenligning . . . 45
7.2 Kolla-Ansible . . . 45
7.2.1 Effektivitet . . . 45
7.2.2 Kompleksitet . . . 46
7.2.3 Trend / popularitet i bransjen . . . 47
7.2.4 Sikkerhet . . . 47
7.2.5 Oppsummering . . . 47
7.3 Openstack Helm . . . 48
7.3.1 Effektivitet . . . 48
7.3.2 Kompleksitet . . . 49
7.3.3 Trend / popularitet i bransjen . . . 50
7.3.4 Sikkerhet . . . 50
7.3.5 Oppsummering . . . 51
7.4 Selvbygde konteinere . . . 51
7.4.1 Effektivitet . . . 51
7.4.2 Kompleksitet . . . 52
7.4.3 Trend / popularitet i bransjen . . . 52
7.4.4 Sikkerhet . . . 52
7.4.5 Oppsummering . . . 53
7.5 Sammenligning . . . 53
7.5.1 Effektivitet . . . 53
7.5.2 Kompleksitet . . . 54
7.5.3 Popularitet og trender . . . 54
7.5.4 Sikkerhet . . . 54
7.5.5 Kvantitativ vurdering . . . 55
8 Diskusjon og konklusjon . . . 56
8.1 Diskusjon . . . 56
8.1.1 Resultater . . . 57
8.2 Videre arbeid . . . 57
8.3 Evaluering av gruppearbeidet . . . 58
8.4 Konklusjon . . . 58
Bibliografi . . . 59
A Feilsøking av OpenStack Kolla-Ansible . . . 65
A.1 ansible -i multinode.ini -m ping all . . . 65
A.2 kolla-ansible -i multinode.ini bootstrap-servers . . . 65
A.3 kolla-ansible -i multinode prechecks . . . 66
A.4 kolla-ansible -i multinode deploy . . . 69
A.5 Kolla-ansible -i multinode deploy med ekstern konteiner database . . . 69
A.6 kolla-ansible -i multinode deploy med ekstern ikke-konteiner database . . . 70

B Feilsøking av OpenStack-Helm **71**
B.1 AllInOne 71
B.2 Multinode 72
B.3 Eget oppsett av Kubernetes 77
C Kode **78**
C.1 Puppet-kode for oppsett av Kolla-Ansible 78
D Møtereferat **82**
E Timelister **96**
F Opprinnelig oppgavebeskrivelse **100**
G Prosjektplan **102**

Figurer

1 Organisasjonskart for prosjektgruppen 5
2 Use case diagram 10
3 Gruppens Trellotavle 15
4 Komponentene i OpenStack 17
5 Kolla-ansible multinode arkitektur 20
6 Helm artefakter 21
7 Konteinerarkitektur sammenlignet med arkitektur for virtuelle maskiner . 22
8 Det OpenStack-brukere benytter av verktøyer for PaaS og konteinere . . . 26
9 Mirantis sin MCP kombinert med k8s kluster 28
10 Simplifisert arkiteturskisse av Airship 29
11 Arkitektur i testdeployment av Kolla-Ansible 31
12 Arkitektur i testdeployment av OpenStack-Helm 35
13 Arkitektur i testdeployment av selvbygde konteinere 39

Tabeller

1 Use case 1 10
2 Use case 2 11
3 Use case 3 11
4 Use case 4 12
5 Kvantitativ vurdering av implementerte løsninger 55

List of Listings

1 Eksempelkode fra manager.pp for oppsett av Kolla-Ansible 33
2 Eksempelskript som brukes for oppsett av tjenester i OpenStack-Helm . . . 36
3 Kodeeksempel for inventory-fil i OpenStack-Helm 37
4 Eksempelkode fra Helm environment-fil 37
5 Utkast av docker-compose 40

Forkortelser

NTNU - Norges teknisk-naturvitenskapelige universitet.
IIK - Institutt for informasjonssikkerhet og kommunikasjonsteknologi.
IaaS - Infrastructure as a service
VM - Virtuelle maskiner
K8s - Kubernetes
PoC - Proof of concept
API - Application Programming Interface
CMS - Configuration management system
IRC - Internet Relay Chat

Terminologi

SkyHiGh - Den private skyløsningen ved NTNU i Gjøvik.
Mikrotjeneste - på engelsk Microservice- En betegnelse for en tjeneste eller applikasjon som kjører i konteiner.
Monolittisk tjeneste - på engelsk Monolithic service/ application. En tjeneste eller applikasjon som kjører i en virtuell maskin.
Open source - Det er et utbredt begrep innen IT- miljøet for prosjekter der kildekoden ligger ute og andre kan nytte av det, i tillegg er det åpent for å bidra til disse prosjektene.
Konteinerorkestrering - på engelsk Container Orchestration- Organiseringsprosessene for å imøtekomme et gitt design for tjenestene eller tilfredstillende tilstand.
Token - En kode som blir tildelt etter at en applikasjon eller bruker har blitt autorisert. Token brukes til å utføre tillatte handlinger mot en API.
Devops - Nyere trend/kultur hvor utviklere og driftere jobber og samarbeider tettere enn før eller hvor arbeidere gjør begge deler.
Idempotent - En handling som kan gjøres utallige ganger uten at det vil påvirke allerede fungerende resultat, men vil påvirke ikke-fungerende resultat.
Configuration drift - En tilstand som oppstår når maskiner ikke konfigureres konsistent og systematisk, for eksempel når de konfigureres ad hoc.
konteinerteknologi og konteinerløsning - brukes om hverandre.
Prosjektgruppen, gruppen, vi - brukes om hverandre.
Kubernetes rolling update - En Kubernetes funksjonalitet. Oppdatering av applikasjoner skal skje smertefritt og uten nedetid, samt tilby tilbakerulling til forrige stabile versjon ved behov. [1]
fossefall-modell - En statisk utviklingsmodell.
immutable - Enhet som blir slettet og erstattet ved oppdatering.

1 Innledning

1.1 Bakgrunn

Skyløsninger har vokst sterkt fram innen IT-bransjen i de siste årene[2]. Mange bruker kommersielle skyplattformer som Amazon Web Services, Google Cloud og Microsoft Azure. Andre velger å sette opp private skyløsninger, hvor alt foregår innad i eget selskap. Ved NTNU er det satt opp en implementasjon av open source-skyplattformen OpenStack. Prosjektet er kalt SkyHiGh og ble opprettet i 2012. Plattformen brukes av både studenter og lærere ved NTNU for diverse formål. Dette kan være behov for å arbeide med virtuelle maskiner, sette opp egne infrastrukturer, eller sette opp test-miljøer og lignende enten for utdanning eller i forskningsøyemed. SkyHiGh driftes av NTNU IIK. Kodebasen til SkyHiGh består for det meste av Puppet-kode. De forskjellige tjenestene i OpenStack kjøres per dags dato i virtuelle maskiner, fordelt på flere noder. Oppdateringer og endringer innebærer mer arbeid og kompleksitet enn det IIK ønsker.

Konteinertjenester er en teknologi som har blitt meget populær i de siste årene[3], og er kjent for å kunne forenkle arbeidsflyt og tilrettelegge for automatisering. IIK ønsker å få utredet muligheten til å bruke konteinertjenester til å styre kontrollplanet for OpenStack for å forbedre driftsrutinene. IIK foreslo dermed en oppgave til bachelorprosjekt for IT-studenter ved NTNU i Gjøvik. Denne oppgaven ønsket prosjektgruppa å jobbe med og fikk den tildelt for vårsemesteret 2019.

1.2 Oppgaven

1.2.1 Problemområdet

Med skytjenestenes entre inn i IT-verdenen har tjenester begynt å gradvis flyttes over til skyen. Skyer er ikke lenger noe kun Apple, Google, og et fåtall nisje-selskaper driver med. Kommersielle applikasjoner som er direkte tilgjengelig for sluttbrukere så vel som interne tjenester, slik som interne lønningssystemer, kjøres fra skyen. Dette krever endringer og nye rutiner da det gjelder både utvikling og vedlikehold. Konsepter som DevOps har vokst fram og blitt store, slik at utviklerne og drifterne begge er med på hele syklusen for en software, fra konseptuell design til langtidsstøtte og support. Driftingen foregår ofte innad i bedriften og vil kreve folk som kan rutinene for det. Dette kan innebære oppsett, videreutvikling, skalering, feilsøking, programvareoppdateringer og support. Tjenestene kan kjøres på lokale servere eller på offentlige skyer som AWS. Området er ikke foruten utfordringer og grunnet hyppigheten av gjennombrudd i teknologien skjer det ofte endringer.

For tiden har det vært store bevegelser i overgangen til å levere tjenester ved bruk av konteinere istedenfor i virtuelle maskiner, og IIK ønsker nå det samme. I den eksisterende implementasjonen av SkyHiGh kjører tjenestene i virtuelle maskiner. Med et slikt oppsett har det seg slik at operativsystemer på de virtulle maskinene må oppdateres individuelt. I

praksis vil dette innebære å lage nye virtuelle maskiner, hvor tjenestene maskinen kjører vil måtte bli utrullet på nytt. Dette er en tidkrevende rutine som IIK gjerne skulle vært foruten.

Prosjektet OpenStack slipper ut ny versjon to ganger i året og kun til støttede Unix distribusjoner, det vil si at man blir nødt til å oppgradere distribusjonen for å få ligge stødig på nyeste versjoner av OpenStack. Dette er en tidkrevende, manuell og tungvint prosess for oppdragsgiveren. Infrastrukturen IIK bruker er allerede satt opp på slik måte som en typisk konteiner-infrastruktur, hvor hver virtuell maskin er dedikert til en tjeneste. I tillegg kjører disse virtuelle maskinene også en del unødvendige prosesser som tjenestene ikke avhenger av. Dette kan lede til noe unødvendig ressursbruk. Unødvendige prosesser kan også føre til configuration drift.

Disse faktorene ved dagens oppsett fører til vanskeligheter når det kommer til generell drifting og da spesielt oppgraderinger av tjenester. Det hadde vært ønskelig om dette var en enklere prosess som krevde mindre tid og manuelt arbeid. Gruppen håper å kunne forenkle ovennevnte utfordringer for IIK ved å levere et forslag til bruk av konteinerteknologi.

1.2.2 Oppgavebeskrivelse

Prosjektgruppen har fått i oppgave å gjøre en utredning av konteiner-teknologi for NTNUs private skyløsning SkyHiGh. Denne tjenesten brukes av NTNU sine ansatte og studenter i forbindelse med forsking og opplæring. SkyHiGhs tjenester kjører per idag i virtuelle maskiner, noe vår oppdragsgiver ønsker å finne alternative arkitekturløsninger for i konteinere. Oppgaven omfatter å gjøre en utredning av en eller flere typer konteinerløsninger. Disse utredningene gjøres for å kartlegge hvilken tilnærmingsmetode for bruk av konteinere som egner seg best til oppdragsgivers formål og problemområdet. Kriteriene blir satt for å finne en løsning som forenkler arbeidsrutene for IIK, og for å finne en teknologi som er framtidsrettet og vil støttes over tid. Oppgaven består av tre hoveddeler:

- En utredning og utvalg av best egnet konteinerteknologi.
- Implementasjon og vurdering av utvalgte kandidater.
- En konklusjon og anbefaling av best egnet løsning.

Oppgaven omfatter å gjøre en vurdering av utvalg av konteinere enten fra OpenStack-prosjektet Kolla eller selvbygde konteinere, kombinert med verktøy for bygging og styring av disse. Da gruppen referer til en løsning menes kombinasjonen av en kilde for konteiner-images med verktøyene som brukes for bygging og styring. Det vil først gjøres en grunnleggende vurdering av alle relevante kandidater gruppen finner i sin forskning av området. De aktuelle kandidatene vil bli satt opp i et testmiljø, der de kan prøves ut i detalj. Her vil det bli satt opp diverse proof of concepts som vil vises til oppdragsgiver gjennom prosjektperioden. Disse forslagene vil inkludere driftsrutiner, kodeoppsett, oppgraderingsprosesser og annen relevant informasjon for IIK som vil bli beskrevet senere i rapporten. Dette gjøres for at IIK har best mulig forutsetninger om det skulle være ønskelig å bruke konseptetoppsettet etter avsluttet prosjekt. Det vil i tillegg settes kriterier

som er relevante for IIK og gjøres en detaljert vurdering av kandidatene opp mot disse. Til slutt vil gruppen komme med en akademisk anbefaling til IIK om bruk av konteinertjenester.

1.2.3 Rammer

Den viktigste rammen for oppgaven er tid. Hvis en løsning blir ansett som ikke rimelig å få laget et proof of concept av innen utgangen av April 2019 vil den bli nedprioritert, men vil fortsatt dokumenteres og nevnes i rapporten.

1.2.4 Avgrensinger

Fokuset for denne oppgaven er først og fremst en faglig vurdering av hvilken konteinerteknologi som egner seg best for NTNUs IIK. Ettersom konteinere har vokst enormt finnes det mange forskjellige teknologier og implementasjoner for konteinerisering av OpenStack. I denne oppgaven vil kun konteinere bygget av OpenStack Kolla eller selvbygde Docker-konteinere og styringsverktøy for disse bli vurdert. Det vil heller ikke være et hovedfokus å implementere konteinerorkestrering for selvbygde Docker-konteinere, slik som Docker Swarm eller Kubernetes. Men dersom tidsrammen tillater det skal gruppen forsøke å implementere dette.

Det foreslåtte proof of concept'et skal kun være en isolert demonstrasjon, og ikke noe som skal ut i produksjonsmiljø. Konseptoppsettet er begrenset til utrulling av autentiseringstjenesten Keystone på grunn av den begrensede tiden gruppen har til rådighet. Dette vil enten bli satt opp på virtuelle maskiner i SkyLow eller satt opp med gruppens ressurser i SkyHiGh, eller i SkyLow som er et speilet testmiljø for SkyHiGh. Løsningen vil kun virke som et forslag til hvordan tjenester kan kjøre i konteinere, ikke en endelig løsning. Gruppen selv vil ikke ha noe innvirkning på om verken utredningen eller implementasjonen skal tas i bruk av IIK etter at oppgaven er levert.

1.3 Formål

Oppgaven skal ha som hovedformål å effektivisere driftingen av SkyHiGh sin infrastruktur. Dette innebærer å forenkle og effektivisere oppgradering av de forskjellige komponentene, i tillegg til å sørge for god skalerbarhet, redundans ved bruk av flere konteinere og god dokumentasjon slik at det blir lettere å gjenskape eller sette opp løsningsforslag. Dette for å kunne hjelpe til med å redusere arbeidsmengde og sørge for gode rutiner for teamet som drifter SkyHiGh, slik at de kan bruke mindre tid på driftingen av tjenesten, og kan prioritere andre oppgaver.

Oppgaven skal være en utredning på om det vil være hensiktsmessig å kjøre SkyHiGh i konteinere og i så fall hvilken løsning vil være den beste. Den skal finne de forskjellige alternativene for konteinerløsningen, vurdere disse, og komme med en anbefaling til IIK. Demonstrasjonen av modellen og erfaringene som blir gjort i å sette denne opp skal ha som formål å muliggjøre videre bruk av modellen for IIK, for å bidra til at NTNU i Gjøvik har en god og moderne skyløsning som virker attraktiv for potensielle søkere og som er god i bruk for studenter og ansatte som bruker tjenesten.

1.4 Målgruppe

Primær målgruppe for prosjektarbeidet er NTNUs IIK, representert ved oppdragsgiverne Lars Erik Pedersen og Eigil Obrestad. Lars Erik og Eigil er ansvarlige for driften av SkyHiGh, og det er de som må feilsøke, gjøre endringer og oppdateringer ved behov. De ønsker redusert arbeidsbehov for drifting og kanskje spesielt oppdateringer av SkyHiGh.

En sekundær målgruppe er brukerne av SkyHiGh, som vil ha så god ytelse og lav nedetid på tjenesten som mulig.

En tredje målgruppe er de som har interesse for å lese rapporten. Det vil si veileder og sensor, oppdragsgiver, i tillegg til muligens andre grupper som kan ta inspirasjon fra oppgaven i fremtiden.

Siden SkyHiGh er en av de større prosjektene relatert til driftslinjen ved NTNU i Gjøvik vil NTNU som organisasjon også kunne få en positiv effekt i form av omdømme ved å holde tjenesten moderne.

Legges eventuell fungerende løsning ut offentlig på Github eller lignende vil også andre driftere som skal gjennom samme prosess ved å konteinerisere levering av OpenStack tjenester få bruk for publisert kode, oppsett og dokumentasjon.

1.5 Prosjektgruppens bakgrunn

Prosjektgruppen består av studenter fra studiet IT-drift og informasjonssikkerhet ved NTNU i Gjøvik. Gruppemedlemmene har samarbeidet tidligere på andre prosjekter i regi av universitetet og har opplevd solide resultater sammen. I tillegg har gruppen opplevd god forståelse for hverandre som gir god kommunikasjon og arbeids moral.

Medlemmene på gruppen har hatt et studieløp sammensatt av koding, sikkerhet, nettverk, og drifting. Vi har alle erfaring med bruken av OpenStack via SkyHiGh. Et relevant fag har vært IMT3003, Drift av Tjenestearkitekturer, der gruppen ble introdusert til teknologier som Docker, og lærte seg grunnleggende drift av servere. I tillegg til dette har hele gruppen hatt valgfaget IMT3005, Infrastructure as Code. Dette er et fag som også er relevant for selve oppgaven da det innebærer både praktisk og teoretisk arbeid innen områder som skyplattformer og tjenestedrift. Både fagene IMT3005 og IMT3003 har gitt gruppen en god basis forståelse av relevante teknologier innen drift som serverstyring, konfigurasjonsstyring, automatisering, skyløsninger og konteinerteknologier. Gruppen lærte også om relevante konsepter som snowflake servers, configuration drift, redundans, backup, og idempotent.

De tre medlemmene i gruppen er:

Morten Bjerke, 23 år. Bachelorstudent IT-drift og informasjonssikkerhet. Unike fag: SMF2051 - Ledelse med Arbeidslivsjus, IMT3501 - Programvaresikkerhet.

Hamse Abdi Hashi, 28 år. Bachelorstudent IT-drift og informasjonssikkerhet. Unike fag:

IMT2681 - Cloud technologies, IMT2291 - WWW-teknologi.

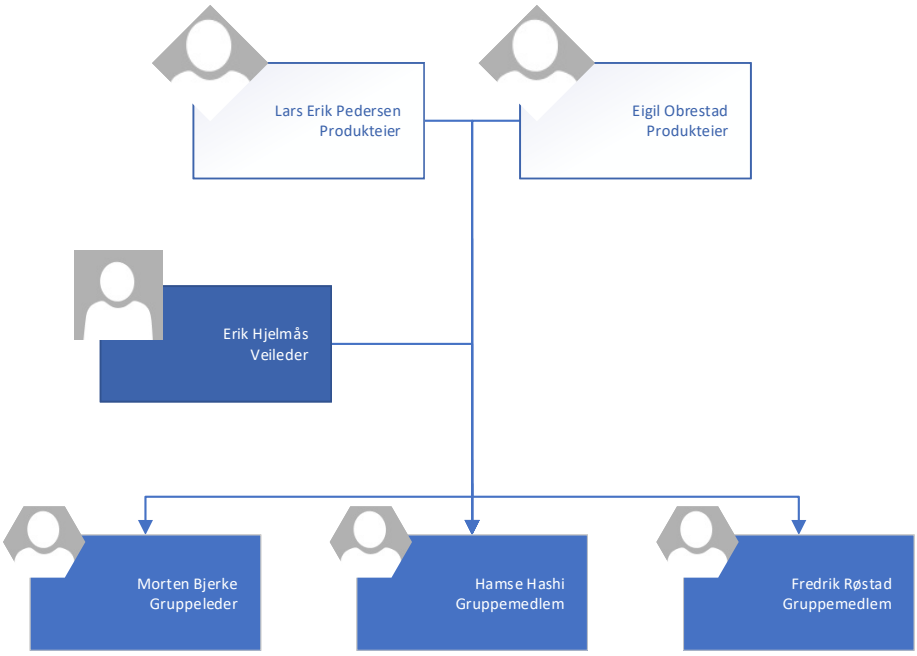
Fredrik Røstad, 28 år. Bachelorstudent IT-drift og informasjonssikkerhet. Unike fag: SMF1042 - Økonomistyring, IMT3691 - Campusnettverk og internettarkitektur.

1.6 Roller

Oppdragsgiveren for prosjektet er NTNU IIK v/ Eigil Obrestad som er universitetslektor og Lars Erik Pedersen som er overingeniør, begge ansatt ved IIK. De jobber begge med drifting av SkyHiGh, hvor det stadig gjøres oppdateringer og vedlikehold. Begge er også tidligere studenter ved HiG/NTNU i Gjøvik som har blitt ansatt ved universitetet etter utdannelsen.

Veilederen for prosjektgruppa er Erik Hjelmås, førsteamanuensis ved IIK. Han er også foreleser for fagene IMT3005 Infrastructure as Code og IMT2282 Operativsystemer.

Prosjektgruppens leder er Morten Bjerke og Scrum master er Hamse Hashi. Innad i gruppen vil oppgavene bli fordelt likt. Ingen anser seg selv som spesialist på noen områder i forhold til andre, det vil da bli variert arbeid med tekniske utfordringer, feilsøking, forskning og rapportskriving. Det vil forsøkes å rullere på arbeid slik at alle for jobbet med det de ønsker samt ikke gå lei av mindre foretrukkede oppgaver.



Figur 1: Organisasjonskart for prosjektgruppen

1.7 Rapportens innhold

Det er forsøkt å bruke norske ord og uttrykk der det er gode oversettelser for ordene, men noen vil bli beholdt på engelsk. Da rapporten referer til prosjektgruppen/gruppa/gruppen menes gruppa som har skrevet rapporten, da rapporten refereres til menes denne rapporten om bacheloroppgaven, og da oppgaven refereres til menes dette bachelorprosjektet om konteinertjenester for SkyHiGh. Terminologi kan finnes i appendix. Kildehenvisninger vil være klikkbare og lede til bibliografien. Innholdsfortegnelse, liste over figurer, liste over tabeller, og liste over kodesnutter kan alle finnes på starten av oppgaven, og er klikkbare for å lede til elementet.

1.7.1 Rapportstruktur

Rapporten inneholder innledning, hoveddel, avslutning og appendix eller vedlegg. Totalt inneholder rapporten åtte kapitler. Innledningen inneholder informasjon om oppgaven og personer relevant for den. Kapittel to til fire er metode, kravspesifikasjon og teori. Hoveddelen er kapittel fem til sju som inneholder initial utvalg av kandidater, implementasjon av valgte kandidater og detaljert vurdering av implementerte teknologier. Rapporten avsluttes med en diskusjon av resultater og en konklusjon i kapittel åtte. I konklusjonen gir prosjektgruppa en akademisk anbefaling basert på utredningen gjort i oppgaven. Appendix inneholder kode, prosjektplan, møtereferater, timelister, kontrakt, og andre tilleggsdokumenter.

2 Forarbeid, inndeling og kravspesifikasjon

Dette kapittelet vil beskrive hvordan vi delte opp oppgaven for å få bedre oversikt over problemstillingen samt gjøre det lettere for å oss selv å løse oppgaven. Det vil bli kort beskrevet en gjennomgang av hvert stadie av prosjekt, inkludert oppstartsfasen for oppsummeringsvis. I tillegg beskriver vi kravspesifikasjonene vi har drøftet med oppdragsgiver og gir noen eksempler på use cases.

2.1 Oppstartsfase

Starten av prosjektet gikk ut på å etablere kontakt med veileder og oppdragsgiver. Det ble fort etablert kontakt med begge parter og vi fikk tidlig et godt innblikk i hva oppgaven gikk ut på og hva som var forventet av gruppen. Etter dette ble det klargjort hva et proof of concept skulle innebære og på hvilken måte arbeidsgiveren ville få det levert.

2.1.1 Hovedinndeling av prosjektoppgave

Inndelingen står hovedsakelig av seks deler (1+5), hvor forprosjektet er uavhengig forhold til selve oppgaven. Gruppen har da valgt å dele hoved oppgaven i fem biter.

Del 0: Forprosjekt

Dette var starten av prosjektet. Denne delen av oppgaven hadde ikke direkte tilknytning til den faktisk problemstillingen. Forprosjektet gikk hovedsakelig ut på å lage et solid fundament for å komme igang, arbeide, og fullføre selve oppgaven. Forprosjektet inneholder detaljerte beskrivelser av prosjektet, mål og rammer, tidsskjema og annen relevant informasjon. Prosjektplanen ble levert 1. Februar. Den ble da sett igjennom av veileder og godkjent slik at oppgaven kunne fortsette.

Del 1: Forskning av relevante teknologier

Det er på denne delen vi faktisk startet med selve oppgaven. Her gjøres det grundig forskning på hvilken teknologier som egner seg best, både av konteinere og styringsverktøy for disse. I oppgavebeskrivelsen ble det nevnt at OpenStack Kolla og selvbygde konteinere var relevante alternativer for bygging av konteiner-images. På dette punktet måtte vi finne de relevante løsningene både for konteinere og styringsverktøy for disse, og veie dem opp mot hverandre med en høynivå vurdering. Prosjektets tidsramme gir ikke nok tid til å utføre lavnivå vurdering av mange forskjellige teknologier, så vi må raskt velge hva som ble ansett som mest relevant teknologi for å løse problemstillingen.

Del 2: Initialutvalg av kandidater

Dette er den første av tre helhetlige utredningsdeler, og sett på som en vital del for prosjektoppgaven. I utredningsdelen er det veldig viktig å få sammenlignet flere aktuelle kandidater uten å bruke for lang tid. Det er nødvendig og ikke bare finne informasjon om teknologien men også om hvordan løsningene vil fungere i henhold til oppgavens problemstilling og ressurser. Hovedpoenget med initialutvalget er å finne kandidatene som egner seg for å konteinereisere NTNUs SkyHiGh slik at de kan testes videre i neste

fase eller utredningsdel, implementasjon og testing av utvalgte løsninger.

Del 3: Implementasjon og testing av utvalgte løsninger

Etter initialutvalget jobbes det på teknisk arbeid som implementasjon og testing. På dette punktet, opparbeider gruppen seg praktisk og teoretisk forståelse av kandidatene for å gjøre videre vurderinger av egnetheten.

Implementasjonen og testingen blir først utført på egne stacker i SkyHiGh. Hovedmålet er å få rullet ut en Keystone i et konteineremiljø basert på enten Kolla-konteinere eller selvbygde konteinere, og få verifisert at det virker. Om tiden tillater det kan de ferdige løsningskonseptene bli forsøkt implementert i SkyLow.

Del 4: Utredning og hovedvurdering av utvalgte kandidater

Etter gruppen har gjort implementasjon og testing av utvalgte kandidater, gjør gruppen dypdykk i vurdering av kandidatene, både hver for seg og satt opp mot hverandre. Denne delen av utredningen er svært viktig for å komme frem til en kandidat som løser oppgavens problemstilling. Det er også denne kandidaten gruppen kommer til å anbefale senere i konklusjonsdelen.

Del 5: Diskusjon, konklusjon og akademisk anbefaling

Etter at løsningskonseptene er vurdert vil det diskuteres hvilke kandidater som egner seg best, som leder til en konklusjon og akademisk anbefaling til IIK. Det vil også foretas diskusjon rundt prosjektets måloppnåelse, ting som kunne vært gjort annerledes, vurdering av om oppgavestrukturen gruppen valgte egnet seg, og eventuelle forslag til videre arbeid.

Del 6: Ferdigstilling av rapport

Rapporten vil bli jobbet på litt parallelt med alle tidligere deler av prosjektet, men vil ikke hatt fått hovedfokus før løsningen er ansett som ferdig eller det blir vurdert at praktisk arbeid må opphøre grunnet mangel på tid. Gruppen setter en frist på å ferdigstille løsningen innen 12. April. Rapporten vil uavhengig av resultat få hovedfokus etter satt dato. I rapporten vil alt fra start til slutt av prosjektet bli inkludert samt vedlegg og kilder.

2.2 Kravspesifikasjon

Kriteriene skal være med på å bedømme og avgjøre hvilken konteineriseringsløsning som egner seg best til å løse vår problemstilling, nemlig konteinerisering av SkyHiGh. Etter klar ønske fra oppdragsgiveren, vil gruppen vektlegge effektivitet høyest. De andre kriteriene er kompleksitet, trend og popularitet, og til slutt sikkerhet. Sikkerhet blir vektlagt noe lavere enn de andre kriteriene. Dette er ikke grunnet at sikkerhet er av mindre viktighet, men fordi løsningen gruppen plukker ut til slutt vil ta god nytte fra allerede eksisterende sikkerhetstiltak i SkyHiGh.

Effektivitet: Oppdragsgiveren ønsker å effektivisere driften av SkyHiGh, dette innebærer å effektivisere følgende:

- **Sette opp og ta ned OpenStack tjenestene eller tilhørende tjenester**. Her er det ønskelig at løsningen forenkler installasjonen og om nødvendig sletting av en eller flere OpenStack-tjenester.
- **Daglig drifting og konfigurasjonsendringer**. Dersom det er behov for å endre noe med tjenesten, for eksempel oppdatere en eller flere variabler, bør dette kunne gjøres på en effektiv og ryddig måte.
- **Oppdateringer og oppgraderinger**. Per idag er oppgraderinger av SkyHiGh en tungvint og tidkrevende prosess for IIK. Et av målene til gruppen er derfor å finne en konteinerløsning for SkyHiGh som er mer oversiktlig, bruker mindre tid å utføre og bidrar til forbedring av hele prosessen som helhet.

Kompleksitet: Kompleksiteten er et viktig aspekt for å bedømme og velge en løsning. Gruppen skal strebe etter å bedømme dette så objektivt så mulig og ikke bare om gruppen synes en løsning virker komplekst. Hovedgrunnen for dette kriteriet er at driftsavdelingen til SkyHiGh består av kun to personer per dags dato. Derfor er det viktig å ta hensyn til dette når gruppen skal velge ut en løsning. Kompleksitet vektlegges på bakgrunn av:

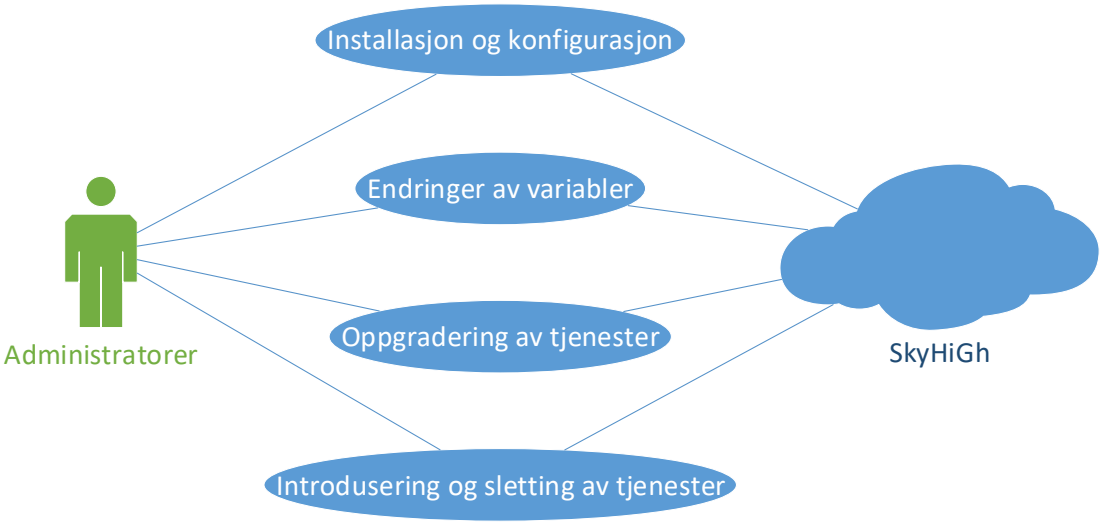
- Hvor komplekse de underliggende teknologiene er og hvor mye kunnskap som kreves for å kunne bruke løsningen.
- Hvor vanskelig og tungvint løsningen er å administrere.
- Hvor godt nettsamfunn eller hjelpeforumer som står bak løsningen eller prosjektet er. Dette er helt avgjørende da man alltid vil støte på hindringer eller problemer, derfor er det viktig å ha gode forumer og bugreports-sider å henvende seg til.

Trend og popularitet: Det andre selskaper bruker for å konteinerisere OpenStack gir en god indikasjon på hvilken løsninger og prosjekter gruppen bør utforske og vurdere som kandidater. Når et prosjekt har god popularitet og stort community bak seg, vil det stort sett gi flere forumer, bug tracks, jevn og god support og dokumentasjon til brukerne. Dette er viktig for påliteligheten og varigheten til prosjektet.

Sikkerhet: Sikkerhet er alltid et viktig aspekt og kriterium for dagens IT-verden. Alle som utvikler eller drifter digital infrastruktur bør ha dette i bakhodet. Som nevnt tidligere består brukergruppen i SkyHiGh blant annet av forskere og studenter som har forskning- og læringsverdier i skytjenesten. Derfor ser gruppen viktigheten av å ha god sikkerhet for en eventuell løsning, slik at den ikke vil kompromittere den sikkerheten som allerede eksisterer i SkyHiGh. Dette innebærer blant annet at en eventuell løsning ikke skal åpne opp for nye angrepsvektorer mot skyplattformen.

2.2.1 Use cases

I figur 2 er det fire use cases som er relevante for oppgaven og driften av SkyHiGh. Hver av disse use casene har en tilhørende tabell for mer utdypning og beskrivelse.



Figur 2: Use case diagram

Case 1

Case	Installasjon og konfigurasjon
Aktør	SkyHiGh driftsgruppe
Betingelser	Løsning er valgt på forhånd og konfigurasjon er spesifisert. Løsningen er prøvd ut i testmiljø. Maskinvare er gjort tilgjengelig.
Beskrivelse	Maskinvare kobles opp og forberedes for installasjon av programvare. Tidligere bestemt programvare og dets avhengigheter anskaffes. Forhåndsbestemt konfigurasjon defineres i konfigurasjonsfilene. Løsningen rulles ut i produksjonsmiljøet. Tester gjennomføres for å sørge for at løsningen fungerer som den skal.

Tabell 1: Use case 1

Case 2

Case	Endringer av variabler
Aktør	SkyHiGh driftsgruppe
Betingelser	Løsningen er allerede rullet ut. Det er gjort en vurdering om produksjonsmiljøet må tas ned for å gjennomføre handlingen. Endringene er testet i testmiljøet.
Beskrivelse	Produksjonsmiljø tas ned om nødvendig. Endringer foretas i konfigurasjonsfiler. Produksjonsmiljø rulles ut igjen. Tester gjennomføres for å sørge for at løsningen fungerer som den skal og for å sikre at endringene som ble gjort har blitt gjennomført.

Tabell 2: Use case 2

Case 3

Case	Oppgradering av tjenester
Aktør	SkyHiGh driftsgruppe
Betingelser	Detaljene for oppgraderingen har blitt gjennomgått, inkludert eventuelle endringer i forutsetninger som f.eks. databasetabeller. Det er gjort en vurdering om produksjonsmiljøet må tas ned for å gjennomføre handlingen. Oppgraderingen har blitt testet i testmiljøet.
Beskrivelse	Produksjonsmiljø tas ned om nødvendig. Oppgradering av tjeneste foretas enten ved å hente ned aktuell programvare for så å installere, eller ved å definere kilde og versjon i konfigurasjonsfiler. Metode avhenger av hva slags løsning som brukes. Eventuelle endringer som må gjøres som følge av ny versjon gjennomføres. Tester gjennomføres for å sørge for at løsningen fungerer som den skal, at oppgraderingen har blitt gjennomført, og at eventuelle endringer oppgraderingen førte til stemmer overens med forventet resultat.

Tabell 3: Use case 3

Case 4

Case	Introdusering av nye tjenester og sletting av tjenester
Aktør	SkyHiGh driftsgruppe
Betingelser	Konsekvenser for introdusering eller sletting er vurdert. Eventuelle endringer som må gjøres som følge av den planlagte prosessen er redegjort for. Det er gjort en vurdering om produksjonsmiljøet må tas ned for å gjennomføre handlingen. Introdusering eller slettingen har blitt testet i testmiljøet.
Beskrivelse	Produksjonsmiljøet tas ned om nødvendig. Ved introdusering av nye tjenester anskaffes kildefiler eller kilde defineres i konfigurasjon. Introduseringen eller sletting gjennomføres. Eventuelle endringer som kreves som følge av handlingen blir gjennomført. Tester gjennomføres for å sikre at de tjenestene som skal være tilstede fungerer, og at ønsket resultat har skjedd.

Tabell 4: Use case 4

3 Utviklingsprosess

3.1 Utviklingsmetode

Det er viktig for et prosjekt å etablere en struktur med mål og retninger slik at prosjektet holder stødig kurs mot gitt mål. En utviklingsmetode er et rammeverk for selve utviklingen av oppgaven og vil hjelpe prosjektets formål med å nå det korrekte resultatet. Det finnes flere forskjellige typer utviklingsmetoder som alle fungerer til sine formål og har sine optimale måter å brukes på. Det er ikke alltid like lett å vite hvilken metode som passer hvilken type oppgaver, derfor er det viktig at det gjøres en enkel utredning av den mest relevante metoden for prosjektet, for deretter å finne ut hvorfor den metoden virker best for et gitt prosjekt.

3.2 Begrunnelse for valg av metodikk

Slik nevnt, gjøres det utredning av mange relevante teknologier, de fleste av disse blir filtrert vekk relativt tidlig (innen en måned etter prosjektstart). Det vil sørge for at gruppen har tydelige rammer og en klarere vei videre. Deretter gjøres det fler forskjellige utredninger og først da vil oppgavens endelige mål bli definert. Oppgaven er av en slik natur at arbeidsoppgavene kan endres ut fra hvordan utredningene våre går. Det vil si at en mer statisk utviklingsmodell slik som fossefall-modellen, som egner seg mer for tydelige mål, ikke vil være passende for denne oppgaven. Derfor er det logisk å gå for en mer smidig utviklingsmodell, og har med det bestemt oss for å bruke Scrum.

Scrum som utviklingsmodell er utmerket i en situasjon som vår, ettersom modellen er veldig tydelig og definert, og den er i tillegg lett å ta i bruk. Ved å bruke en tydelig definert utviklingsmodell som gruppemedlemmene allerede har vært borte i før sørges det for at vi ikke bruker overdrevent mye tid rundt utviklingsmodellen og heller kan fokusere mer på selve oppgaven. Alle på gruppen er kjent med Scrum fra tidligere semestre, da vi har jobbet med forskjellige varianter av den i flere tidligere prosjekter. Blant annet vanlig Scrum og en variasjon av Kanban og Scrum, Scrumban. Modellen er også passende for oss med tanke på antall medlemmer i gruppen.

Scrums smidighet er hovedpunktet for valget av metodikken. Dette er en metodikk som er bygget for endringer underveis i prosjektet ved bruk av sprinter og milepæler. En mer statisk metodikk vil ha en mer rettet vei mot enden, mens Scrum tilbyr mindre men fler milepæler gjennom hele prosessen.

Karakteristika ved prosjektet som er relevant for modellvalg

- Prosjektgruppen aner ikke utfallet av oppgaven ved start av prosjektarbeid.
- Fleksibel oppgave som kan endres flere ganger over tid.
- Endelig løsningsforslag kan forandres ved arbeid på nøkkelpunkter midt i prosjektet.

- Allerede kjent metodologi for prosjektgruppen.

3.3 Gjennomføring av metodikk

Siden prosjektperioden er forholdsvis kort og krever klar progresjon på deloppgaver jevnlig, så har vi valgt å benytte oss av to ukers sprinter. Vi kommer til å operere med *daily scrum* møter på starten av hver dag, der vi går igjennom hvordan vi ligger an og hva dagens gjøremål skal være. Det vil også gjennomføres *sprint review* og *sprint planning* møter. Disse møtene vil ledes av en *scrum master*. Der har vi valgt Hamse, da han er den i gruppen med mest tidligere erfaring i Scrum. *sprint review* og *sprint planning* vil begge foregå på mandager, siden vi har møte med veileder og i oppstartsfasen også produkteiere. Det vil bli ført opp deloppgaver i en *product backlog*. Oppgavene i *Backlog* vil flyttes over til *Sprint backlog* da de velges for neste sprint.

Når en oppgave blir tildelt et eller flere gruppemedlemmer blir den flyttet til seksjonen *Arbeides på*. Oppgaven blir så flyttet over til seksjonen *Til vurdering* da den/de som jobbet på oppgaven er fornøyd med utkastet. Deretter blir oppgaven flyttet over til seksjonen *Ferdig* da den er vurdert som godkjent av enten ett gruppemedlem som ikke deltok på oppgaven, eller hvis alle gruppemedlemmene er enige på oppgaver der alle deltok.

3.4 Detaljert bruk av Scrum

Utviklingsmodellen har en del unike punkter den er bygget på. Disse punktene er viktig å følge for gjennomføring av prosjektet på mest mulig ryddig og effektiv måte.

3.4.1 Sprint planning

Siden prosjektperioden er forholdsvis kort og krever klar progresjon på deloppgaver jevnlig, så har vi to ukers sprint om gangen. Dette har gruppen bestemt skal være annenhver mandag. Sprintene ble drøftet innad i gruppen hvor de diskuterte for det meste består av progresjon, forventet progresjon, og hva som skal arbeides med neste uke. Det ble så lagt inn eller overført fra gamle Trello-tavler oppgaver i en ny Trello-tavle for den nåværende sprinten.

3.4.2 Daily sprint

Gruppen operer med en daglig ti minutters sprint for planlegging av dagens gjøremål. Dette gjøres på starten av dagen og hjelper gruppen med å raskt tildele oppgaver for dagen. Disse møtene foregår kun muntlig i og med at det ikke forekommer noen drastiske diskusjoner og endringer. Om noe mer alvorlig haster å bli tatt opp, noteres det ned til neste mandagsmøte eller innkalles til ekstramøte.

3.4.3 Sprint review

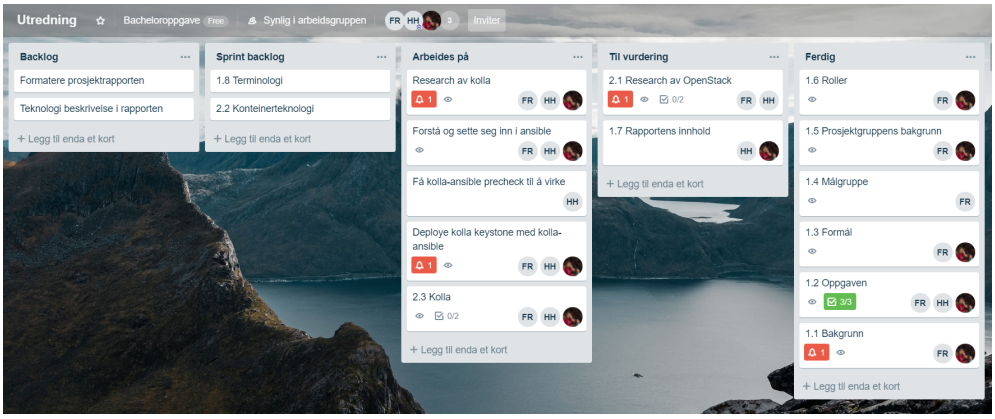
Annenhver mandag blir det sprint review møte, der gruppen diskuterer hva gruppen har fått utført i forrige sprint, dette foregår på samme dager som sprint planning, naturligvis rett før planleggingen av neste sprint. Det gir gruppen et oppfrisket minne om hva som ble utført og hva som ble igjen fra forrige sprint slik at vi kan planlegge det til neste sprint. I tillegg noterte vi fra hvert sprint review møte, hvor vi diskuterte hva vi har gjort foregående uke, hva som gikk bra, hvilken utfordringer vi støtte på, for også gjøre en overgang til sprint planning med hva som er de neste gjøremålene.

3.4.4 Møte med veileder og oppdragsgiver

Gruppen har faste møter med veileder hver mandag for å sjekke status for fremdrift og for å få veiledning. Oppdragsgivere vil også jevnlig bli invitert til disse møtene. Ved oppstartsfasen av prosjektoppgaven gjennomføres det et utvidet møte med oppdragsgiver for å forklare oss oppgaven, infrastrukturen i SkyHiGh, kravspesifikasjoner, og ønsket resultater. Under disse møtene noteres det også spørsmål til veileder og oppdragsgiver, og eventuelle svar og diskusjoner rundt spørsmålene. Resultatene fra dette gåes gjennom i påfølgende sprint planning og sprint review.

3.4.5 Scrum board

Som allerede nevnt tidligere benyttet vi Trello for å lage et sprint planning board 3.



Figur 3: Gruppens Trellotavle

Brettet er delt opp i backlog, sprint backlog, arbeides på, til vurdering, og ferdig. Gruppemedlemmene tildeler seg selv til oppgaver som trengs arbeid. Det gir en god oversikt til gruppen for å finne ut hvem som gjør hva eller hvem har gjort hva.

4 Teori

4.1 OpenStack

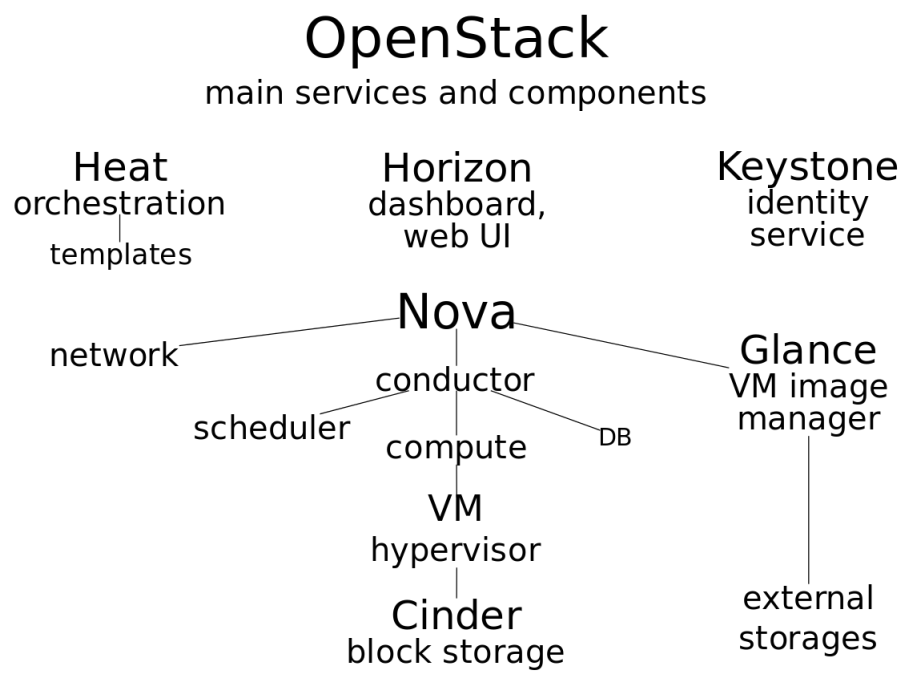
OpenStack er en open-source platform for skyløsninger. Det er hovedsakelig sett på som en infrastrucutre-as-a-service platform. OpenStack er en privat og offentlig skyløsning. Det vil si at enten er skyen kunden leier dedikert til kun kunden (privat sky) og vil da få mer kontroll og sikkerhet enn om kunden skulle brukt offentlig skyløsning, hvor ressursene er delt mellom flere kunder. OpenStack er en løsning der man leier ut virtuelle maskiner, konteinere, stacks, eller lignende til kunder. OpenStack består av flere komponenter som leies ut til kunden eller som kunden kan ta i bruk selv enten via det web-baserte grensesnittet Horizon, eller ved bruk av kommandolinja og API-er.

OpenStack ble startet i 2010 av Rackspace og NASA [4] og kom med sin første offentlige utgivelse Austin 21. Oktober samme år[5]. I 2011 adopterte Ubuntu utviklerne OpenStack og ga støtte for plattformen [6], deretter fulgte blant annet Debian [7] og SUSE [8] og mange fler [9]. I 2012 valgte NASA å trekke seg ut som aktiv bidragsyter til OpenStacks utvikling og gikk over til en av gigantene innen offentlige skyløsninger, Amazon Web Services [10]. OpenStack er nå styrt av OpenStack Foundation og har totalt over 600 kunder globalt [9]. OpenStack har som mål å produse en ubikvitær open source cloud computing plattform [11]. Den skal være lett å bruke, enkel å implementere, høy grad av interoperabilitet mellom forskjellige API-endepunkter og fungere bra i alle skalaer. Den skal møte både kundens og administratorens behov når det kommer til både offentlige og private skyer.

Hver sjette måned slippes det ut ny versjon av OpenStack i alfabetisk rekkefølge [5]. Disse versjonene er bundet til støttede Unix distribusjoner. Første versjon er Austin som ble utgitt 21. oktober 2010 [5]. Nåværende versjon er Stein som ble utgitt 10. april 2019 [5]. Mellom disse versjonsutslippene samler OpenStack-samfunnet seg på forskjellige steder i verden under noe som blir kalt OpenStack Summit. Her blir nyutviklede teknologier og løsninger presentert og diskutert foran eksperter, brukere og interesserte fra hele verden.

4.1.1 Arkitektur

OpenStack har en modulær arkitektur, der det kan velges hvilke komponenter som brukes i forskjellige implementasjoner. Komponentene gir tilgang til hverandres tjenester ved bruk av APIer. Selv om en modul kan gjøre en jobb alene, vil funksjonaliteten ofte utvides da tilgang til andre tjenesters APIer blir lagt til. Komponentene tilpasser seg selv til den arkitekturen de får tilgang til. Figur 4 viser OpenStacks hovedkomponenter.



Figur 4: Komponentene i OpenStack
[12]

4.1.2 Komponenter

Keystone

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[13][14][15]. Keystone er identitetstjenesten i OpenStack. Tjenesten er ansvarlig for autentisering og autorisering. Keystone bruker en database for å lagre brukere. Den kan bruke flere autentoriseringsmetoder som brukernavn og passord og tokens. AWS autentisering er også støttet. API endepunkter fra andre tjenester registreres i Keystone, slik at tjenesten også gjør service-discovery. Keystone tar aldri initiativ til kommunikasjon, men svarer kun på forespørsler fra andre tjenester.

Glance

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[16][17][15]. Glance er imagetjenesten i OpenStack. Glance muliggjør søking/oppdagelse, registrering og lagring av images som kan brukes av andre tjenester. Den gir også mulighet for lagring av backups. Glance kan lagre disker og images i flere backends, som for eksempel Swift. Tjenesten kommer med et API som kan bli brukt for å hente images ved hjelp av spørringer. Andre OpenStack tjenester som bruker images kommuniserer gjennom Glance, men kun Glance kan opprette, slette, foreta endringer, eller lignende på images. VMware kan og integreres inn for å gi ekstra funksjonalitet til tjenesten. APIet i Glance kan gi informasjon om images eller gjøre imagene i seg selv tilgjengelige for brukere.

Neutron

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[18][19][15]. Neutron er OpenStacks system for å administere nettverk og IP-adresser. Tjenestene til Neutron inkluderer å gjøre det mulig å bygge avanserte nettverkstopologier samt å utvikle regler (policies). Sluttbrukere har gjennom Neutron verktøyer for å håndtere IP på instansene sine. IP-adressene kan settes statisk eller ved bruk av DHCP. Neutron kan også sette floating IP, som er en offentlig IP adresse for tilgang til instansene utenfor nettverket. Brukere kan i tillegg lage nettverk, konfigurere nettverkene, og koble instanser til nettverk. Neutron tilbyr forskjellige nettverks modeller for forskjellige applikasjoner og brukergrupper. I tillegg kan Neutron tilby utvidelser som OpenFlow, IDS, lastbalansering, brannmur og virtuelle private nettverkstjenester.

Nova

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[20][21][15]. Nova, også kjent som Compute, er komponenten i OpenStack som står for håndtering og fordeling av dataressurser. Nova er designet for å administrere og automatisere samlinger av maskinressurser. Den arbeider både med virtualisering samt med fysisk maskinvare (baremetal). Da en virtuell maskin opprettes i OpenStack vil ressursene bli reservert i Nova og et image blir instansert og provisjonert. Nova har også et API for avansert logikk og automatisering av provisjoneringen av instanser. Novas arkitektur gjør rede for horisontal skalering på standard maskinvare uten å være låst til noen produsenter, både på maskinvare og programvare, og gir støtte til eldre systemer og tredjeparts teknologier. Nova er avhengig av Keystone, Glance og Neutron for å kunne utføre sin basisfunksjonalitet.

Heat

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[22][23][15]. Heat er OpenStacks orkestrerings program, det er et verktøy som bruker templates for orkestrering av infrastruktur. Heat bruker en orkestreringsmotor for å deploye en eller fler enheter basert på yaml tekstfiler. Ressurser som kan bli beskrevet i Heat er for eksempel servere, flytende IP-er, sikkerhetsgrupper, volumer også videre. I tillegg kan Heat templates også spesifisere forhold mellom ressurser. Heat kan brukes sammen med teknologier som Puppet og Chef, og har APIer som gjør det mulig å jobbe med blant annet AWS Cloud Formation.

Horizon

Teksten er basert på informasjon fra OpenStack-dokumentasjon og Wiki-sider[24][25][15]. Horizon er et webbasert brukergrensesnitt for de fleste av OpenStack sine tjenester. Grensesnittet er tilgjengelig både for administrator og brukere, med forskjellig funksjonalitet for hver av de. Horizon er dynamisk og tilpasser seg hvilke komponenter og tjenester som er aktive. Det er også støtte for verktøy fra andre leverandører, og det er et API for kompabilitet med AWS.

4.1.3 Støtteverktøy

RabbitMQ

Teksten er basert på informasjon fra RabbitMQ-dokumentasjon[26]. RabbitMQ er meldingskøen som brukes i OpenStack for kommunikasjon mellom noen av tjenestene. Det brukes av mer enn 35000 brukere i små til store konserner, og er det den mest populære meldingskøen med åpen kildekode. RabbitMQ er lett å bruke og rulle ut enten lokalt eller i skyen. Det støtter også flere meldingsprotokoller og kan bli distribuert og konfigurert til å møte store skalerings -krav og behov. I tillegg støttes det av mange operativsystemer og distribusjoner samt gir støtte til utviklingsverktøy og flere programmeringsspråk.

OpenStackClient

Dette er kommandolinjeverktøyet for OpenStack. Det finnes også tjenestespesifikke verktøy, men OpenStackClient har kjernefunksjonaliteten for alle tjenestene[27].

4.2 SkyHiGh

SkyHiGh er NTNUs egen private sky som er bygget på OpenStack. For tiden er SkyHiGh driftet av to tidligere studenter, som begge nå er ansatt ved NTNU som ansvarlige for SkyHiGh. Skyens arkitektur ligner allerede en type konteinerarkitektur, hvor det er dedikerte virtuelle maskiner som kjører en gitt tjeneste og kun det. SkyHiGh kan tilby kunder (studenter for det meste) egne virtuelle maskiner, eller egne prosjekter hvor brukerne kan sette opp stacks.

Det finnes tre typer OpenStack installasjoner ved NTNU.

- skyhigh.iik.ntnu.no - IIKs installasjon i Gjøvik. Brukt for læring og forsknings aktiviteter.
- skylow.iik.ntnu.no - IIKs egne testing og utviklings plattform.
- stack.it.ntnu.no - NTNUs IT-avdeling sin OpenStack sky for generelle formål.

En type praktisk eksempel ville da vært en gruppeoppgave hvor man er to til tre personer per gruppe der gruppen får tildelt en kvote på diverse ressurser som for eksempel: 10 instanser, 20 virtuelle cpu-er, 60 GB ram, 20 flytende IP-adresser, 10 sikkerhetsgrupper, 10 volumer og 100GB volumlagring. Her kan studente selv ta i bruk ressursene for å løse oppgaver ved hjelp av tjenester som SkyHiGh tilbyr. Alt gjøres gjennom OpenStack.

4.2.1 Arkitektur

Slik nevnt i 4.2, ligner SkyHiGhs arkitektur allerede en type infrastruktur-oppsett som kjører tjenester i konteinere ved at de virtuelle maskinene kjører dedikerte oppgaver. Det vil da ikke bli foretatt eller foreslått arkitekturendringer siden løsningsforslag vil kunne utbedres i allerede implementert arkitektur.

Nettverksarkitekturen er delt opp i intranett og ekstranett. Studentene har kun tilgang til nødvendige ressurser mens administratorene og tjenestene har tilgang til og kommuniserer internt innad i administrasjonsnettet. Det er en standard explicit deny policy på nettverket, det vil si alt av porter er som standard lukket med mindre det er åpnet av administratorene.

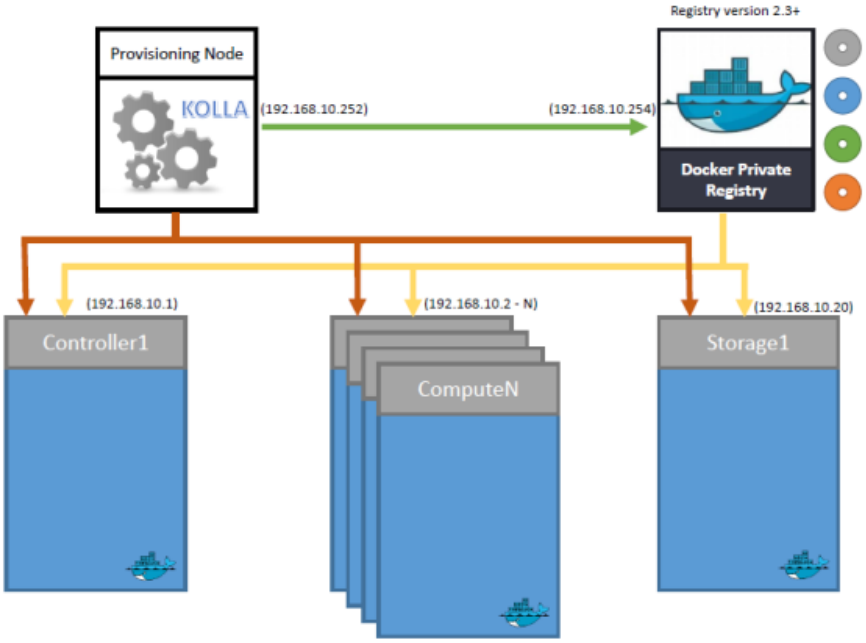
4.3 OpenStack-Kolla

OpenStack Kolla er et prosjekt som har som mål å bygge produksjonsklare konteinere ved å distribuere images av tjenestene i OpenStack for bruk i konteinere, og verktøy for å kjøre disse konteinerne. For noen år siden ble prosjektet splittet opp i to deler, og det som nå heter Kolla er den delen av prosjektet som gir ut nye images av de forskjellige tjenestene. Utrulling av konteinerne ble flyttet til et søsterprosjekt kalt Kolla-Ansible. Da vi referer til Kolla menes altså kun prosjektet som bygger konteinerimages, ikke styringsverktøyene i Kolla-Ansible.

4.3.1 Images

OpenStack-Kolla gir flere muligheter for hvordan man kan bygge og lagre images. Den enkleste måten er å hente images fra Kolla prosjektet sine ferdigbygde images som kan hentes fra OpenStack sitt domene. Kolla støtter også repositories som DockerHub, der images kan hentes eller lastes opp. Images kan også bygges lokalt. Hvilke tjenester og kilde for imagene som skal være med i imaget defineres i konfigurasjonsfiler. Kilden for imagene vil oftest være enten OpenStack sitt domene eller et repository. Kolla bygger tjenestene som er definert inn i et image som gjøres tilgjengelig lokalt.

4.3.2 Kolla-Ansible



Figur 5: Kolla-ansible multinode arkitektur
[28]

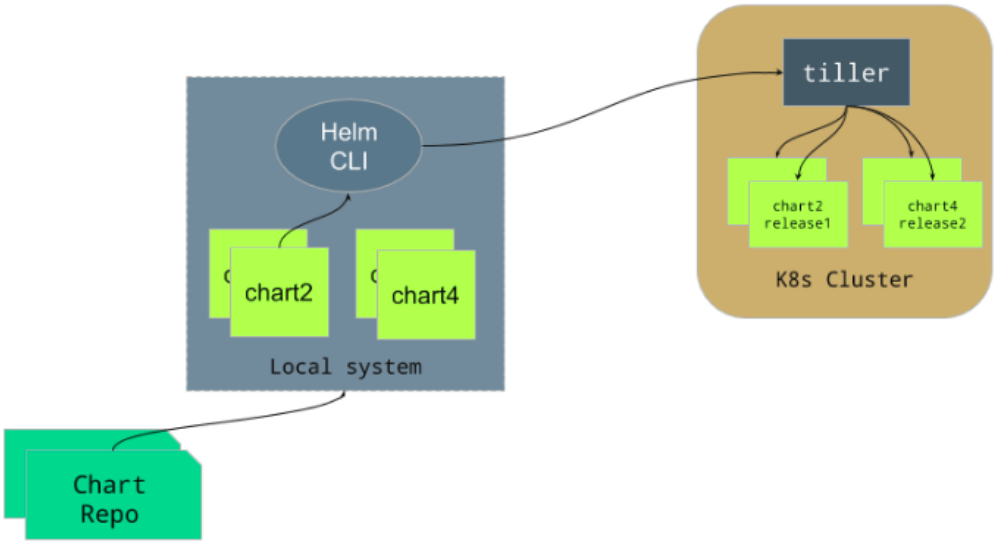
Kolla-Ansible er et søsterprosjekt av OpenStack-Kolla og er et prosjekt som gir et styringsverktøy for deployment og administrasjon av Kolla-konteinere. Dette prosjektet benytter Ansible for å kunne gjennomføre en rekke med tester og bootstrap aktiviteter før den til slutt deployer konteinere med tjenester og konfigurasjoner definert i OpenStack og Ansible sine konfigurasjonsfiler. Hvilke noder som skal brukes defineres i en Ansible

inventory-fil. Tjenesten bruker enkle kommandoer og har en enkel inventory-fil som krever minimal kunnskap om Ansible fra før av.

Den vanligste arkitekturen for Kolla-Ansible bruker Control og Compute arkitekturinndeling med en provisjonsnode som vist i figur 5. Provisjonsnoden har ansvaret for å konfigurere alle noder i arkitekturen ved hjelp av Ansible. Denne bruker nøkler for å kommunisere med de andre nodene. Man kan også ha en privat Docker-registry for å oppbevaring av Docker images. Nodene som skal kjøre konteinere vil hente imagene fra denne. Control noder inneholder normalt sett alle komponentene til OpenStack utenom Nova-Compute, lagringstjenester. Under Compute finnes det Nova og "computed instancesöm vil si instanser laget ved bruk av OpenStack[28]. Storage inneholder alle tjenester som står for lagring og backup.

4.4 OpenStack-Helm

OpenStack-helm er et prosjektet laget av OpenStack for konteinerisering av OpenStacks tjenester ved bruk Kubernetes og Helm. Målet med Openstack-Helm er å tilby en samling av såkalte Helm charts som enkelt, robust, og fleksibelt kan rulle ut OpenStack tjenester på toppen av Kubernetes. Helm er et pakkehåndteringsverktøy for Kubernetes [29]. Slik som apt blir brukt for å installere pakker på en Ubuntu maskin, brukes Helm blant annet til å installere pakker og kjøre dem i Kubernetes kluster [30]. Charts er pakker som inneholder prekonfigurerte Kubernetes ressurser. Helm har to komponenter, den første er Helm og finnes på klientsiden. Den andre er Tiller og lever i Kubernetes klusteret slik vist på figur 6. Tiller har ansvaret for å holde oversikt over release- versjonen [30]. Helm tillater enkel håndtering av ferdige Kubernetes ressurser og tilbyr blant annet tilbakerulling til en tidligere versjon, fordi historikken for utslipp eller realese lagres [29].



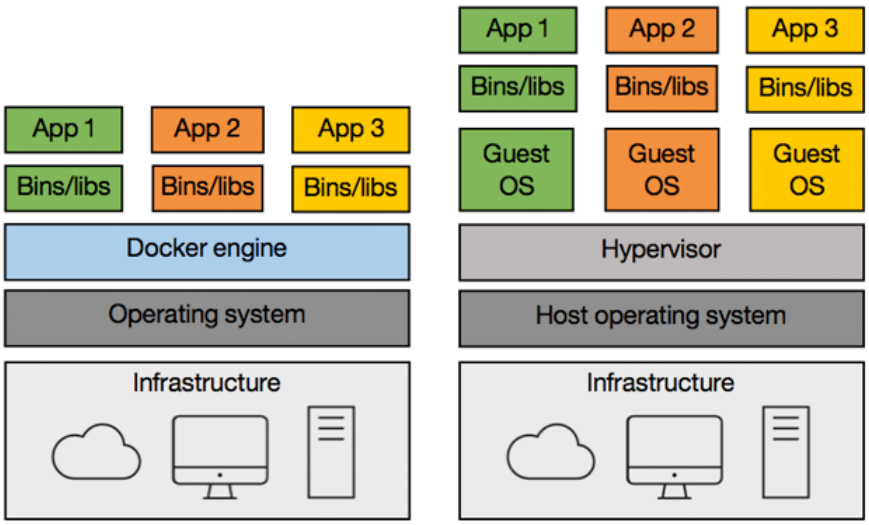
Figur 6: Helm artefakter
[31]

I OpenStack-Helm sitt tilfellet brukes Helm i skripter til å sette opp OpenStack tjenester slik som Keystone, og støttetjenster som MariaDB, Memcached og lignende [32]. OpenStack-Helm tilbyr to metoder for å sette opp OpenStack i konteinere. All-In-One deployment setter opp de tjenestene man ønsker på en enkel maskin enten fysisk eller virtuelt. Denne er mer egnet til utprøving og testing av Helm, og vil ikke være hensiktsmessig med tanke på single point of failure og har ikke skaleringsmuligheter [33].

Den andre og mer relevante metoden de tilbyr heter Multinode deployment. For begge tilfeller så er det forventet at du har Kubernetes installert og klar for bruk. Multinode har en ekstra forutsetning, å ha et Kubernetes kluster klar. OpenStack tilbyr riktignok guider for å sette dette opp for utprøving ved hjelp av scripts og Ansible, men det anbefales ikke å bruke dette oppsettet for produksjonsmiljø [34]. Helm består av to git repositories. Det ene er openstack-helm-infra for å sette opp Kubernetes kluster. Det andre er openstack-helm som inneholder alle Chartsene for de ulike OpenStack tjenestene [35].

4.5 Konteinerteknologi

Konteinerteknologi har hatt en sterk vekst i senere år. Docker er standarden i bransjen for øyeblikket[3]. Det ble først lansert i 2013 som et open source prosjekt. Konteinere av en abstraksjon på applikasjonsnivå, der virtuell maskin er en abstraksjon på maskinvarenivå [36]. En konteiner pakkes sammen til en eneste enhet, bestående av kode og all dens avhengigheter [36]. Docker er tilgjengelig for de fleste operativsystemer. En konteiner er ment å ha kun en oppgave, selv om den kan kjøre flere oppgaver. Den benytter ressursene til det underliggende operativsystemet til vertsmaskinen. Dette er en av grunnene mange velger mikrotjenester fremfor monolittiske tjenester[36].



Figur 7: Konteinerarkitektur sammenlignet med arkitektur for virtuelle maskiner [37]

Noen av fordelene med konteinere fremfor den tradisjonelle monolittiske tilnærmingen:

- Lettvektig og ressurseffektiv. Krever ikke egen OS sammenlignet med en applikasjon som kjører i virtuelt miljø [36].
- Isolasjon og applikasjon sandboxing. Konteinere har sitt eget miljø med egne prosesser, mapper osv. Gir ekstra lag med sikkerhet da applikasjonen ikke kjører direkte på vertsmaskinen [38].
- Konteinere er portable og kan kjøres nesten hvor som helst. Windows, Mac og Linux, virtuel eller fysisk maskin. Dette gir færre faktorer å ta hensyn til for utviklere[38].
- Enkelt å starte og stanse. Det tar bare noen sekunder å starte og stanse en konteiner. Dette er betraktelig bedre enn monolittisk miljø der det tar mye lenger tid.
- Bidrar til oppnåelse av 'loosely coupled'- tjenester eller isolerte tjenester som er en viktig fordel når tjenester senere for eksempel skal oppdateres hver for seg.
- Mulighet for orkestrering, gir økt redundans og skalerbarhet.

5 Initielt utvalg av teknologier

For å velge ut hvilke teknologier som skal testimplementeres og vurderes i detalj gjøres det først forskning av mulige kandidater for konteinerisering av OpenStack. De valgte kandidatene vil bli brukt til å utvikle et Proof of Concept med hver teknologi, der de vurderes opp mot hverandre til slutt slik at det kan fattes en konklusjon om anbefalt teknologi.

5.1 Kriterier

Det er store eksisterende aktører innen skytjenester som allerede bruker OpenStacks konteinertjenester [9]. Hva disse aktørene bruker vil være en god indikator på hvilke teknologier vi burde vurdere. Teknologien må kunne sette opp en eller flere tjenester fra OpenStack i konteinere, ha mulighet for selvdefinert konfigurasjon, og kunne bli koblet sammen med eksterne tjenester. Databaser må kunne defineres som ekstern tjeneste. Teknologien bør være kompatibel med automatisering slik at den kan settes opp i for eksempel Puppet som allerede er i bruk idag. Løsningen skal ha gode muligheter for feilsøking. Gode brukergrupper og forumer slik at det blir enkelt å finne ut av mangler, bugs, utfordringer og problemer ved løsningen regner gruppen som en god indikator på dette. Dette punktet er også viktig fordi det gir forventet levetid og stabilitet i forhold til fremtidige utgivelser. Løsningen må også være egnet for en sky av SkyHiGhs størrelse.

Liste av kriterier for initielt utvalg:

- Populæritet, trend og community
- Arbeidsflyt (Git, enkelt å gjøre feilsøking osv.)
- Oppsummering av kompleksitet, hvor praktisk løsningen vil være for IIK og forventet arbeidsmengde i henhold til tidsbegrensningen.

5.1.1 Utforskning og metoder

Gruppen har gjort forskning på eksisterende løsninger og hva andre bruker for å løse konteinerisering av OpenStack. En del store aktører tar i bruk Kubernetes og OpenStack Helm, slik som AT&T og SK-Telecom[39]. Flere andre aktører har tatt i bruk Kolla-Ansible, for eksempel RedHat[40] og noen av de store aktørene innen spillindustrien[41]. I forskningsdelen har gruppen brukt flere verktøy for å finne aktuelle kandidater. Gruppen har brukt nettsøk som hovedverktøy for informasjonsinnhenting. Dette har ledet oss til OpenStack sine sider, offisielle dokumentasjonssider, forumer og bloggposter. Gruppen har også forsøkt å komme i kontakt med andre både på skolen og utenfor for å finne tips og anbefaling til å plukke ut kandidater.

En av personene vi tok kontakt med er Jan Ivar Beddari etter anbefaling fra vår veileder. Han jobber til daglig med blant annet OpenStack i firmaet SafeSpring. Vi hadde håpet på å få høre fra han om hvilke konteinerløsninger han synes er lurt å se på for

konteinerisering av OpenStack. Vi har dessverre ikke fått noe respons fra han.

I tillegg tok vi kontakt med Kyrre Begnum, som tidligere har vært vår foreleser i emnet IMT3003 - Drift av tjenestearkitektur. Vi tok kontakt med han av samme grunn som Jan Ivar Beddari. Han har god kjennskap til OpenStack da han var den første til å implementere skolens private skyløsning, SkyHiGh. Han hadde dessverre ikke noe aktuelle tips å gi for å implementere OpenStack i konteinere, men anbefalte oss å se på Mirantis sin løsning for konteinerisering av OpenStack. Veilederen vår Erik Hjelmås anbefalte oss også å se om Mirantis kunne ha en potensiell løsning, da han hadde positiv erfaring med Mirantis fra da IIK planla den nåværende arkitekturen til SkyHiGh.

5.2 Kandidater

5.2.1 OpenStack Kolla-Ansible

OpenStack Kolla har hatt flere prosjekter som har blitt videreutviklet samtidig. Etter å ha gjort research om hvilke prosjekter i Kolla som arbeidet med hva og hvilke som fortsatt er aktive, fant vi ut at den mest aktuelle varianten var et prosjekt kalt Kolla-Ansible.

Popularitet, trend og community

Kolla-Ansible gir gode resultater både på søketreff og på forumaktivitet. Den har flere forumer for feilsøking og hjelp til å sette opp løsningen korrekt. Det er blant annet en side på launchpad for bugs [42]. Det eksisterer også god dokumentasjon fra OpenStack selv for å sette opp tjenesten som AllInOne eller Multinode [43]. Lenker til kildekode, bidragsytere, IRC-kanal og mer finnes på wiki-siden [44] til Kolla-prosjektet. Selv om Kolla og Kolla-Ansible nå er forskjellige prosjekter er det veldig mye overlapping i bidragsyterne og brukermiljøet til begge prosjektene. Kolla-Ansible har i noen år vært en av de ledende løsningene for konteinerisering av OpenStack i større bedrifter, og har derfor en god markedsandel og sunt med bidrag fra disse. Det har derimot vært flere bedrifter som har stoppet å gjøre bidrag til prosjektet, som tyder på en noe redusert vekst.

Arbeidsflyt

Det er lagt opp slik at man følger en steg for steg anvisning fra den offisielle OpenStack installasjonsveiledningen. Den bruker Ansible til å rulle ut konteinere, som ikke utelukker muligheten til å bruke et annet konfigurasjonsstyring system slik vi ser det. Dette kan gjøre det mulig å integrere det i for eksempel Puppet, som brukes allerede idag av IIK. Det eksisterer også Puppetmoduler [45] for Ansible-konfigurasjoner slik at Puppet kan styre de få kommandoene som er nødvendig for å rulle ut konteinere med Ansible.

Oppsummering

Etter å ha lest visjonen til prosjektet ser man at den ville passet godt for IIK, både når det kommer til kompleksiteten og ikke minst fordi den tilbyr fleksibilitet når det kommer til konfigurasjonsendringer.

5.2.2 OpenStack Helm

Popularitet, trend og community

OpenStack Helm brukes av flere av de største aktørene innen Cloud-infrastruktur som nevnt i teorikapittelet. Dette inkluderer store selskaper slik som AT&T og SK-Telecom

[39], som er to av de største teknologiaktørene i USA og Sør-Korea. OpenStack-Helm har et Storyboard-forum for bugs o.l. for brukerne [46]. Prosjektet har også en aktiv Slack-kanal. Det er lite offisiell dokumentasjon som gir vanskeligheter i forhold til kompleksiteten til et prosjekt av Helms kaliber, som bruker komplekse teknologier som Kubernetes og Helm Charts [47]. Vurdert ut i fra hyppigheten på det offisielle GitHub repoet og Slack kanalen til OpenStack- Helm, har prosjektet derimot bra vedlikehold og support.

Arbeidsflyt

OpenStack-Helm tilbyr guider som kan brukes for å sette opp kluster i Kubernetes for bruk i OpenStack-Helm. Oppdateringer og rutinearbeid gjøres ved bruk av Kubernetes-verktøy. Kubernetes blir brukt til styring av konteinere som kjører tjenester. OpenStack-Helm bidrar med flere fordeler, slik som modularitet, tilbakerulling og versjonskontroll for å holde oversikt over nye endringer [30] [29].

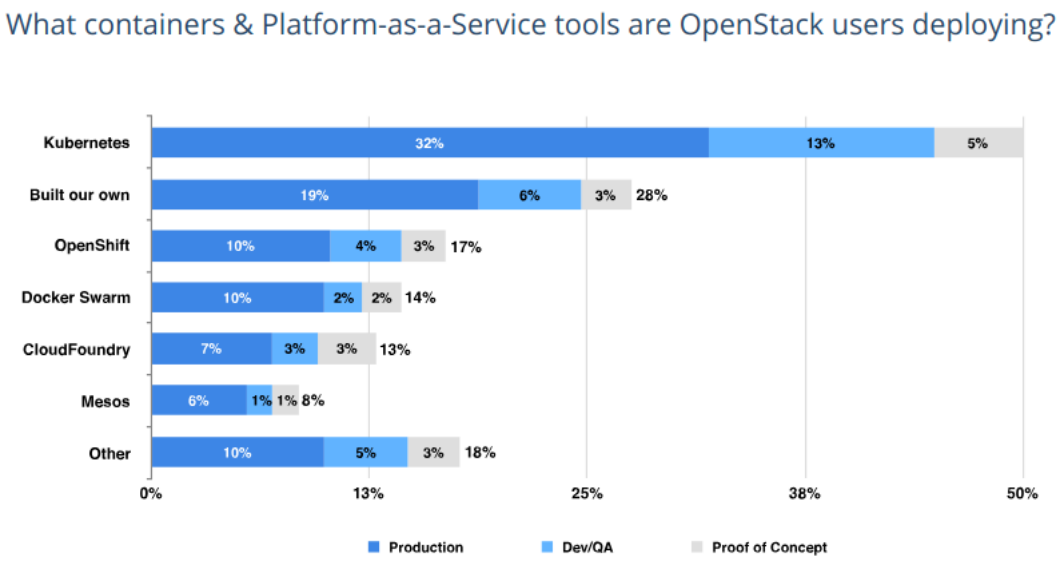
Oppsummering

Etter vi leste om denne løsningen så vi på den som den ideelle kandidaten. Blant annet fordi den er et OpenStack prosjekt og bruker Kubernetes som blir ansett som den mest komplette konteinerorkestreringen. Den muliggjør automatisering og CI/CD ved bruk av verktøyer slik som Codefresh. De andre løsningene vi hadde sett på tidligere slik som Kolla-Ansible tilbyr ikke konteinerorkestrering.

5.2.3 Selvbygde konteinere

Popularitet, trend og nettsamfunn

Selvbygde konteinere involverer både å bygge konteinerimages selv og styring av de. Flere aktører kjører en egen implementasjon med selvbygde konteinere for å sette opp OpenStack i konteinere, illustrert i figur 8.



Figur 8: Det OpenStack-brukere benytter av verktøyer for PaaS og konteinere [48]

Egenbygde konteinere ble også nevnt som et ønske fra oppdragsgiveren vår. Denne løsningen er veldig åpen for arkitekturdesign, implementasjon, kravspesifikasjon og konteinerteknologier man ønsker å bruke. Dette kan være både positivt og negativt. På grunn av tidsbegrensingen og at vi skal gjøre en vurdering og testimplementasjon av flere andre kandidater har vi valgt å lage en enkel proof-of-concept for denne løsningen. Dette innebærer at vi setter opp en fungerende Keystone tjeneste med alle dens avhengigheter. Den største fordelen med denne løsningen er at man står fritt til å velge hva som skal inngå i løsningen og det kan settes opp slik at det oppfyller de kriteriere vi og oppdragsgiveren har spesifisert tidligere.

Arbeidsflyt

Selvbygde konteinere kan implementeres ved hjelp av Git og konfigurasjonsstyringsystemer slik som Puppet, der Puppet ved bruk av Git henter katalogen som inneholder Dockerfilene til de forskjellige tjenestene. Puppet kan initialisere miljøvariable og lage filene ved hjelp av Hiera. Når det kommer til lagring av Docker-images finnes det flere muligheter, blant annet lagring privat i et Docker registry eller på Docker Hub. Det ville vært ideelt å ha noe form for orkestrering av konteinere, slik at konteinere kan startes fra en Swarm Leader for eksempel. Dette også for å videreføre systemet som IIK allerede har kjørende i VM-er. Dette er viktig for å skape robusthet og redundant system slik at det ikke skader skytjenestens funksjonalitet ved å fjerne en vilkårlig node i systemet.

Oppsummering

Med selvbygde konteinere står man fritt til å velge akkurat det man ønsker utifra behov. Det gir valgmuligheter når det kommer til konteinerteknologi, arkitektur, med eller uten orkestrering og hvor imagene lagres. Det er også mulighet for å sette opp noe form for konteinerorkestrering slik som Docker Swarm eller Kubernetes for ekstra robusthet og pålitelighet.

Ulempen med å sette opp OpenStack ved bruk selvbygde konteinere er at det lages fra bunnen av igjen og dette vil kreve mye tid og ressurser, ikke bare for å sette den opp men også med å vedlikeholde det. Dette vil gjøre denne løsningen mindre praktisk for IIK. Det gir også rom for at feil, sårbarheter knyttet opp mot systemet og mulige sikkerhetshull når du bygger et slikt system fra bunnen av. Dette kan gjøre det nødvendig å utføre sikkerhetsanalyse av løsningen for å avdekke sårbarheter og sikkerhetshull.

5.2.4 Andre kandidater

Kolla-Kubernetes

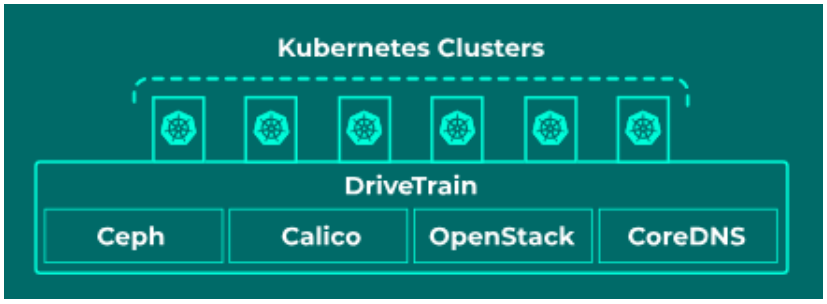
Kolla-Kubernetes er et OpenStack prosjekt som har som mål å plassere OpenStack på toppen av konteinere ved bruk av Kubernetes [49]. Siden dette prosjektet er under Kolla-familien og fordi prosjektets mål samsvarer med målet for vår oppgave valgte gruppen å se på den.

Prosjektet er lagt ned og er ikke lenger støttet. Det anbefales at man heller ser på prosjektet OpenStack-Helm. Dette prosjektet er ikke egnet for å kjøre i virtuelle maskiner selv om det kanskje fungerer i virtuelle maskiner ifølge OpenStack [50]. Gruppen har ikke viet mye tid til dette prosjektet etter at vi fant ut at den er nedlagt og ikke er støttet

lenger. Gruppen så på denne løsningen fordi den bruker Kubernetes og er et OpenStack prosjekt relatert til Kolla.

Mirantis

Gruppen har sett til Mirantis som er en kommersiell privat skyleverandør for inspirasjon til å løse problemstillingen vår etter anbefaling fra vår tidligere foreleser Kyrre Begnum og fra veilederen vår. Mirantis har en løsning der de bruker Kubernetes sammen med MCP (Mirantis Cloud Platform) [51]. Selskapet har slått sammen den allerede eksisterende MCP løsningen for å rulle ut OpenStack sammen med K8s etter krav fra markedet og sine kunder ifølge medgrunnleggeren i Mirantis, Renski [51]. Nevnte løsning skal gi mulighet for utrulling av VM-er og konteinere. Da Mirantis tilbyr kommersielle produkter er det lite detaljert informasjon vi finner om løsningen utenom arkitekturen og litt overordnet informasjon. Dette har gjort at gruppen ikke har mye å gå på og gått videre for å vurdere andre alternativer. Figur 9 viser hvordan Kubernetes brukes sammen med de andre komponentene i MCP.



Figur 9: Mirantis sin MCP kombinert med k8s kluster
[52]

Kayobe

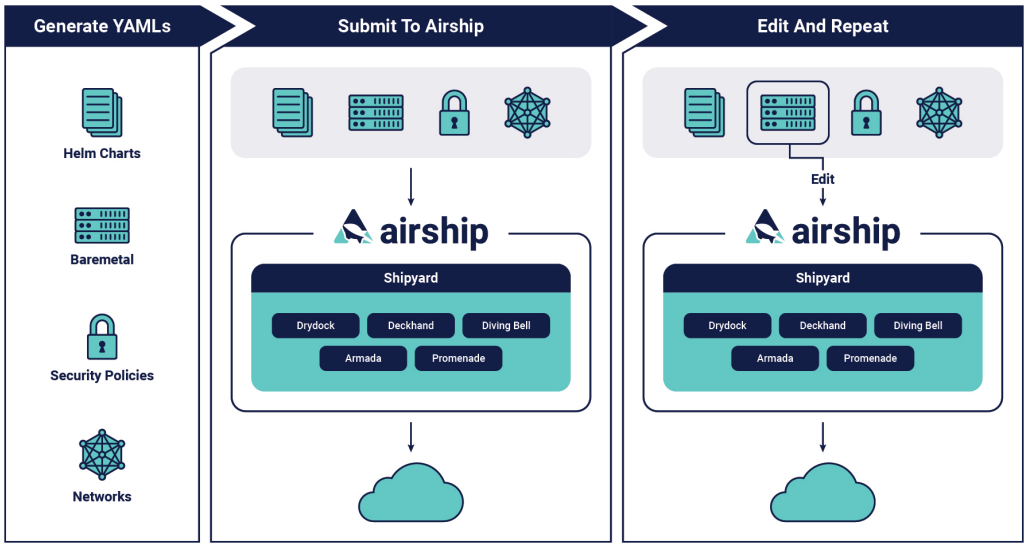
Kayobe er et annet prosjekt med hensikt å deploye konteinerisert OpenStack direkte på maskinvare [53]. Dette prosjektet er basert på Kolla og Kolla-Ansible og har nylig utgitt sin femte store utgivelse [54]. Kayobe har støtte for OpenStack-versjonen Rocky og jobber for tiden med støtte til neste versjon Stein [54]. I likhet med Kolla bruker Kayobe også Ansible for automatisering. I tillegg inkluderes det også en egen OpenStack-lignende CLI.

Problemene med Kayobe er hovedsaklig prosjektets manglende ressurser og dokumentasjon. Det har opplevdes vanskelig å finne informasjon om teknologien utenom det de selv har dokumentert. Det finnes svært lite om use-caser og brukere av Kayobe. Det nevnes også på hjemsiden deres at: *Kayobe and its documentation is currently under heavy development, and therefore may be incomplete or out of date. If in doubt, contact the project's maintainers* [53]. Utifra dette valgte vi derfor å se etter andre løsninger med bedre dokumentasjon og teknologier som hadde mer stabile utgivelser.

Airship

Airship er en samling av open-source verktøy for å deklarativt automatisere sky-providering. Airship administrerer hele livssyklusen fra bare-metal datasenter til levering av produk-

sjonsklare kubernetes klynger. Enklere sagt er hovedfokuset til Airship implementasjon og vedlikehold av OpenStack på Kubernetes gjort enklere. Det unike med Airship er at driftere gjør alt gjennom deklarative YAML dokumenter som beskriver forskjellige Airship miljøer. Airship skryter selv av at det er enkelt, repeterbart og fleksibelt. Disse punktene er alle nøkkelpunkter når det kommer til moderne drift og skyteknologier.



Figur 10: Simplifisert arkiteturskisse av Airship
[55]

Dette er et relativt nytt prosjekt, og fikk sin offisielle 1.0 utgivelse 1. Mai 2019 i sammenheng med OpenStack samlingen i Denver [56]. Til tross for dette er Airship allerede godt integrert i enkelte bedrifters prosjekter. AT&T bruker blant annet Airship til å deploye 5G mobilnett rundt omkring i USA [57]. Airship er også på vei mot å bli fundamentet til SK-Telecoms skyinfrastruktur-utrullingsprosess [58][59].

En av grunnene til at Airship ikke ble inkludert i den endelige vurderingen er at teknologien er såpass ny. Det er manglende ressurser å finne, lite treff på nettet og mangelfull dokumentasjon. Prosjektet var også ikke offisielt utgitt da vi drev research av det. Det gjorde det utfordrende for oss å vurdere på en lik linje med de andre teknologiene. En annen grunn er at Airship er et søsterprosjekt av Helm. Helm har stort potensial til å direkte løse problemstillingen og er mye mer utbredt og brukt blant bedrifter og enkelt personer enn Airship. Airship er bygget mer rundt utrulling og styring av Helm og vil ikke direkte være i stand til å løse problemstillingen, det vil fortsatt gjøres via Helm bare på en annen deklarativ måte. Det kunne vært enklere via Airship, men vi vurderer det som et ekstra system å utforske som ville gitt oss mer arbeid og kompleksitet. Sammen med at det er såpass nytt, så falt vurderingen på at vi ikke valgte å ta det med videre til endelig detaljert vurdering.

5.2.5 Resultat for initielt utvalg

Resultatet for antall løsninger for videre vurdering og testimplementering er bestemt ut fra gruppens kapasitet og oppgavens tidsbegrensning. Vi velger å gjøre utredning av tre løsninger. Kolla-Ansible, selvbygde konteinere og Openstack-Helm blir de tre løsningene som skal settes opp i en testimplementasjon og vurderes i detalj.

Oppdragsgiver ønsket at gruppen skulle se på Kolla og selvbygde konteinere som mulige konteinere og finne styringssystemer for disse. Gruppen ønsker å se på alternativer for begge konteiner-alternativene gitt av oppdragsgiver. Selvbygde konteinere kan gi images som er spesielt tilpasset IIK sitt oppsett, men har lite alternativer til eksisterende styringsverktøy. Gruppen skal bygge selvbygde konteinere og se på mulighetene for å utvikle egne metoder for styringen av konteinerne. Noen av fordelene ved selvbygde konteinere er:

- Total kontroll over hva konteinere inneholder
- Uslåelig mulighet for tilpasning av eget oppsett
- Kan potensielt integreres som en del av en større løsning eller plattform

Det finnes mange alternativer for løsninger som benytter seg av OpenStack-Kolla sine konteinere. Som et søsterprosjekt til Kolla var Kolla-Ansible den første løsningen vi fant og studerte. Det viste seg å være en god løsning som også passet kriteriene som ble definert. Noen av fordelene er:

- Opprettet av OpenStack Foundation.
- Tett koblet opp mot OpenStack-Kolla.
- Stabil markedsandel.
- Godt nettsamfunn.
- Gode alternativ til konfigurering og styring.

Den andre løsningen basert på Kolla-konteinere som gruppen har valgt å gjøre en vurdering av er OpenStack-Helm. OpenStack-Helm ble valgt fordi den oppfyller mange av kriteriene gruppen satt for initialutvalget. OpenStack-Helm har også mange unike fordeler, for eksempel:

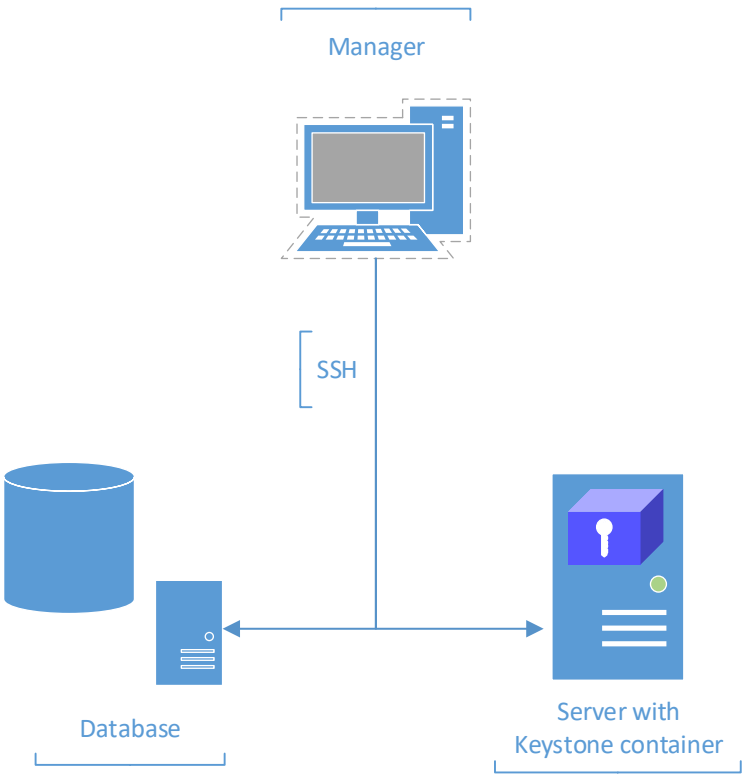
- Andre store aktører bruker den [39].
- Kan fint brukes sammen med konfigurasjonsstyringssystemer slik som Puppet.
- Opprettet av OpenStack Foundation.
- Det egner seg godt for både små og store installasjoner [60].
- Enormt potensiale og fordeler da den bruker Kubernetes, noen av disse er:
    - Gode orkestreringsmuligheter.
    - Automatisk skalering.
    - Mulig integrasjon med CI/CD verktøyer laget for konteinere slik som Codefresh.
    - Rollback ved behov.

6 Implementering av utvalgte kandidater

For å få bedre grunnlag for vurdering av de valgte teknologiene Kolla-Ansible, OpenStack Helm og selvbydge konteinere settes det opp et testmiljø der hver av alternativene skal testes ut og vurderes individuelt. Teknologien skal sette opp tjenesten Keystone i konteiner. Tjenesten må kunne konfigureres med selvdefinerte verdier og kunne settes i bruk i eksisterende miljø, dvs. bruke eksisterende database og støttetjenester som RabbitMQ og MemCached. Det er ønskelig om tjenesten kan automatisere oppsett av flere tjenester samtidig i et multinode miljø der tjenestene kjører på separate maskiner. Etter oppsett og konfigurasjon skal det testes ut hvordan utføring av driftsrutiner fungerer på teknologien. Dette vil blant annet si oppgraderinger, endring av konfigurasjon i eksisterende miljø eller innføring av nye noder eller tjenester.

6.1 Oppsett av Kolla-Ansible

6.1.1 Installasjon og konfigurasjon



Figur 11: Arkitektur i testdeployment av Kolla-Ansible

Gruppen valgte å implementere Kolla-Ansible ved å bruke Puppet. Dette er ikke nødvendig, og er kun gjort på Kolla-Ansible, ikke på de andre løsningene. For å sette opp denne løsningen brukes tre maskiner slik figuren over viser. Dette oppsettet er i tråd med Kolla-Ansible arkitekturen vist i figur 5 beskrevet tidligere i rapporten. Her brukes det en provisjonsnode som styrer utrullingen ved bruk av Ansible og Puppet. En node som kjører Keystone-konteinere og en separat maskin for database. Gruppen har satt det opp slik at Puppet installerer og setter opp all nødvendig programvare slik som Docker, Python, Ansible etc. Puppet setter også opp nødvendige konfigurasjonerfiler ved bruk av Hiera.

IIK bruker Puppet til å styre og sette opp OpenStack og dens tjenester, derfor er det naturlig å bruke dette konfigurasjonsstyringssystemet over andre alternativer da vi ønsket å teste en av løsningene i et CMS. Dette kan også demonstrere at de kan gå over til konteinerisert kontrollplan i OpenStack og fortsatt bruke sitt eksisterende Puppet-oppsett. Gruppen hadde også erfaring med Puppet men dette er av mindre betydning. Riktignok brukes Ansible fortsatt til å rulle ut konteinere, det kan gjøres deklarativt i Puppet, men på grunn av tidsbegrensninger fikk vi ikke implementert full-automatisk utrulling ved bruk av Puppet. Planen var å demonstrere bruken av konfigurasjonsstyring system for utrulling og styring av konteinere. Det finnes også Puppetmoduler for Ansible som gjør det mulig å styre Ansible med Puppet. Kodesnuttet 1 viser et utkast av Puppet-kode for managernoden i arkitekturen nevnt over 11.

Det ble konfigurert slik at provisjonsnoden kan koble seg til maskinene ved hjelp av SSH. Dette kreves for at Ansible skal kunne rulle ut konteinere på Keystonenoden, siden Ansible bruker SSH for kommunikasjon. Alle konfigurasjonsendringer skjer hovedsaklig i filene *multinode.ini*, *globals.yml*, *ansible.cfg* og *passwords.yml*, der sistnevnte trenger passordet til den eksterne databasen. I globals.yml defineres det variabler for konfigurasjon av Kolla-Ansible og tjenestene som deployes med det. I tillegg kan spesifikke tjenester videre konfigureres i */etc/kolla/config/<tjenestenavn>.conf*. Noe av det som kan konfigureres er:

- Om databasen skal være i konteiner satt opp av Kolla-Ansible eller om den skal være utenfor.
- Hvilken versjon av Keystone som skal settes opp.
- Operativsystemet til konteinerne, i hovedsak to valg her, Ubuntu eller CentOS.
- Om støttetjenester som HAproxy, RabbitMQ, MemCache skal settes opp, eventuelt adresse for eksterne oppsett av de.

Kolla-Ansible i seg selv krever ingen støttetjenester, men flere av OpenStack sine tjenester krever det, og det må tas med i betraktning ved deployment. Keystone krever en database. Kolla-Ansible kan konfigureres til å sette opp en ny database i konteinere, men dette er ikke noe som vil passe for denne deploymenten da SkyHiGh allerede har en eksisterende database. Det defineres da at ekstern database skal brukes, adressen settes inn som en variabel i globals.yml, og aksessinformasjon legges inn i passwords.yml. Repoet for å sette opp deployment miljøet ved bruk Puppet kan finnes på lenken `https://github.com/phoenix090/control-repo/tree/production`.
Repoet som setter opp stacken og installerer Puppet ved hjelp av Heat og to shell-skripter kan finnes på `https://github.com/Morten95/kollastack`.

```
class profile::manager {

   $ansible_config = lookup('ansible_config')
   $globals_config = lookup('globals_config')
   $multinode_config = lookup('multinode_config')

   package {'ansible':
      ensure   => 'latest',
   }

   python::pip { 'ansible':
      ensure   => '2.4',
   }

   python::pip { 'kolla-ansible':
      ensure   => 'latest',
      before   => File['kolla', 'inventory', '/root/kolla-ansible/etc/kolla'],
   }

   # Install dependencies
   package {['libffi-dev', 'gcc', 'libssl-dev', 'python-selinux', 'python-setuptools']:
      ensure    =>  'present',
      provider  =>  'apt',
   }

   # Installing requirements for kolla and kolla-ansible
   python::requirements { '/root/kolla/requirements.txt':
      require    => Vcsrepo['/root/kolla'],
   }
```

Listing 1: Eksempelkode fra manager.pp for oppsett av Kolla-Ansible

6.1.2 Driftsrutiner

Endringer av variabler

Endringer av variabler i Kolla-Ansible kan gjøres ved å endre på verdiene i konfigurasjonsfilene */etc/kolla/globals.yml*, */etc/kolla/passwords.yml* eller en av konfigurasjonsfilene for spesifikke tjenester, for så å bruke Kolla-Ansible kommandoer for å bake de inn i tjenesten. Dette kan enten gjøres med en phoenix-deployment hvis det er mange endringer ved å ta ned den gamle stacken med

```
kolla-ansible inventory destroy
```

og generere en ny med

```
kolla-ansible inventory deploy
```

eller ved å bruke

```
kolla-ansible inventory reconfigure
```

hvis det kun oppdateres et mindretall enkle variabler.

Oppgradering av tjenester

Ved oppgradering av tjenester i Kolla-Ansible må først Kolla-Ansible oppgraderes. Her antar vi en oppgradering til versjon 6.0.0. Selve Kolla-Ansible klientprogramvaren oppgraderes med

```
pip install --upgrade kolla-ansible==6.0.0
```

Hvis den nye versjonen har gjort endringer i databaseoppsett eller format i inventory mellom versjonene må også dette endres til nytt format. Behovet for disse endringene opplyses om i release notes fra OpenStack Foundation. Eksempler på nye formater for inventory vil også ligge i */usr/share/kolla-ansible/etc_examples/kolla* for CentOS eller */usr/local/share/kolla-ansible/etc_examples/kolla* for Ubuntu. Ønsket versjon for tjenestene som deploymenten skal defineres ved variabelen openstack_version i */etc/kolla/globals.yml*. Nye images hentes så ned med

```
kolla-ansible pull
```

Oppgradering til ny versjon kan nå gjennomføres med

```
kolla-ansible upgrade
```

Introdusering av nye tjenester og sletting av tjenester

Introdusering av nye tjenester i Kolla-Ansible gjøres ved å endre variabler i konfigurasjonsiler for å definere at tjenesten skal aktiveres, legge til en maskin tjenesten skal deployes på i inventoryfilen, og så kjøre

```
kolla-ansible reconfigure
```

eller slette deploymenten og sette opp en ny med de nye definisjonene. Hvis det skal fjernes en tjeneste fra deploymenten er det for øyeblikket ingen muligheter for å bruke Kolla-Ansible til å slette kun en tjeneste. For å gjøre dette må enten hele deploymenten tas ned og settes opp igjen etter at tjenesten er slått av i konfigurasjonsfilene og fjernet fra inventory, eller ved å manuelt fjerne endepunktet fra OpenStack og slette konteineren i Docker.

6.1.3 Feilsøking

Vi har fått ut endel verdifull informasjon ved å bruke kommandoer som

```
docker logs konteiner-navnet
```

Sjekke logs for Keystone

```
docker exec <konteiner navnet> cat /var/log/kolla/keystone/keystone.log
```

Det er også mye nyttig informasjon fra Ansible operasjonene når de kjøres med flere debug parametere.

Kolla-ansible har blant annet en post-check kommando

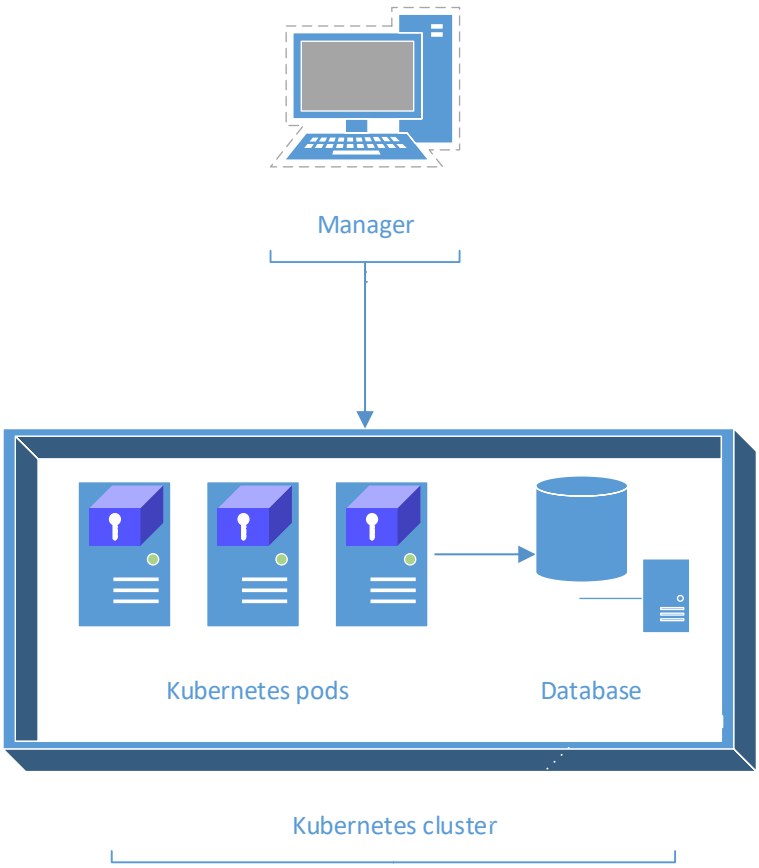
```
kolla-ansible check
```

Denne kjører noen Ansible operasjoner for å verifisere at tilstanden til de kjørende tjenestene er bra eller ikke.

Kolla-Ansible har også et greit nettsamfunn og mye hjelp å finne når det søkes etter en feilmelding.

6.2 Oppsett av OpenStack-Helm

6.2.1 Installasjon og konfigurasjon



Figur 12: Arkitektur i testdeployment av OpenStack-Helm

For å sette opp løsningen fulgte vi et lignende oppsett som installasjonen av Kolla-Ansible. En forskjell ved oppsettet her er at det ble satt opp i to forskjellige arkitekturer. Først ett All-In-One oppsett, det vil si at alle tjenestene kjører kun på en maskin. I et slikt oppsett vil all kommunikasjon mellom tjenestene være lokalt. Det utelater eventuelle problemer som åpning av porter og kommunikasjon mellom noder.

OpenStack-Helm ble også satt opp i et multinode oppsett, et mer realistisk scenario som skulle simulere noe lignende IIK sitt produksjonsmiljø. Oppsettet bestod av en mana-

gernode og to arbeidernoder. Managernoden har som oppgave å administrere clusteret, mens de to arbeidernodene har som oppgave å kjøre tjenestene. Tjenesten som ble forsøkt satt opp I dette tilfellet var Keystone. Først ble alle tjenestene forsøkt satt opp ved OpenStacks installasjonsscripts for en hel deployment uten oppfølgning, men det forårsaket mange problemer. Beslutningen ble da å kun sette opp Keystone for å forenkle prosessen.

Det er i teorien ikke noe særlig mer arbeid å sette opp flere heller en én enkelt tjeneste ettersom man bare kjører script for script. Interaksjonen mellom de forskjellige tjenestene fører derimot til betydelig større sjanse for feil i oppsettet og krav for kunnskap om hvordan tjenestene kommuniserer og avhenger av hverandre. OpenStacks egne guide gir et enkelt oppsett med kun Keystone og støttetjenester med Helm. Guiden er en enkel steg-for-steg prosedyre med en liten forklaring for noen av stegene [35].

Hvordan tjenestene blir satt opp hver for seg ved bruk av skripter:

```
./tools/deployment/multinode/010-setup-client.sh
./tools/deployment/multinode/020-ingress.sh
./tools/deployment/multinode/030-ceph.sh
./tools/deployment/multinode/040-ceph-ns-activate.sh
./tools/deployment/multinode/050-mariadb.sh
./tools/deployment/multinode/060-rabbitmq.sh
./tools/deployment/multinode/070-memcached.sh
./tools/deployment/multinode/080-keystone.sh
```

En av disse skriptene, 080-keystone.sh, ser slik ut 2:

```
#!/bin/bash
set -xe

#NOTE: Deploy command
helm upgrade --install keystone ./keystone \
    --namespace=openstack \
    --set pod.replicas.api=2 \
    ${OSH_EXTRA_HELM_ARGS} \
    ${OSH_EXTRA_HELM_ARGS_KEYSTONE}

#NOTE: Wait for deploy
./tools/deployment/common/wait-for-pods.sh openstack

#NOTE: Validate Deployment info
helm status keystone
export OS_CLOUD=openstack_helm
sleep 30 #NOTE(portdirect): Wait for ingress
controller to update rules and restart Nginx
openstack endpoint list
helm test keystone --timeout 900
```

Listing 2: Eksempelskript som brukes for oppsett av tjenester i OpenStack-Helm

Før disse scriptene var kjørt ble det manuelt installert Helm ved hjelp av en annen litt mindre guide fra OpenStack. Deretter var det å generere og konfigurere en inventory-fil

og en environment-fil for også kjøre dem med *make*. Make er en type støtteverktøy for å bygge eller opprettholde grupper av programmer eller filer fra kildekode. Henter Helm fra Git.

```
git clone https://git.openstack.org/openstack/openstack-helm-infra.git \
/opt/openstack-helm-infra
git clone https://git.openstack.org/openstack/openstack-helm.git \
/opt/openstack-helm
```

Ansible inventory-filen som OpenStack-Helm bruker for å sette opp et enkelt Kubernetes kluster ser slik ut 3:

```
#!/bin/bash
set -xe
cat > /opt/openstack-helm-infra/tools/gate/devel/multinode-inventory.yaml
<<EOF
all:
children:
    primary:
    hosts:
        node_one:
        ansible_port: 22
        ansible_host: $node_one_ip
        ansible_user: ubuntu
        ansible_ssh_private_key_file: /etc/openstack-helm/deploy-key.pem
        ansible_ssh_extra_args: -o StrictHostKeyChecking=no
    nodes:
    hosts:
        node_two:
        ansible_port: 22
        ansible_host: $node_two_ip
        ansible_user: ubuntu
        ansible_ssh_private_key_file: /etc/openstack-helm/deploy-key.pem
        ansible_ssh_extra_args: -o StrictHostKeyChecking=no
EOF
```

Listing 3: Kodeeksempel for inventory-fil i OpenStack-Helm

Enviroment-filen fra vårt kluster ser slik ut 4:

```
#!/bin/bash
set -xe
function net_default_iface {
 sudo ip -4 route list 0/0 | awk '{ print $5; exit }'
}
cat > /opt/openstack-helm-infra/tools/gate/devel/multinode-vars.yaml
<<EOF
kubernetes_network_default_device: $(net_default_iface)
EOF
```

Listing 4: Eksempelkode fra Helm environment-fil

I motsetning til Kolla sitt oppsett ble det ikke installert noe ved bruk av Puppet. I dette tilfellet forekom det ikke noen nødvendige avhengigheter som OpenStack-Helm

sin installerings og oppsett-guide ikke håndterte. Tjenestene settes opp i arbeidernodene i utgangspunktet, da Kubernetes sin standard er slik siden den kun ønsker å kjøre K8s kontrollplanet i K8s-mesteren. Det er mulighet for å styre antall instanser av OpenStack-tjenestene med K8s.

6.2.2 Driftsrutiner

Endringer av variabler

OpenStack-Helm sine charts bruker verdier fra fra en fil kalt *values.yaml* for å konfigurere en tjeneste. Hver tjeneste som allerede er ferdig implementert av OpenStack-Helm sine utviklere kommer med en slik fil i hver chart. Verdiene kan for eksempel være Keystoneversjonen som benyttes, gruppe- og brukerrettigheter, porter og diverse passord som adminpassordet for Keystone. Disse verdiene kan endres via denne filen.

Dersom mange av disse verdiene skal endres er det anbefalt å lage en ny variabelfil som overskriver verdiene fra *values.yaml*. Denne legges under mappen *tools/overrides/mvp* [61]. Filen navngis *<tjenesten>.yaml*. Når tjenesten rulles ut vil denne filen under mvp-mappen brukes slik det står i OpenStack dokumentasjonen [61]. Gruppen har ikke testet dette eller oppgradering av tjenesten Keystone under implementasjonen, da gruppen møtte på utfordringer under utrullingen av OpenStack-Helm. For mer utfyllende beskrivelse av dette, se vedlegg B.

Oppgradering av tjenester

Som nevnt rett ovenfor har vi ikke hatt anledning til å teste oppgradring av tjenester i OpenStack-Helm, da det ble støtt på utfordringer under deployment av multinode-miljø. Beskrevet nedenfor er prosessen for oppgradering av tjenester i OpenStack-Helm slik det forekommer i følge dokumentasjonen til OpenStack-Helm.

OpenStack-Helm forventer at alle oppgraderinger skjer via pakkehåndteringsverktøyet Helm. Dersom det gjøres endringer på en chart eller konteinerimaget, slik som filer, skripter og lignenede, resulterer det i en Kubernetes rolling update. Dette innebærer at oppgraderinger eller endringer blir utført uten at dette medfører å ta ned tjenesten, for så å rulle den ut igjen. OpenStack-Helm sjekker jevnlig etter endringer, og kjører rolling update uten noen menneskelig involvering. Det finnes per i dag feil og mangler når det kommer til oppdatering av konteinerimages som allerede er i bruk, som OpenStack-Helm jobber aktivt med å forbedre [62].

Introdusering av nye tjenester og sletting av tjenester

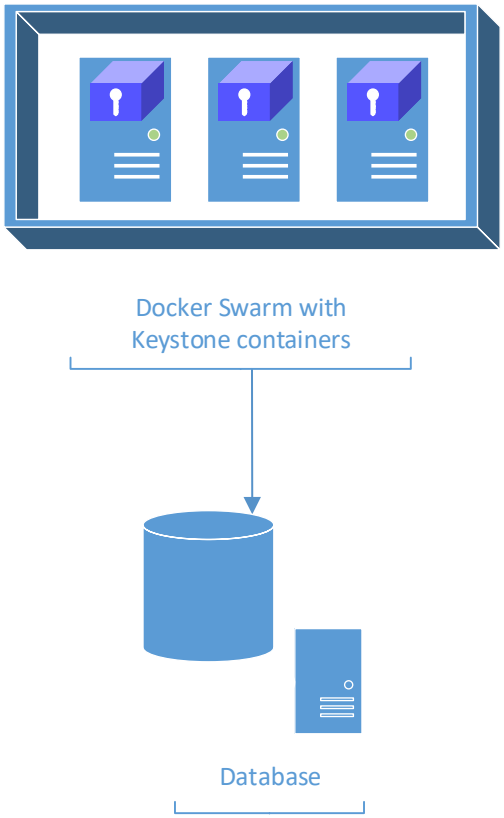
I OpenStack-Helm repoet har utviklerne laget charts for de fleste OpenStack tjenestene. Utrulling av disse utføres med skript som har følgende format NNN-service-name.sh, der NNN står for rekkefølgen skriptet kjøres i forhold til de andre tjenestene [61]. Per OpenStack-dokumentasjonen lages det en ny chart for å kunne introdusere en ny tjeneste. Et script for kjøring av denne chart-en legges til i ./tools/deployment/multinode. Chart'en bør også inkluderes i gates-kolleksjonen. Dette slik at utvikleren og gruppen finner ut eventuelle feil med den nye tjenesten [61].

6.2.3 Feilsøking

Det er ikke enkelt å finne løsninger for feil i OpenStack-Helm av to grunner. Den første er at feilmeldingene ikke gir noen gode indikasjoner på hva som har gått galt eller i hvilket ledd feilen har oppstått. Den andre grunnen er at det ikke finnes mange åpne forumer hvor feilmeldinger blir diskutert. Dette har gjort det frustrerende for gruppen å finne ut av feil. Feilsøkingsprosessen gruppen brukte ved problemer i OpenStack-Helm er dokumentert i vedlegg B, Feilsøking av Openstack-Helm.

6.3 Oppsett av selvbygde konteinere

6.3.1 Installasjon og konfigurasjon



Figur 13: Arkitektur i testdeployment av selvbygde konteinere

Gruppen har gjort forskning på hvordan andre implementerer Keystone i konteiner fra bunnen av, samtidig sett kodebasen til oppdragsgiveren for hvordan de idag setter opp denne tjenesten ved hjelp av Puppet (ntnusky-repoet). Keystone settes opp ved bruk av en Dockerfile og et bootscript for å bygge keystoneimaget og sette opp nødvendige programvareavhengigheter og konfigurasjonsfiler. Det tas i bruk miljøvariabler for å gjøre noen konfigurasjoner dynamiske. Dette for å slippe å endre selve Dockerfilen eller scriptet hver gang en endring av disse variablene er nødvendig. Noen eksempler på disse

miljøvariablene er IP- adressen til databasen og Keystoneversjonen som skal brukes.

Det brukes og en Docker Compose-fil for også å rulle ut Memcached samtidig i en annen konteiner. Vi har implementert Docker Swarm for bedre robusthet og skalering av tjenestene. Dette er også en av grunnene til at vi bruker Docker Compose, da dette kan rulles ut som en stack ved bruk Docker Compose. Ved bruk av Docker Compose blir det i tillegg da brukt Docker Registry for å lagre Docker images slik at de kan brukes av alle nodene i Docker Swarm klusteret. Oppsettet av selvbygde konteinere ligger på GitHub under lenken https://github.com/phoenix090/skyhigh_containerization.

```
services:
  keystone:
    image: 127.0.0.1:5001/keystone
    build:
      context: keystone-microservice/keystone/
      args:
         KEYSTONE_VERSION: ${KEYSTONE_VERSION}
         KEYSTONE_REMOTE_DB_PASSWD: ${KEYSTONE_REMOTE_DB_PASSWD}
    ports:
      - 5000:5000
      - 35357:35357
    volumes:
      - fernet:/etc/keystone/fernet-keys/
    env_file:
      - .env
    container_name: keystone
  memcached:
    image: memcached:latest
    ports:
      - 11211:11211
    container_name: memcached
volumes:
    fernet:
```

Listing 5: Utkast av docker-compose

For å rulle ut Keystone sammen med Memcached som en stack i Docker Swarm klusteret kjøres:

```
docker stack deploy --compose-file docker-compose.yml keystone_stack
```

6.3.2 Driftsrutiner

Endringer av variabler

Når Docker imagene lages bruker Keystoneimaget miljøvariabler for å bestemme noen konfigurasjoner. Eksempler på disse er versjonen som benyttes, databasepassordet, IP-adressen til den eksterne databasen o.l. Miljøvariablene er satt opp for å kunne endre Dockerfile enkelt, som gir fleksibilitet og gjør tjenesten mottagelig for endringer. Derfor har gruppen laget en miljøvariabelfil for disse konfigurasjonene, slik at enkle endringer kan utføres på denne filen. Et eksempel på dette kan være fremtidige oppdateringer av Keystoneversjonen. Det kan legges til flere variabler i denne filen om nødvendig.

Oppdatering av tjenester

For å oppdatere Keystoneversjonen endres en miljøvariabel, deretter oppdateres imaget ved at den bygges på nytt. Kommandoene under bygger imagene på nytt, pusher til Docker Registry, deretter deployer stacken til Swarm-klusteret:

```
docker-compose up -d --build
docker-compose push
docker stack deploy --compose-file docker-compose.yml keystone_stack
```

I GitHub representerer versjonsnummeret en numerisk utslippsversjon av en tjenesten. Dette versjonsnummeret kan brukes til å hente en spesifikk release eller versjon av Keystone ved bruk av denne kommandoen:

```
RUN git clone -b ${KEYSTONE_VERSION} https://github.com/openstack/key
stone.git
```

Dette kan innføres for alle tjenestene i OpenStack slik at oppdateringen går fort og effektivt.

Introdusering av nye tjenester og sletting av tjenester

For å introdusere nye tjenester for selvbygde konteinere slik som Horizon, Glance eller lignende, er det mange måter å løse dette på når det kommer til hvordan det skal implementeres. Dersom fremgangsmåten gruppen har brukt for å sette opp Keystone følges, blir det å lage egen mappe for den nye tjenesten. Deretter lage Dockerfile og tilhørende konfigurajsonsfiler og eventuelle scripter. Etter dette innføres en ny service i Docker Compose for tjenesten. Det kan også bli nødvendig å lage et Docker-volum for tjenester ved replikering, slik at flere instanser av samme type deler lagret data.

6.3.3 Feilsøking

For å sjekke feilmeldinger knyttet til en konteiner, kan følgende kommando benyttes:

```
docker logs <konteinerID eller navn>
```

Dette viser loggen til en konteiner, som kan inneholde feilmeldinger og annen nyttig output fra kjørende program. Keystone er blant de tjenestene som bruker webserveren Apache2 til å håndtere forespørsler inn mot tjenesten. For å feilsøke Apache2, finnes det logfiler for de tjenestene webserveren eksponerer, i dette tilfelle Keystones API-endepunkter. Her loggføres alle forespørsler inn mot Keystone, som det er mulig å sjekke for feilmeldinger knyttet til Apache2. Det er også mulig å sjekke tilstanden til Keystone-konteineren ved bruk av kommandoen:

```
docker exec <konteinernavnet> keystone doctor
```

Selvbygde konteinere er etter gruppens erfaring den enkleste å feilsøke da vi har utviklet den selv. Her har gruppen en mer detaljert oversikt over hvordan tjenestene settes opp, tas ned og endres sammenlignet med eksterne prosjekter utviklet av andre.

7 Detaljert vurdering av implementerte teknologier

I de to forrige kapitlene har vi gått igjennom initielt utvalg 5 av aktuelle kandidater for å løse problemstillingen til oppgaven, samt gjort en testimplementasjon av dem 6. Med tanke på tidsomfanget er prosjektgruppen nødt til å forhold seg til de tre utvalgte løsningene fra kapittel fem, under punkt 5.2.9. Disse tre løsningene er OpenStack-Kolla, selvbygde konteinere og OpenStack-Helm. I dette kapittelet skal gruppen utføre en dypere vurdering av disse kandidatene. Vurderingen vil først og fremst basere seg på om den vurderte løsningen effektiviserer allerede implementert løsning. Det vil kun gjøres detaljert vurdering av de tre implementerte teknologiene ettersom tidsbegrensninger ikke tillater dypdykk i mer enn disse tre relevante teknologier.

Løsningene vil aldri tilfredstille alle krav, så det defineres vurderingskriterier for å kunne sammenligne løsningene opp mot hverandre og mot eksisterende løsning. Til slutt gjør gruppen en kvantitativ vurdering av alle tre løsningene. Vurderingene som gjøres i dette kapittelet vil veie tungt i den endelige vurderingen av kandidatene. Det vil bli valgt den løsningen som egner seg best for IIK og ikke nødvendigvis den med best funksjonalitet.

7.1 Vurderingskriterier

Gruppen gjennomfører en vurdering av de best egnede teknologiene ved å bedømme de på noen spesifikke og relevante kriterier i forhold til fagområdet. Tidligere har Red Hat Enterprise gjort en lignende vurdering da deres oppgave også var å konteinerisere OpenStacks kontrollplan og tjenester [63]. Denne vurderingen involverte også Helm-baserte og Ansible-baserte teknologier, som OpenStack-Helm, Kolla-Kubernetes og Kolla-Ansible.

For å få utredet en helhetlig vurdering av disse teknologiene brukte Red Hat relevante kriterier for å bedømme best egnet teknologier. Disse kriteriene bestod hovedsakelig av: Voksende og aktivt samfunn, simplisitet, fleksibilitet, og enkel adopsjon av teknologien. Gruppen tok utgangspunkt i disse kriteriene og kravene fra oppdragsgiver ved starten av utvalget av kriterier. For å ende opp med kriterier som er relevant for oppgavebeskrivelsen og problemstillingen blir kriteriene valgt ut fra disse prinsippene:

- Tidligere relevante vurderinger som Red Hat sin vurdering.
- Drøfting med oppdragsgiver om hva de er ute etter.
- Prosjektgruppens dømmekraft basert på vår kompetanse, forståelse for oppgaven og kunnskap opparbeidet om området i løpet av researchen.

Kriteriene er sett på som svært viktige for at valgt teknologi skal oppfylle oppdragsgivers krav og ønsker. De utvalgte kriteriene er

- Effektivitet
- Kompleksitet
- Popularitet og trender

- Sikkerhet

Løsningene vil bli vurdert på hver av disse punktene individuelt i tillegg til en oppsummering av nøkkelfunnene, samt styrker og svakheter for hver løsning. Det vil også gjøres en kvantitativ sammenligning av de tre utvalgte teknologiene i henhold til de satte kriteriene.

7.1.1 Effektivitet

Her blir det lagt fokus på hvor effektiv løsningen er å drifte for IIK. Å vurdere det fra dette perspektivet anses som hensiktsmessig basert på oppgavens fagområde. Noen nøkkelpunkter som gruppen vil bruke for å måle dette er som følger:

- Hvor effektiv løsningen som vurderes er når det kommer til oppgraderinger av både OpenStack og programvare knyttet til OpenStack-tjenestene. OpenStack har to utgivelser hvert år, derfor er det viktig å se hvordan løsningen håndterer disse oppgraderingene. Av det oppdragsgiveren har fortalt oss, er denne prosessen per idag tidkrevende, knotete og uoversiktlig. Dette er ønsket at teknologien løser.
- Hvor effektivt kan den daglige driften utføres? Dette er også et problem som gruppen må adressere og finne en bedre løsning på for IIK.

7.1.2 Kompleksitet

Dette kriteriet er en vurdering av kompleksiteten av løsningen i forhold til prosjektoppgaven og ikke en vurdering av kompleksitet i konteineriseringsteknologi generelt. I prosjektoppgavens tilfelle har oppgaven i seg selv en relativ kort tidsperiode og oppdragsgiver har et lite team som ikke er så godt kjent med konteinere fra før av. Sett i et globalt perspektiv er også infrastrukturen forholdsvis liten. Sett fra et annet ståsted hvor ressurser og antall driftere ikke er en betydelig faktor ville det vært enklere å veie effektivitet over kompleksitet, men det er ikke tilfelle i vårt scenario. Kompleksitet vil derfor være vell så tellende effektivitet i den endelige vurderingen.

Nøkkelpunkter for vurdering av kompleksitet er:

- Hva slags kunnskap som kreves om teknologier og konsepter for å kunne forstå hvordan tjenestene fungerer og hvordan filstrukturen er satt opp.
- Hvor avanserte og omfattende prosessene og kommandoene er for oppsett av tjenestene.
- Hvor avanserte og omfattende prosessene og kommandoene er for driftsrutiner og konfigurasjonsendringer.
- Hvor mye det er å sette seg inn i og hvor stor forståelse av tjenestene og dets variabler det kreves for å kunne konfigurere tjenestene.

7.1.3 Popularitet og trender

Popularitet og trender er enda et viktig kriterium som forteller om det er hjelp tilgjengelig og om prosjektet har en sunn status for å hindre at det mister støtte. Det første delkriteriet for dette er størrelsen på brukerbasen og utbredelse i teknologibedrifter. Andre vurderinger i dette kriteriet er hvor ofte prosjektet blir vedlikeholdt og oppdatert, og hvor god hjelp det er å finne i brukerstøtte eller hjelpeforumer. Nevnte punkter gir en indikasjon på hvor lenge prosjektet kommer til å være støttet og om prosjektet er i vekst.

Det er også viktig for IIK med godt forum, bugtracks eller lignende å vende seg til når de støter på problemer. Dette vil minske sannsynligheten for at de blir de første til å bli sittende fast med en feil eller et problem uten å finne noen løsning. Det vil skape vanskeligheter for ansatte når det kommer til feilsøking av problemer, og i noen tilfeller kan løsningen på helt enkle problemer være svært utfordrende å komme frem til. Dette kan være kritisk om tjenester går ned uten noen enkel måte å rulle de ut igjen på. Et aktivt miljø for læring, videreutvikling og erfaringsdeling tyder på at løsningen vil ha god støtte over tid og være stabil.

Nøkkelpunkter for vurdering av popularitet og trender

- Hvor aktivt er nettsamfunnet til teknologien. Er det stadig nye løsninger rundt nye problemer eller er informasjonen utdatert?
- Hvor stort er nettsamfunnet. Hvis det finnes flere treff og alternative løsninger på problemer øker det sannsynligheten for å finne hjelp som passer til spesifikke situasjoner og ikke bare generelle tips. Dette inkluderer litteratur, blogger, innlegg og lignende.
- Hvordan brukes løsningen i relevante bedrifter? Store teknologibedrifter er ofte tidlig ute med å bruke teknologi og er kritiske bidragsytere til OpenStack sin utvikling.

Hvordan andre og mer erfarne profesjonelle aktører løser problemet med å konteinerisere OpenStack og ikke minst hva slags løsning de tar i bruk er en god indikasjon for oss. Dette kan ikke følges slavisk ettersom det ikke nødvendigvis egner seg til størrelsen på IIKs infrastruktur og behov.

7.1.4 Sikkerhet

Sikkerhet er alltid en viktig faktor i oppgaver hvor protokoller skal kjøres, porter skal åpnes og kommunikasjon skal gjennomføres. NTNU holder på sensitiv forskning- og studentinformasjon, så sikkerhet er da et helt nødvendig kriterie. OpenStack-Helm og Kolla-Ansible er profesjonelt utviklede styringsverktøy for OpenStack-tjenester og har dermed naturligvis blitt utviklet med sikkerhet i fokus. Vi vil foreta noe detaljert vurdering på programvaresikkerheten i disse teknologiene.

Konteinersikkerhet har kjernefaktorer som er felles for alle de vurderte alternativene. Disse er derfor nevnt her, og er relevante for vurderinger opp mot eksisterende løsning. Eventuelle elementer ved konteinersikkerhet relevant for spesifikke alternativer blir nevnt i de individuelle vurderinger.

Konteinertjenester har i seg selv både positiv og negativ påvirkning av sikkerheten i et miljø. Mange tjenester vil trenge å få inn data, og konteinere er designet 'med en åpen frontdør' for at den skal kunne motta det av data som er tiltenkt den[64][65]. Konteinere i seg selv har veldig lite beskyttelse fra operativsystemet. Dette fører til at det er mange typer angrep som kan være utfordrende å håndtere fra et sikkerhetsperspektiv[66]. En positiv side er at isoleringen i en konteiner fører til en redusert angrepsflate. Porter er lukket med mindre de er spesifisert til å være åpne, og støttetjenester som kan være po-

tensielle svakheter er fraværende.

En konteiner har ikke utfordringer med å kunne bruke feilkonfigurert SSH eller brukerkredentialer på avveie som fører til uønsket tilgang [67]. Dette kan derimot fortsatt være en trussel ved at maskinen der konteineren kjører kan bli kompromittert.

Siden konteinere har lite beskyttelse innebygd, vil sikkerheten avhenge av hvordan konteineren er bygget og hva den er satt til å gjøre. Sikkerheten som må tenkes på er ting som åpning av porter, eventuelle kjente sikkerhetssvakheter med teknologien, kodekvalitet, tildeling av roller og rettigheter, isolering og segmentering og hva som velges å inkludere i imaget. Det burde også følges Docker best practices da det gjelder konteinersikkerhet[68]

Nøkkelpunkter for vurdering av sikkerhet er:

- Svekker løsningen den allerede implementerte sikkerheten i SkyHiGh?
- Er komponenter isolert eller kan en angriper foreta lateral bevegelse?
- Har teknologien kjente svakheter og gir den rimelige handlings og adgangsrettigheter?
- Er konteinerne bygget på et sikkert image som følger beste standard for konteinersikkerhet eller er fra en pålitelig kilde?

7.1.5 Oppsummering

En oppsummering av alle kriterium for å skaffe et helhetlig og oversiktlig bilde. Under oppsummering skal det vurderes styrker og svakheter, ikke bare i forhold til teknologien, men også i forhold til prosjektoppgavens mål. En løsning kan være tilnærmet feilfri, men likevel ikke egne seg til formålet for oppgaven. Det er viktig å få oppsummert nøkkelpunktene til teknologien slik at man raskt kan sammenligne disse punktene opp mot hverandre. Dette gir leseren en kort og lettfattelig vurdering.

7.1.6 Sammenligning

Etter at hver løsning er vurdert hver for seg, gjøres det en kvantitativ vurdering i forsøk på å gjøre en så god helhetlig vurdering som mulig. Dette blir gjort helt til slutt i kapittelet. Det vil bli brukt en skala for å sette tall på hvor godt hver løsning oppfyller hvert kriterium. Denne skalaen går fra en til seks. Dette skal bidra til å fatte en beslutning om anbefalt løsning.

7.2 Kolla-Ansible

7.2.1 Effektivitet

Kolla-Ansible har funksjonalitet som kan spare mye tid for personalet som drifter OpenStack. Grunnet det faktumet at å sette variabler er det eneste som trengs, og tjenesten startes ved en enkelt kommando, så er prosessen forkortet flere steg. Denne reduksjonen er spesielt sterk hvis flere tjenester settes opp i samme Kolla-Ansible deployment. Det er noe arbeid ved å sette opp Python, Ansible og Kolla som prerekvisitter for hver enkelt unik installasjon av tjenesten. Hvis et oppsett med mange individuelle installasjoner brukes kan det med fordel brukes et skript for oppsett av disse.

Ved oppdateringer eller endringer i tjenesten vil også prosessen være forkortet i forhold til tradisjonelt oppsett. Konfigurasjonsendringer gjennomføres raskt ved å gjøre endringer på variabler i konfigurasjonsfiler for så å enten gjennomføre sletting av gammelt oppsett og følge det opp med en ny deploy-kommando, eller ved å gjennomføre en rekonfigurering Kolla-Ansible. Det er noen støttetjenester i Kolla som er som standard, som det må spesifiseres om ikke skal bygges. Dette er blant annet Memcached, RabbitMQ og database. Siden dette er tjenester som IIK allerede har kjørende og ikke ønsker at skal være en del av konteineriseringprosessen er dette et ekstra steg som må tas i deploymenten av Kolla. Siden det er en variant med ekstern database som vil være aktuelt for IIK, så er det ingenting som blir rullet ut av Kolla-Ansible-installasjonen som må lagres. Et tilfelle som vil kreve ekstra arbeid er hvis en ny versjon har endret hvordan databasen skal være utformet, da dette krever at databasen er endret på forhånd for at den skal fortsette å ta i bruk samme datasett. Dette er endringer som vil være annonsert i god tid, slik at vedlikeholdere kan ta stegene som trengs for å klargjøre databasen.

Introduksjon av nye tjenester er også effektivt med Kolla-Ansible, da det er en veldig lik prosess som det å gjøre konfigurasjonsendringer eller oppdateringer. Siden det også må defineres i inventoryfilen hvor den nye tjenesten skal installeres, så er denne prosessen noe mer omfattende, spesielt om maskinen den nye tjenesten skal settes opp på ikke er med i oppsettet fra før av.

Skalering med Kolla-Ansible kan gjennomføres til en viss grad. Det kan defineres flere maskiner for hver tjeneste, og hver av maskinene vil få en instans av tjenesten. Flere instanser av én tjeneste kan ikke settes opp på samme maskin. Flere forskjellige tjenester kan derimot settes opp på én maskin. Det er ingen mulighet for automatisk skalering og utrulling utifra antall noder ønsket, alt må defineres i inventory for hvert enkelttilfelle. Kolla-Ansible vurderes derfor til å fungere godt i et lite til middels stort miljø, men alternativer burde vurderes i større miljøer.

7.2.2 Kompleksitet

Det er veldig lite kompetanse som kreves for bruken av Kolla-Ansible i seg selv. Oppsettet krever bare et par kommandolinjer. Alle handlinger som gjøres i selve tjenesten er kommandoer som kun krever en generell forståelse av hvordan Kolla-Ansible fungerer og hva konsepter som prechecks og deployment går ut på. Ved at Ansible-kommandoer kan brukes for å sende instrukser til alle maskinene i inventoryen, for eksempel for å sjekke oppetid, økes kompleksiteten noe ved at det vil kreve kunnskap om hvordan Ansible fungerer.

Konfigurasjonen i seg selv er enkel, men det krever mye kunnskap om hvilke variabler som finnes, hva de gjør, og hva som er gyldige verdier. Mye av dette er standard OpenStack konfigurasjon, men det er også en del andre konfigurasjoner. Spesielt ved høyt krav om kontroll og tilpasning av tjenester vil dette kreve god kunnskap om Kolla-Ansible og om OpenStack generelt.

Håndtering av inventory er et annet element som også kan ha noe kompleksitet og krav

om forhåndskunnskap og forståelse, i form av oppsett av inventory-fil, kommunikasjonsmuligheter mellom maskiner, og eventuelle DNS-utfordringer.

7.2.3 Trend / popularitet i bransjen

OpenStack Kolla og Kolla-Ansible har mange av de samme utviklerne, og i stor del et felles community. Kolla og Kolla-Ansible brukes aktivt i flere store bedrifter som beskrevet i kapittel fem. I disse miljøene er Kolla-Ansible som oftest enten en del av en større infrastruktur av skytjenester, eller supplementert av tjenester som OpenStack Senlin for skalering. Prosjektet er også godt utbredt blant mindre firmaer og i private prosjekter på grunn av Kolla-Ansibles mål om å forenkle utrullingen av OpenStack uten krav om ekspertise [69].

Hver gang gruppen hadde problemstillinger med Kolla-Ansible var det søkeresultater tilgjengelige for å hjelpe. Utviklerne av Kolla og Kolla-Ansible er tilgjengelige i en IRC-kanal. Det finnes over 1000 Kolla images på Docker hub [70], med flere av de på over 100000 nedlastinger. Nye images legges stadig ut. Mange av de nyeste imagene er oppdatert i løpet av det siste døgnene.

Brukerbasen og nettsamfunnet til OpenStack-Kolla og Kolla-Ansible oppleves relativt godt, men veksten framstår noe redusert den siste tiden. Dette skyldes mest sannsynlig veksten av Kubernetes og dets relaterte OpenStack-prosjekter, deriblant OpenStack-Helm og OpenStack-Airship, som Kolla-Ansible har mistet en andel til. Kolla har derimot fortsatt en god posisjon innen OpenStack-miljøet, og deltar med presentasjoner på OpenStack Summit, senest i Denver i april-mai 2019.

7.2.4 Sikkerhet

Sikkerheten i Kolla-Ansible er avhengig av sikringen av maskinen der konfigurasjonsfilene ligger. Denne maskinen har ikke noe redusert sikkerhet i forhold til en OpenStack implementasjon uten konteinere. Det er en sikkerhetsutfordring at denne maskinen trenger å kommunisere med de resterende nodene via SSH. Den kan dermed utgjøre en trussel som potensiell angrepsvektor mot andre mål om den skulle bli kompromittert. Spesielt i et oppsett hvor det styres flere tjenester i én installasjon av OpenStack kan det dermed kreve spesiell god sikkerhet på denne enheten.

En annen potensiell trussel er svakheter i Kolla-Ansible. Det har ikke vært funnet noe som tyder på at det er svakheter i den nåværende versjonen, men som tidligere nevnt har vi ikke gjort noen detaljert analyse av programvaresikkerheten så vi baserer oss kun på hva som ligger på nettet.

Kolla-Ansible bruker images fra Kolla-prosjektet, som vurderes som en god og pålitelig kilde, så konteinersikkerheten burde ikke være noen trussel utover det som allerede er diskutert.

7.2.5 Oppsummering

Med god kontroll på standard implementasjon av OpenStack så er det veldig enkelt å sette opp Kolla-Ansible. Det krever nesten ingenting annet enn å sette opp en Ansible

inventory, sette inn ønsket konfigurasjon i OpenStack, skaffe avhengigheter, og å kjøre precheck->bootstrap->deploy med Ansible. Bruken av OpenStack sine egne konfigurasjonsfiler for tjenestene gir meget gode muligheter for konfigurasjonsendringer for å tilpasse ønsket oppsett. Prosjektet har ingen ekstra sikkerhetsutfordringer i forhold til konkurrenter. Brukerbasen er sunn og aktiv, men veksten virker noe redusert i senere tid, da det har vært stort fokus på Kubernetes-fokuserte konkurrenter. Målet til Kolla-prosjektet er å lage produksjonsklare konteinerimager, slik at overgangen fra VM'er til konteinere blir enklere [71]. Ved å gjøre tilgjengelig produksjonsklare konteinere er mye av arbeidet med konteineriseringsprosessen gjort. Disse imagene er derimot også tilgjengelige for bruk i andre prosjekter, så selv om de er utviklet av Kolla så kan de brukes i andre løsninger.

7.3 Openstack Helm

OpenStack-Helm er den mest utbredte blant store aktører [39], men samtidig den mest komplekse løsningen av de utvalgte. Dette gir noen interessante vinkler på vurderingen.

7.3.1 Effektivitet

OpenStack-Helm som Kolla-Ansible har funksjonaliteter som vil spare tid for personalet. Selve oppsettet av en standarisert eksempelinfrastruktur av OpenStack-Helm er rask å gjennomføre. Dette gjøres ved å kjøre scripts som installerer både Helm og OpenStack-Helm, deretter kan man installere tjeneste for tjeneste etter behov ved flere scripts. NTNUs IIK ønsker ikke å konteinerisere hele den eksisterende arkitekturen sin, men heller konteinerisere spesifikke tjenester som kan integreres i deres eksisterende arkitektur. Metoden for oppsett av tjenester i OpenStack-Helm egner seg godt for dette.

OpenStack-Helm har flere forskjellige egenskaper som gjør den meget effektiv og unik. OpenStack-Helm tilbyr blant annet basis avhengighetsadministrering. Her kan det spesifiseres hvilken rekkefølge tjenester skal bli rullet ut. Dette har sin verdi når det kommer til oppgraderinger eller når en ny infrastruktur skal settes i drift. Driftere vil enkelt ha oversiktlig og god kontroll på avhengigheter og rekkefølge. Avhengighetene spesifiseres normalt i *requirements.yaml* filen. I filen spesifiseres hovedsakelig noen påkrevde variabler; *name*, *version* og *repository*. I tillegg til disse variablene kan det også spesifiseres noen valgfrie variabler som: *alias*, *tags* og *conditions*.

Helm-toolkit er en annen god egenskap som virker som et slags delt bibliotek for OpenStack-Helm. Det vil si en samling av verktøy som kan bli brukt til konfigurering. Helm-toolkit gir mulighet til effektiv administrering av endepunkter, manifester, scripts og andre verdifulle nytteverktøyer. Disse funksjonene er svært nyttig for effektiv drift. Endepunkter kan spesifiseres kodelinje for kodelinje. Manifester kan lages for fulle utrullinger som holder standard format slik at det kan bli gjenbrukt. Det er og mulighet for gjenbruk av shellscripts til bruk av for eksempel installering eller oppdateringer. Disse forskjellige funksjonalitetene kan bli konfigurert i *values.yaml*, der man kan for eksempel spesifisere akkurat hvilke endepunkter man vil ha, og i tillegg delegere spesifikke roller og oppgaver. Det er viktig at verdiene i values.yaml er overskrivbare slik at alt kan bli endret på en enkel og sentralisert måte. Dette med hensikten av at driftere ikke behøver å re-enginere Helm-chartsene hver gang en skal gjøre endringer til deployments. Det fører til stor øk-

ning av fleksibilitet og konsistens.

Bruken av Helm-charts regnes som en god egenskap. Det hjelper med å definere, installere og oppgradere deploymenten. Charts kan kommunisere med hverandre ved for eksempel at en chart skal være avhengig av en annen. OpenStack-Helm som nevnt tidligere er bygget oppå Kubernetes. Det gir en god mulighet for skalering og automatisering av noder, hvor man enkelt beskriver hvor mange pods en node skal ha og den noden vil opprettholde det antallet ved hjelp av kuberneteskomponenten som blir kalt scheduler.

Da det kommer til mer spesifikke relevante use-caser for IIK som oppgraderinger, vil prosessen gjennom OpenStack-Helm være forkortet i forhold til dagens rutine. I OpenStack-Helm som i Kolla-Ansible spesifiseres OpenStack-versjonen i en egen fil. Her vil det være muligheter til å kun endre en variable fra for eksempel Rocky til Stein, for og så rulle ut podsene på nytt med den nye versjonen. Da det kommer til implementering av nye tjenester finnes det allerede scripts fra OpenStack som kan installeres enten med standard variabler som er satt, eller endre dem til eget behov. Installering av nye tjenester er enkelt og meget effektivt.

7.3.2 Kompleksitet

Selve kompleksiteten i OpenStack-Helm varierer utifra hva man vil gjøre. Å sette opp et standard oppsett eller en type proof-of-conecpt kan gjøres veldig enkelt ved å følge installerings- og oppsettguider for OpenStack-Helm. Da det kommer til seriøst vedlikehold og alt som er mer detaljert enn standard oppsett, kreves det en god del dypere kompetanse. Dette kommer av at OpenStack-Helm er bygget på Kubernetes og det er da nødvendig at drifterne er godt kjent med teknologien. Kubernetes er et kompleks system som krever bred forståelse av teknologiens helhet for å virkelig kunne ta kontroll over det.

Selve konseptet med at driftere er nødt til å forstå Kubernetes som er et avansert system, på et mellomnivå for å være komfortable med å drifte det, er punktet som drar OpenStack-Helm ned i forhold til Kolla-Ansible og selvbygde konteinere. Dette hadde ikke vært et stort problem om ressursene og drifternes kapasitet var større. IIK består som allerede nevnt kun av to driftere som ikke har avansert kompetanse med konteinerteknologi fra før av. Det opplevdes også utfordrende for gruppen å lære seg Kubernetes og sammenhengen mellom dens komponenter og OpenStack-Helm. Det å kun forholde seg til konfigurasjonsfiler som values.yaml er ikke nok om problemer oppstår eller avanserte endringer må foretas.

Som nevnt ovenfor krever det lite kunnskap for å sette opp en helt ny infrastruktur ved bruk av OpenStacks guider. Fra dette punktet kan det bygges kunnskap ved hjelp av praktisk arbeid og prøving og feiling. Det var nettopp dette som var vår tilnærming til læring av Kubernetes og OpenStack-Helm. I vårt tilfelle møtte vi tidlig på problemer som vi ikke var i stand til å løse innen gitt tidsramme. Med litt mer tid ville vi nok kommet en god del lenger med denne tilnærmingen.

7.3.3 Trend / popularitet i bransjen

Av de tre vurderte kandidatene har OpenStack-Helm raskt blitt et populært alternativ for bedrifter som ønsker å bygge en konteinerbasert OpenStack-installasjon. AT&T og SK-Telecom er eksempler på bedrifter som tar i bruk OpenStack-Helm [39]. At så store bedrifter benytter seg av OpenStack-Helm legger grunnlag for at det absolutt er en aktuell kandidat å vurdere. Dokumentasjonen til løsningen er også greit skrevet med en god del ressurser. Det mangler litt på forklaringer på enkelte områder samtidig som dokumentasjonen kan bli litt uoversiktlig.

Feilsøking og treff rundt hyppig forekommende problemer har vært et problem for oss. Ved spesifikke feil finnes det av og til ikke mer en eller to resultater som er direkte tilknyttet og de erfaringene som eksisterte var ofte lite hjelpsomme. Det er mangelfult med søkeresultater på dette området og det gir brukere vanskeligheter med feilsøking der dokumentasjon om feilsøking ikke strekker til. Ved første øyekast av søking etter spesifikke problemer og informasjon om OpenStack-Helm oppleves nettsamfunnet veldig lite. Det er lite informasjon å hente som ikke kommer direkte fra OpenStack selv. Det kommer av at OpenStack-Helm har et ganske lukket nettsamfunn.

Derimot finnes det en Slack-kanal som er veldig aktiv [46]. Her diskuterer utviklere/-brukere, både profesjonelle og amatører konstant. Kanalen gir god mulighet for direkte kontakt med ansatte for OpenStack eller andre med god erfaring. I tillegg til dette har også OpenStack-Helm vært et tema flere ganger ved OpenStack sine samlinger, OpenStack Summit. Disse presentasjonene består av oppdateringer og nyheter om prosjektet og det gir brukere verdi ved å holde dem engasjerte og inkludert i samfunnet.

7.3.4 Sikkerhet

På samme linje som Kolla-Ansible er sikkerheten rundt selve systemet basert på sikkerheten til maskinene der løsningen styres fra. Det vil normalt sett være en en eller fler mester-maskiner som vil ha kontroll på resten av clusteret. Om en eventuell angriper skulle få kontroll over denne maskinen vil angriperen da ha full kontroll over clusteret.

En annen sikkerhetsfaktor er konteinersikkerhet, mer spesifikt Docker-konteinere. Dette er beskrevet tidligere i kapittelet, og det er ingen spesifikke forhold for OpenStack-Helm i forhold til andre konteinertjenester da det kommer til dette.

Enda en potensiell trussel er allerede kjente svakheter med OpenStack-Helm. Det er ikke noen kjente sårbarheter i nåværende versjon av programvaren. Det har blitt funnet sårbarheter i tidligere versjoner [72]. Et system er aldri hundre prosent sikkert, men utviklerne av OpenStack-Helm gjør en god jobb med å spesifisere egne best-practices og hvordan man sikrer løsningen [73]. En annen faktor med å konteinersikkerhet er å kunne stole på hvor konteiner-imagene blir hentet fra. I dette tilfellet blir alle imagene hentet fra OpenStack sine offisielle repoer for Kolla-prosjektet, som regnes som en trygg kilde.

7.3.5 Oppsummering

OpenStack-Helm er den mest komplekse løsningen, men samtidig den løsningen med potensial til å være mest effektiv, med tanken på at det er bygget oppå Kubernetes. Med mulighetene OpenStack-Helm har for orkestrering har det gode muligheter for skalering, som gjør at det passer bra for både mindre og veldig store deployments. Her skiller OpenStack-Helm seg positivt ut i forhold til Kolla-Ansible og selvbygde konteinere. Det finnes masse informasjon på nettet om Kubernetes men det er et komplekst system å sette seg inn i. OpenStack-Helm lider av litt mangefull dokumentasjon og manglende, samt lite tilgjengelige ressurser ved feilsøking. Det finnes private samfunn hvor problemer og løsninger blir diskutert på daglig basis, men OpenStack-Helm sin brukergruppe gir et inntrykk av å være noe lukket.

OpenStack-Helm er mer enn kapabel til å utføre IIKs oppgaver, men på bekostning av høy kompleksitet.

7.4 Selvbygde konteinere

Selvbygde konteinere er en av alternativere for konteinerisering som IIK på forhånd ønsket en vurdering av. Dette alternativet har størst fleksibilitet og mulighet for egenstyring.

7.4.1 Effektivitet

Selvbygde konteinere krever at du setter deg godt inn i hvordan OpenStack sine tjenester settes opp og konfigureres fra bunnen av. Denne kunnskapen videreføres til å sette opp tjenesten i konteiner. Nettopp på grunn av dette blir prosessen tidkrevende og rotete når flere tjenester skal slås sammen. En fordel her er at man slipper å sette seg inn et prosjekt utviklet av noen andre og bruke tid på å forstå hvordan alt henger sammen. For gruppens tilfelle har det blitt satt opp en Keystone-konteiner og en Memcached-konteiner.

Når det kommer til oppgraderingen av denne løsningen kommer det først og fremst an på hvordan det implementeres. Dersom det brukes miljøvariabler for nøkkelvariabler, gir det en mer dynamisk løsning for senere endringer og oppgraderinger. Slik gruppen implementerte Keystone, er det enkelt å oppgradere for eksempel versjonen på tjenesten ved å endre en miljøvariabel, med forbehold om at fremtidige utslipp ikke krever nye støtteprogrammer o.l. Dette forbedrer hvordan dagens oppgraderinger foregår. Tanken til gruppen har vært å implementere andre tjenester med samme mulighet slik at det forenkler oppgraderinger.

Når det gjelder endringer og programvareoppdatering er det effektivt når tjenesten kjører i konteiner. Det er mulig å bruke en immutable containers-tilnærming. Dette går ut på å ta ned konteineren hver gang en endring er nødvending og lage den på nytt. Da konteinere er lettvektige prosesser som er enkle å ta ned og starte på nytt vil ikke dette påvirke effektiviteten nevneverdig. Den viktigste fordelen med denne tilnærmingen er repeterbarhet i stedet for å gjøre en ad hoc endring i konteinerne med docker update og lignende.

Selvbygde konteinere kan i høy grad integreres inn i konfigurasjonsstyringssystemer. Det

er riktignok ikke implementert av gruppen grunnet tidsbegrensninger. Gruppen fikk kun tid til å implementere Kolla-Ansible i CMS.

Gruppen har brukt Docker Swarm til å orkestrere den selvbygde løsningen. Dette ble gjort for å demonstrere redundans og effektivisering når det kommer til opp- og nedskalering av tjenester i selvbygde konteinere. Dersom det er behov for å ha flere Keystone enn én, skjer dette på noen få sekunder med Docker Swarm. Swarm har ikke autoskalering, derfor må dette defineres manuelt.

Hvis konteinerimages ikke bygges riktig kan det føre til økt ressursbruk og dårlig ytelse. Dette går utover effektiviteten til skyplattformen og ressursene IIK har til rådighet. I konteinerne vi bygger ser vi lite ressursbruk. De bruker 0,2-0,3% av tilgjengelige CPU-kapasitet for tre Keystone-konteinere sammen med Memcached, maskinen de kjører i har to VPCU-er. Størrelsen på Keystone imaget er 1.09GB. Dette kan gjøres mer ressurseffektivt. Gruppens prioritet har vært å lage et proof-of-concept uten å ta noe særlig hensyn til imagestørrelse.

7.4.2 Kompleksitet

Implementasjon av selvbygde konteinere var enklere enn gruppen forventet, men vil variere utifra hvordan tjenesten implementeres og om det innføres konteinerorkestrering slik som Swarm eller Kubernetes. Det kreves mer kunnskap for å sette seg inn i disse. Skalering og automatisering øker kompleksiteten. Det er enklere å feilsøke selvbygde konteinere enn de andre løsningene da man naturligvis vil ha mer direkte kontroll over tjenestene.

Denne løsningen krever god kompetanse innen mikrotjenester og hvordan tjenestene i OpenStack installeres og ikke minst kommuniserer eller samhandler med hverandre. Da oppgaven vår er begrenset til kun Keystone er det vanskelig å vurdere kompleksiteten for et oppsett der flere tjenester kommuniserer med hverandre. Kompleksiteten øker dersom konteiner orkestrering innføres samt nye tjenester innføres.

7.4.3 Trend / popularitet i bransjen

Av det gruppen finner av selskaper som bruker OpenStack, benytter de fleste en etablert løsning, slik som Kolla-Ansible eller OpenStack-Helm. Gruppen har ikke funnet spesifikke bedrifter som bruker selvbygde konteinerne uten å kombinere det med andre løsninger. Det er endel OpenStack-brukere som benytter selvbygde løsningener i form av konteinere kombinert med PaaS-platformer, se figur 8. Utifra figuren ser vi at det er god del som benytter enten Built our owneller Docker Swarm når de ruller ut skyplattformen sin. Det er 19 prosent som benytter selvbygd og 10 prosent Swarm fra undersøkelsen denne figuren er basert på. Dette er en klar indikasjon for at en god del av brukerbasen til OpenStack benytter seg av selvbygde løsninger.

7.4.4 Sikkerhet

Løsningen vil ta god nytte av den sikkerheten som Docker har innebygget, som blant annet tilbyr isolasjon fra host VM-en og stenger alle porter som standard. Det blir viktig å følge det som er best practice[74] når tjenestene implementeres og kjøres i konteinere. Dette innebærer at løsningen ikke åpner unødvendige porter og setter korrekt eierskap

for konfigurasjonsfiler og prosesser. Det blir også essensielt å sikre selve VMene som kjører tjenestene, spesielt Swarm Leader noden, da denne noden har alle rettigheter til å administrere alle tjenestene i klusteret. En eventuell løsning vil heldigvis ta nytten av den sikkerheten IIK har implementert, som blant annet stenger adgang til alle som ikke er koblet til skolens nettverk, enten via VPN eller direkte.

Docker Swarm har innebygget lastbalansering, noe som medfører at en trenger kun å kommunisere med en av nodene til klusteret, selv om tjenesten ikke kjører i den VMen så ordner lastbalanseringen det slik at det rutes til riktig node innad i klusteret. Dette har Docker Swarm valgt å håndtere ved å la alle portene som tas i bruk av tjenestene i klusteret være åpent på alle maskinene, dette ser vi på som mindre optimal løsning.

7.4.5 Oppsummering

Selvbygde konteinere har ikke i seg selv noe system for deployment. Det kan kobles sammen med nesten hva som helst annet og har gode muligheter for skalering og utvidelse av scope hvis det kombineres med riktig verktøy. Det er nødvendig med god kompetanse innen Docker og mikrotjenester for å bruke selvbygde konteinere som en reell løsning.

Ved å gjøre ting fra bunn av selv kan man med nok ressurser og kompetanse få det nøyaktig som man vil. Hvis det implementeres noe form for orkestreringsverktøyer slik som Docker Swarm eller Kubernetes øker kompleksiteten drastisk og prosjektet kan fort få et betydelig ressurskrav da det kommer til arbeidstimer. Det er ingen andre som ordner automatiske løsninger for oppgraderinger etc, dette må gjøres selv. Denne løsningen krever mest tid å sette opp, vedlikeholde og oppgradere av alle løsningene vi vurderer.

Denne løsningen passer for et større team som har endel ressurser i form av arbeidstimer og kompetanse innen mikrotjenester. Dersom det er ønskelig å skreddersy løsningen selv og det ikke gjør noe å investere en god del ekstra timer på å vedlikholde og oppgradere den, passer denne løsningen. Denne løsningen gir full kontroll over egen infrastruktur, samt fjerner risikoen for at man bruker et prosjekt hvor støtten til prosjektet avsluttes.

7.5 Sammenligning

Etter å ha gjort en individuell implementasjon og dypere vurdering av hver av de tre kandidatene over, gjør gruppen her en sammenligning av de forskjellige kandidatene.

7.5.1 Effektivitet

Alle de tre kandidatene vil bidra til å forbedre driftseffektiviteten til IIK, dette innbærer å sette opp tjenestene i OpenStack, endre eller konfigurere tjenestene, oppgradere versjonene til OpenStack og introdusere eller slette nye tjenester. Dette fordi det er enklere å gjøre endringer og oppdateringer med konteinere i motsetning tjenester som kjører i virtuelle maskiner. Gruppen mener derimot at Kolla-Ansible og OpenStack-Helm vil forenkle dette mer enn selvbygde konteinere.

OpenStack-Helm vurderes som den med best effektivitet av alle løsningene. Grunnen til dette er at OpenStack-Helm er bygget på Kubernetes og har gode ressurser for implementasjon av forskjellige tjenester tilgjengelig. Noen av mulighetene eller fordelene til

Kubernetes som Kolla-Ansible ikke har er automatisk skalering av tjenesten når det er behov for det, self healing som betyr at pods eller tjenester som går ned blir automatisk gjenopptatt av Kubernetes i henhold til den forhåndsdefinerte ønskede tilstanden desired state [75]. Kolla-Ansible følger like bak men er ikke like effektiv som OpenStack-Helm på å sette opp flere konteinere av samme tjeneste. Det som gir Kolla-Ansible høy verdi på effektivitet er at det allerede er bygget rundt Ansible noe som gjør det enkelt og effektivt å styre konteinere. Til slutt følger egenbygde konteinere. For å få mer automatisk konfigurasjonsstyring av konteinere bør det bli benyttet noe form for konfigurasjonsstyringssystemer som f.eks Puppet eller Ansible, men også dette vil involvere betydelig mer arbeid enn hos konkurrentene.

Kolla-Ansible og OpenStack-Helm anses som mer effektive løsninger enn selvbygde løsninger. Selvbygde konteinere er en løsning som må planlegges og utvikles selv helt fra når det settes opp og hver gang nye OpenStack-tjenester skal introduseres. Det vil kreve mer å drifte selvbygde konteinere sammenlignet med de to andre løsningene, da det ikke er noen tilgjengelige verktøy for forenkling av driftsprosesser. Dette må for selvbygde konteinere eventuelt utvikles selv, mens konkurrentene har verktøy tilgjengelige.

7.5.2 Kompleksitet

Vurderingen faller på at Kolla-Ansible er den løsningen med lavest kompleksitet, etterfulgt av selvbygde konteinere. Det som gjør selvbygde konteinere noe mer komplekst enn Kolla-Ansible er at det kreves dypere kunnskap om mikrotjenester, spesielt når det settes opp noe form for konteinerorkestrering slik som Docker Swarm eller Kubernetes. OpenStack-Helm har høyere kompleksitet, grunnet behovet for å sette seg inn i Helm-charts og Kubernetes, og at det er begrenset informasjon å finne ved nettsøk og leting i prosjektets dokumentasjon. Det har ført til at gruppen ikke klarte å gjøre en komplett Multinode deployment av OpenStack-Helm slik de hadde som mål å gjøre.

7.5.3 Popularitet og trender

Kolla-Ansible er den løsningen gruppen vurderer som den best etablerte over tid, med gode ressurser og representasjon i større organisasjoner og bedrifter. OpenStack-Helm er en nyere og voksende løsning, og har også søsterprosjekter som videreutvikler teknologien. OpenStack-Helm og dets søsterprosjekter har blitt tatt i bruk av flere store aktører i den siste tiden og er basert på Kubernetes som har sterkt voksende popularitet. Gruppen anser OpenStack-Helm som den løsningen med best utvikling da det gjelder popularitet, selv om nettsamfunnet er noe lukket. Selvbygde konteinere brukes sjeldent frittstående i bedrifter, og anses som den minst populære løsningen av de tre.

7.5.4 Sikkerhet

De tre løsningene har forholdsvis likt sikkerhetsnivå. Hvis det ikke gjøres noe for å sikre selvbygde konteinere vil Kolla-Ansible og OpenStack-Helm være sikrere, men om det legges ressurser i det kan selvbygde konteinere gjøres sikrere enn de andre løsningene grunnet den økte tilpasningsmuligheten. Blir løsningene implementert på normal måte vil ikke sikkerheten verken øke eller minske særlig om IIK følger teknologiens standarder.

7.5.5 Kvantitativ vurdering

Tabell 5 blir brukt for å sammenligne de tre valgte løsningene. Gruppen vil bruke en skala fra en til seks. Mer utdypende informasjon om denne skalaen kan leses under tabellen. Kriteriene i tabellen er definert i kravspesifikasjonen i kapittel to og i forklaring av kriterier i kapittel sju.

Utvalg	Effektivitet	Kompleksitet	Sikkerhet	Popularitet & trender
Kolla-Ansible	4	5	3	4
Openstack-Helm	5	2	3	4
Selvbygde konteinere	3	4	3	2

Tabell 5: Kvantitativ vurdering av implementerte løsninger

Tabellen viser verdier vi har satt forhold til vår vurdering.

- **Effektivitet**

  Høyeste verdi (5-6): Løsningen vil øke effektivitet i forhold til allerede implementert måte på en svært god måte.

  Mellomverdi (3-4): Løsningen vil bidra med noe eller en del effektivering av allerede implementert løsning.

  Laveste verdi (1-2): Løsningen vil ikke effektivisere eller minimalt effektivisere allerede implementert løsning.

- **Kompleksitet**

  Høyeste verdi (5-6): Løsningen har liten eller svært liten grad av kompleksitet og krever lite eller nærmeste ingen kompetanse for å implementere og drifte.

  Mellomverdi (3-4): Løsningen har middels til under middels kompleksitet.

  Laveste verdi (1-2): Løsningen har svært stor grad av kompleksitet og vil være veldig vanskelig for IIK å implementere og drifte.

- **Sikkerhet**

  Høyeste verdi (5-6): Løsningen har veldig stor fokus på sikkerhet og vil styrke allerede implementert løsnings sikkerhet betraktelig.

  Mellomverdi (3-4): Løsningen har moderat til god fokus på sikkerhet og vil ikke svekke eller styrke sikkerheten.

  Laveste verdi (1-2): Løsningen har ingen tilnærming til sikkerhet og vil svekke allerede implementert løsnings sikkerhet.

- **Popularitet**

  Høyeste verdi (5-6): Løsningen er svært populær og foretrukket blant brukere og bedrifter, samt har et godt nettsamfunn for support, vedlikehold og støtte.

  Mellomverdi (3-4): Løsningen er middels til over middels populær, foretrukket blant brukere og bedrifter, samt har middels til over middels godt nettsamfunn for support, vedlikehold og støtte.

  Laveste verdi (1-2): Løsningen er lite populær og mindre foretrukket blant brukere og bedrifter, samt har dårlig nettsamfunn for vedlikehold og støtte.

8 Diskusjon og konklusjon

8.1 Diskusjon

I denne oppgaven har det blitt gjort en utredning av hvilken konteineriseringsteknologi som egner seg best bruk i skolens private skyløsning SkyHiGh. Den planlagte strukturen for arbeidet med oppgaven endret seg da det viste seg at det var flere forskjellige alternativer som alle var potensielt gode kandidater, og ble mindre fokusert på Kolla enn gruppen først hadde antatt. Planen var først å sammenligne Kolla med selvbygde konteinere, men også holde utkikk etter andre aktuelle kandidater. Det viste seg å være opp til flere gode kandidater, så arbeidsplanen og rapportstrukturen ble endret for å tilpasse seg dette. Det har først blitt gjort en initiell vurdering av disse kandidatene som beskrevet i kapittel 5. Det som anses av gruppen som de beste alternativene ble implementert i et testmiljø. Detaljerte vurderinger av disse ble gjort, slik at gruppen sin anbefaling til IIK vil ha så god tyngde og relevans som mulig. Testmiljøet var først planlagt å være SkyLow, men da gruppen endte opp med å implementere flere løsninger ble dette upraktisk og ikke regnet som høyeste prioritet. Implementasjon og testing for vurdering ble derfor gjennomført i virtuelle maskiner satt opp i SkyHiGh. Det ble gjort et forsøk på å simulere et miljø som realistisk kan være et alternativ for IIK, blant annet ved å holde databaser og støttetjenester eksternt der det lot seg gjøre. De tre kandidatene gruppen valgte å implementere var Kolla-Ansible, OpenStack Helm og selvbygde konteinere. Da gruppen ser tilbake på utvalget etter fullført prosess anser vi dette som et godt og korrekt utvalg ut fra de tilgjengelige kandidatene. Etter implementasjonen sitter gruppen igjen med et inntrykk av at alle er gode kandidater og kunne gitt verdi for IIK.

Kolla-Ansible er tett integrert med OpenStack, har gode muligheter for å kunne forenkle driftsrutiner for IIK, og krever lite kunnskap om tjenester utenom OpenStack for å kunne implementeres. Løsningen er godt etablert på markedet og det er den løsningen med best tilgjengelige ressurser og hjelp online. Kolla-Ansible brukes i mange store teknologibedrifter, men det har vært en oppfatning i brukergruppen at det har vært noe stagnert vekst og at andre alternativer får økende oppmerksomhet.

OpenStack Helm er en løsning som har høyt potensiale, men gruppen anser den som noe mer komplisert å sette seg inn i for personer som ikke har erfaring med Helm og Kubernetes. OpenStack-Helm har vokst mye i senere år som vist ved bruken i moderne prosjekter og ved aktiviteten i både relaterte utviklingsprosjekter og i brukersamfunn. Det er en løsning som gruppen mener vil bli enklere å utnytte seg av i fremtiden ved søster-prosjekter som for eksempel OpenStack-Airship. I likhet med Kolla-Ansible brukes OpenStack-Helm av mange store teknologiaktører [39]. Det er gode muligheter for skalering og automatisering ved bruk av Kubernetes, men fører til videre økning av kompleksitet. Det er ikke veldig mye ressurser om OpenStack-Helm online, men det er en Slack-kanal som er meget aktiv med erfarne brukere.

Selvbygde konteinere gir bedre tilpasningsmuligheter enn noen av de andre løsningene. Det har potensiale til å gi den samme funksjonaliteten som de andre løsningene, men det krever en stor mengde arbeid med kode. Selvbygde konteinere med minimalt av automatisering og tilpasning krever kun noe kunnskap om Docker og generell skripting. Denne løsningen vil krever en god del ekstra arbeid å sette opp og vedlikeholde hele skyinfrastrukturen over tid enn de to andre løsningene. Kompleksiteten øker i veldig stor grad ved utvidet omfang i løsningen, både i form av interaksjon mellom forskjellige tjenester og eventuelt ved innføring av orkestrering. Selvbygde konteinere brukes sjeldent alene i bedrifter, men som en del av en større løsning. Det er mye ressurser tilgjengelig om oppsett å bygge konteinere og gode Dockerfiles, men det er generalisert. Det er lite ressurser om selvbygde konteinere av OpenStack sitt kontrollplan.

Gruppen har tatt kontakt med to studenter som skriver bacheloroppgaven CI/CD for SkyHiGh. Denne gruppen leverer et resultat som kan endre deler av grunnlaget for vurdering gjort i denne oppgaven. Gruppen har ikke studert deres foreslåtte løsning i detalj, men etter korte samtaler med CI/CD-gruppen mener vi at de to prosjektene i stor grad kan implementeres ved siden av hverandre. En av representantene for CI/CD-gruppen uttrykte også at Kubernetes var en teknologi deres system hadde funksjonalitet for som de mente var ønskelig. Av de vurderte løsningene i oppgaven vår gjelder dette OpenStack-Helm.

8.1.1 Resultater

Gruppen har oppnådd de fleste av målene for denne bacheloroppgaven. Gruppen har gjort en utredning av tre forskjellige konteineriseringsløsninger for SkyHiGh, Kolla-Ansible, OpenStack-Helm og selvbygd konteinere. Det har også blitt gjort implementasjon av Keystone for alle tre for vurdering og testing. Gruppen har kommet til en konklusjon om en akademisk anbefaling av løsning for konteinerisering av kontrollplanet i SkyHiGh.

Gruppen har ikke hatt anledning til å implementere løsningene i SkyLow, kun på virtuelle maskiner i SkyHiGh. Gruppen hadde noen utfordringer med implementasjonen av OpenStack-Helm. Multinode deployment av løsningen støtte på problemer, og implementasjonen av OpenStack-Helm er derfor satt opp på en enkel virtuell maskin, en All-in-One deployment.

8.2 Videre arbeid

Ved videre arbeid foreslår vi at administratorene av SkyHiGh ved IIK prøver å erstatte den enkleste tjenesten Keystone med konteinere i SkyLow. Klarer de å integrere Keystone i SkyLow og erstatte deres nå kjørende VMer, vil det være enklere å kontinuerlig erstatte tjeneste for tjeneste ved bruk av hvilken som helst av de tre vurderte teknologiene. Velger IIK å ta i bruk OpenStack-Helm anbefaler vi å se på OpenStack Airship. Teknologien er såpass ny at vi ikke fikk sett ordentlig på den men den ser ut til å ha store ambisjoner om å forenkle OpenStack-Helm med deklarativ bruk av yaml. Teknologien er også allerede brukt av andre store selskaper som nevnt tidligere i rapporten.

Det anbefales også å lese seg opp å lære mer om konteinerteknologi generelt. Det vil effektivisere IIKs allerede implementerte løsning uavhengig av hvilken teknologi som velges. Siden konteinere er uavhengig fra operativsystem vil oppgraderinger av Open-

Stack tjenester uansett bli raskere ved bruk av konteinere. I tillegg vil det være lurt av IIK å lese seg opp på implementasjonsguidene fra OpenStack for å skaffe seg et overblikk om hvordan de kan implementere løsningene selv.

8.3 Evaluering av gruppearbeidet

Gruppemedlemmene har allerede jobbet sammen ved tidligere anledninger. Dette ga oss en god start på prosjektet ved at vi visste om hverandres kvaliteter og arbeidsmoral. Det ble tidlig diskutert mål for prosjektet og gruppemedlemmene kom fort til enighet om hva som var forventet. Ved uenigheter ble det alltid diskutert på positiv måte om problemet og hva det kom av, med fokus på en konstruktiv løsning. Det ble så tatt opp med veileder om vi ikke kom frem til noe selv. Alle gruppemedlemmene møtte opp og arbeidet etter forventning.

Det var fire arbeidsøkter i uken fra 09.00 til 17.00. Dette virket bra for alle og hjalp med å opprettholde god kommunikasjon og arbeidsmoral gjennom hele prosjektperioden. Hver morgen på gruppens arbeidsdager ble det holdt et daglig møte i 10 minutter hvor gruppen diskuterte dagens gjøremål og utfordringer fra dagen før. Dette var også en viktig brikke for å stadig holde jevn kurs mot mål ved at vi alltid visste hva som trengtes å gjøre, og hvem som skulle gjøre hva. Disse morgenmøtene førte til at det ble minimalt med tid bortkastet på å ikke vite hva man skulle gjøre. I tillegg ble det også holdt møter med veileder hver mandag hvor vi møtte forberedt med spørsmål og noterte ned svar for å få så godt utbytte som mulig av veiledningen.

Alt i alt er vi fornøyd med egen innsats. Vi nådde forventingene og målene vi satt. Det oppstod minimalt med uenigheter og sure miner innad i gruppen. Noen problemer oppstod, og det var litt stagnering både på det praktiske arbeidet, og på rapporten, men vi motiverte hverandre videre til å arbeide og kom oss fort videre.

8.4 Konklusjon

Konklusjonen er basert på problemstillingen nevnt tidligere i rapporten *Hvilken konteinerteknologi egner seg best for å konteinerisere tjenestene i SkyHiGh.* Ut i fra vår forskning og resultater, mener vi Kolla-Ansible er den teknologien som er best egnet. De andre kandidatene er likevel svært gode alternativer, spesielt OpenStack-Helm. OpenStack-Helm lider av litt for stor kompleksitet og lite nettsamfunn, spesielt med tanke på det faktumet at SkyHiGh driftes av kun to personer. Selvbygde konteinere er også et bra alternativ, men kommer ikke med konfigurasjonsstyring, gode ferdige imager, support og vedlikehold slik som Kolla-Ansible. I tillegg har Kolla-Ansible, hyppige oppdateringer og aktivt nettsamfunn. Selv om gruppen sin konklusjon er en anbefaling av Kolla-Ansible, så er de vurderte kandidatene alle gode nok løsninger til at oppdragsgiver burde ta en grundig vurdering for å selv bedømme hvilken de mener egner seg optimalt for SkyHiGh.

Bibliografi

[1] Authors, T. K. 2018. Performing a rolling update. https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/. [Hentet 18. mai 2019].

[2] Columbus, L. 2018. Roundup of cloud computing forecasts and market estimates, 2018. https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/#6a840cfd507b. [Hentet 10. mai 2019].

[3] Carter, E. 2018. 2018 docker usage report. https://sysdig.com/blog/2018-docker-usage-report/. [Hentet 18. mai 2019].

[4] Sell, L. 2012. Openstack launches as independent foundation, begins work protecting, empowering and promoting openstack. https://www.businesswire.com/news/home/20120919005997/en/OpenStack-Launches-Independent-Foundation-Begins-Work-Protecting. [Hentet 18. mai 2019].

[5] Foundation, O. 09. mai 2019. Openstack releases. https://releases.openstack.org/. [Oppdatert 17. mai 2019; Hentet 18. mai 2019].

[6] Vaughan-Nichols, S. J. 2011. Canonical switches to openstack for ubuntu linux cloud. https://www.zdnet.com/article/canonical-switches-to-openstack-for-ubuntu-linux-cloud/. [Hentet 18. mai 2019].

[7] Goirand, T. 2013. Openstack havana 2013.2 debian packages available. http://thomas.goirand.fr/blog/?p=141. [Hentet 18. mai 2019].

[8] SUSE. 29. august 2012. Suse releases first openstack-based enterprise private cloud solution. https://www.suse.com/c/news/suse-releases-first-openstack-based-enterprise-private-cloud-solution/. [Hentet 18. mai 2019].

[9] Foundation, O. Companies supporting the openstack foundation. https://www.openstack.org/foundation/companies/. [Hentet 18. mai 2019].

[10] Babcock, C. 2012. Nasa drops openstack for amazon cloud. https://www.informationweek.com/cloud/infrastructure-as-a-service/nasa-drops-openstack-for-amazon-cloud/d/d-id/1104911. [Hentet 18. mai 2019].

[11] Foundation, O. Openstack interoperability. https://www.openstack.org/brand/interop/. [Hentet 15. mai 2019].

[12] contributors, W. 2019. Openstack — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/OpenStack. [Hentet 18. mai 2019].

[13] Foundation, O. Openstack keystone wiki. https://wiki.openstack.org/wiki/keystone. [Hentet 18. mai 2019].

[14] Foundation, O. 22. april 2019. Keystone documentation. https://docs.openstack.org/keystone/latest/. [Hentet 18. mai 2019].

[15] contributors, W. 2019. Openstack components wikipedia. https://en.wikipedia.org/wiki/OpenStack#Components. [Hentet 18. mai 2019].

[16] Foundation, O. Openstack glance wiki. https://wiki.openstack.org/wiki/glance. [Hentet 18. mai 2019].

[17] Foundation, O. 08. oktober 2018. Glance documentation. https://docs.openstack.org/glance/latest/. [Hentet 18. mai 2019].

[18] Foundation, O. Openstack neutron wiki. https://wiki.openstack.org/wiki/neutron. [Hentet 18. mai 2019].

[19] Foundation, O. 08. mars 2018. Neutron documentation. https://docs.openstack.org/neutron/latest/. [Hentet 18. mai 2019].

[20] Foundation, O. Openstack nova wiki. https://wiki.openstack.org/wiki/nova. [Hentet 18. mai 2019].

[21] Foundation, O. 26. mars 2019. Nova documentation. https://docs.openstack.org/nova/latest/. [Hentet 18. mai 2019].

[22] Foundation, O. Openstack heat wiki. https://wiki.openstack.org/wiki/heat. [Hentet 18. mai 2019].

[23] Foundation, O. 12. juli 2019. Heat documentation. https://docs.openstack.org/heat/latest/. [Hentet 18. mai 2019].

[24] Foundation, O. Openstack horizon wiki. https://wiki.openstack.org/wiki/horizon. [Hentet 18. mai 2019].

[25] Foundation, O. 24. juli 2017. Horizon documentation. https://docs.openstack.org/horizon/latest/. [Hentet 18. mai 2019].

[26] Pivotal Software, I. Rabbitmq documentation. https://www.rabbitmq.com/documentation.html. [Hentet 20. feb 2019].

[27] Foundation, O. 01. mai. 2019. Openstackclient documentation. https://docs.openstack.org/python-openstackclient/latest/. [Hentet 18. mai 2019].

[28] Rahul. 2016. Openstack and containers (project kolla). https://rahulait.wordpress.com/2016/09/20/openstack-and-containers-project-kolla/. [Hentet 18. mai 2019].

[29] Foundation, C. N. C. Helm glossary. https://helm.sh/docs/glossary/. [Hentet 10. mai 2019].

[30] Bhagwat, M. 2017. Understanding helm charts. https://medium.com/@manoj.bhagwat60/understanding-helm-charts-26f2db983d96. [Hentet 01. mai 2019].

[31] Muhammed, S. K. 2018. Draft vs gitkube vs helm vs ksonnet vs metaparticle vs skaffold. https://blog.hasura.io/draft-vs-gitkube-vs-helm-vs-ksonnet-vs-metaparticle-vs-skaffold-f5aa9561f948/. [Hentet 18. mai 2019].

[32] Kłopotek, M. 2019. Helm. https://github.com/helm/helm/blob/master/README.md. [Hentet 18. mai 2019].

[33] Foundation, O. Installation. https://docs.openstack.org/openstack-helm/latest/install/index.html, Timestamp = Jan 01, 2019, year =.

[34] Foundation, O. 2019. Gate-based kubernetes. https://docs.openstack.org/openstack-helm/latest/install/kubernetes-gate.html. [Hentet 13. mai 2019].

[35] Foundation, O. 15.09.2018. Multinode. https://docs.openstack.org/openstack-helm/latest/install/multinode.html. [Hentet 18. mai 2019].

[36] Docker, I. 2019. What is a container. https://www.docker.com/resources/what-container. [Hentet 18. mai 2019].

[37] Venugopal, S. 2016. Cloud orchestration technologies: Explore your options. https://developer.ibm.com/articles/cl-cloud-orchestration-technologies-trs/. [Hentet 18. mai 2019].

[38] Google, I. Containers at google. https://cloud.google.com/containers/. [Hentet 13. mai 2019].

[39] Whitaker, B. E. Leveraging containers and openstack. https://www.openstack.org/containers/leveraging-containers-and-openstack/. [Hentet 18. mai 2019].

[40] Jameson, J. 2015. Containerize openstack with docker. https://www.redhat.com/en/blog/containerize-openstack-docker. [Hentet 18. mai 2019].

[41] poster, A. 02.01.2019. Reddit post - openstack usage in gaming company. https://www.reddit.com/r/openstack/comments/abr9fr/is_openstack_still_relevant/ed2of3v/?utm_source=share&utm_medium=web2x. [Hentet 09. mai 2019].

[42] Ltd, C. Kolla-ansible launchpad. https://bugs.launchpad.net/kolla-ansible. [Hentet 08. mai 2019].

[43] Foundation, O. 25. september 2017. Kolla-ansible documentation. https://docs.openstack.org/kolla-ansible/. [Hentet 08. mai 2019].

[44] Foundation, O. Kolla wiki. https://wiki.openstack.org/wiki/Kolla. [Hentet 01. mai 2019].

[45] Edgar Silva, V. X. Ansible module - puppet forge. https://forge.puppet.com/otherskins/ansible. [Hentet 02. mai 2019].

[46] Foundation, O. 27. april 2017. Openstack-helm. https://docs.openstack.org/openstack-helm/latest/readme.html. [Hentet 18. mai 2019].

[47] Foundation, O. 27. april 2017. Openstack- helm. https://docs.openstack.org/openstack-helm/latest/readme.html. [Hentet 18. mai 2019].

[48] Foundation, O. What containers platform-as-a-service tools are openstack users deploying? https://www.openstack.org/containers. [Hentet 02. mai 2019].

[49] wangxf. kolla-kubernetes. https://github.com/wangxf1987/kolla-kubernetes. [Hentet 10. mai 2019].

[50] Foundation, O. 24. april 2018. Kolla kubernetes development environment. https://docs.openstack.org/kolla-kubernetes/latest/development-environment.html. [Hentet 18. mai 2019].

[51] Wheatley, M. 2017. Mirantis combines openstack and kubernetes in a single package with continuous update model. https://siliconangle.com/2017/04/19/mirantis-combines-openstack-kubernetes-single-package-continuous-update-model/. [Hentet 18. mai 2019].

[52] Inc, M. Kubernetes software from mirantis. https://www.mirantis.com/software/kubernetes/. [Hentet 18. mai 2019].

[53] StackHPC. 2017. Kayobe's documentation. https://kayobe.readthedocs.io/en/latest/. [Hentet 30. april 2019].

[54] Telfer, S. 2019. Meet the latest release of kayobe: Even easier deployment of containerized openstack to bare metal. https://superuser.openstack.org/articles/kayobe-5-0-release/. [Hentet 08. mai 2019].

[55] Foundation, O. Elevate your infrastructure. https://www.airshipit.org/. [Hentet 06. mai 2019].

[56] Foundation, O. Airship v1.0 release. https://wiki.openstack.org/wiki/Airship#v1.0_Release. [Hentet 02. mai 2019].

[57] Wheelus, A. 2019. Elevating devops and enabling 5g with airship. https://www.airshipit.org/blog/elevating-devops-and-enabling-5g-with-airship.html. [Hentet 08. mai 2019].

[58] OpenStack Foundation, Jaesuk Ahn, R. C. Y. 2018. You can start small and grow (sk telecoms use case on armada). https://www.youtube.com/watch?v=eArc0lOhToo. [Hentet 08. mai 2019].

[59] Youtube, O. F. 2018. Introducing airship. https://www.youtube.com/watch?v=0eEisMm9ykg. [Hentet 07. mai 2019].

[60] Youtube, O. F. 11.05.2017. Openstack-helm- managing the life-cycle of openstack deployments on top of kubernetes. https://www.youtube.com/watch?v=9-EgvlJ0dvY. [Hentet 15. mai 2019].

[61] Foundation, O. 17. oktober 2018. Openstack-helm gates. https://docs.openstack.org/openstack-helm/latest/gates.html. [Hentet 02. mai 2019].

[62] Foundation, O. 07. juni 2018. Upgrades and reconfiguration. https://docs.openstack.org/openstack-helm/latest/devref/upgrades.html. [Hentet 10. mai 2019].

[63] Percoco, F. 2017. Evaluating tools available to deploy openstack on kubernetes. https://next.redhat.com/2017/08/21/evaluating-tools-available-to-deploy-openstack-on-kubernetes/. [Hentet 07. mai 2019].

[64] Coleman, M. 2016. Containers are not vms. https://blog.docker.com/2016/03/containers-are-not-vms/. [Hentet 16. mai 2019].

[65] Rosendahl, H. Containers vs virtual machines (vms) – a security perspective. https://neuvector.com/container-security/containers-vs-virtual-machines-vms/. [Hentet 10. mai 2019].

[66] Huang, F. 15 tips for a run-time container security strategy. https://neuvector.com/container-security/15-tips-run-time-container-security-strategy/. [Hentet 10. mai 2019].

[67] Hirschfeld, R. 2016. Thirteen ways containers are more secure than virtual machines. https://thenewstack.io/thirteen-ways-containers-secure-virtual-machines/. [Hentet 10. mai 2019].

[68] Clemenko, A. 2019. Docker reference architecture: Securing docker ee and security best practices. https://success.docker.com/article/security-best-practices. [Hentet 16. mai 2019].

[69] Foundation, O. 2013. Kolla's deployment philosophy. https://docs.openstack.org/kolla-ansible/4.0.2/deployment-philosophy.html. [Hentet 16. mai 2019].

[70] Inc., D. 2019. Docker hub search - kolla. https://hub.docker.com/search?q=Kolla&type=image. [Hentet 16. mai 2019].

[71] Foundation, O. 25. september 2017. Welcome to kolla-ansible's documentation! https://docs.openstack.org/kolla-ansible/latest/. [Hentet 16. mai 2019].

[72] Butcher, M. 2019. Helm vulnerability: Client unpacking chart that contains malicious content. https://helm.sh/blog/helm-security-notice-2019/. [Hentet 10. mai 2019].

[73] Foundation, C. N. C. securing-your-helm-installation. https://helm.sh/docs/using_helm/#securing-your-helm-installation. [Hentet 08. mai 2019].

[74] Murugiah Souppaya, John Morello, K. S. 2019. Nist special publication 800-190 - application container security guide. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf. [Hentet 15. mai 2019].

[75] Authors, T. K. 26. april 2019. What is kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Hentet 18. mai 2019].

[76] Foundation, O. 13. januar 2019. Bootstrapping servers. https://docs.openstack.org/kolla-ansible/latest/reference/deployment-and-bootstrapping/bootstrap-servers.html. [Hentet 15. mai 2019].

[77] Yogesh. 2016. Continously restarting rabbitmq container for centos. https://bugs.launchpad.net/kolla/+bug/1564773. [Hentet 16. mai 2019].

[78] YaZug. 2019. Rabbitmq image is failing to build and blocking the gate. https://bugs.launchpad.net/kolla/+bug/1814196.

[79] phanindra48. 2018. Liveness/readiness probes are failing with getsockopt: connection refused. https://github.com/kubernetes/kubernetes/issues/62594. [Hentet 15. mai 2019].

[80] viglia. 2018. es-master fails: Liveness probe failed: dial tcp 172.17.0.4:9300: getsockopt: connection refused. https://github.com/pires/kubernetes-elasticsearch-cluster/issues/175. [Hentet 15. mai 2019].

[81] Sodhi, K. 2018. Kubernetes readiness probe failed error. https://stackoverflow.com/questions/48540929/kubernetes-readiness-probe-failed-error. [Hentet 15. mai 2019].

[82] Kubernetes. 2019. Troubleshoot applications. https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application/. [Hentet 15. mai 2019].

A Feilsøking av OpenStack Kolla-Ansible

Her vil det bli vist og beskrevet de forskjellige feilmeldingene og problemene vi støtet på. Hver seksjon i dette kapitellet beskriver forsøket av kjørt kommando, hva problemet virker å komme fra og hvordan vi prøvde å løse det. Kolla-Ansible ble forsøkt implementert på fler forskjellige måter, blant annet med Ubuntu 16.04 og Ubuntu 18.04, og i tillegg i fler forskjellige mindre arkitekturoppsett.

A.1 ansible -i multinode.ini -m ping all

Første problem vi støtte på var kommunikasjon mellom nodene.

```
192.168.51.208 | FAILED! => {
    "changed": false,
    "failed": true,
    ‘‘module stderr": ‘‘Shared connection to 192.168.51.208 closed.\r\n",
    ‘‘module stdout": ‘‘sudo: unable to resolve host keystone\r\n/bin/sh:
    1: /usr/bin/python: not found\r\n",
    "msg": ‘‘MODULE FAILURE",
    "rc" : 0
}
192.168.51.203 | FAILED! => {
    "changed": false,
    "failed": true,
    ‘‘module stderr": ‘‘Shared connection to 192.168.51.203 closed.\r\n",
    "module_stdout": ‘‘sudo: unable to resolve host nova\r\n/bin/sh: 1:
    /usr/bin/python: not found\r\n",
    "msg": ‘‘MODULE FAILURE",
    "rc" : 0
}
```

Feilmeldingen oppstod på grunn av at alle nodene var avhengig av å ha python installert før kommunkasjon kunne bli opprettet. Følgende kommandoer ble kjørt for å komme videre:

```
apt-get update
apt-get install python-pip
pip install -U pip
```

A.2 kolla-ansible -i multinode.ini bootstrap-servers

Bootstrap-servers er en kommando for Kolla-Ansible som foretar konfigurasjoner før selve konteiner-deplyomenten. Kommandoen gjør endringer i etc/hosts, oppretter bruker og grupper, pakke-installeringer og avinstalleringer og mer [76].

Feilen virker å komme fra versjonskollisjon med ubuntu 18.04 og Docker: Feil med at Docker ikke støtter ubuntu 18.04 så umulig å kjøre apt-get update.

Dette var ikke tilfellet etter mer søking. Docker støtter ubuntu 18.04. Vi har prøvd å rebuilde instansene fra 18.04 til 16.04, da virket det. En annen ting vi gjorde var å starte en ny instanse i ubuntu 18.04 for å installere docker igjen på samme måte for også apt-get update for hvert skritt for å finne ut hvor det feilet. Da gikk alt fint, så feilen klarte ikke å bli gjenskapt.

Docker restart feiler:

```
Unable to restart service docker: Job for docker service failed
because the control process exited with error code.
```

Problemer med pip, mulig versjonen ikke er kompatibel med den Kolla-ansible bruker, eventuelt Docker pakken har noen problemer.

Bootstrap-kommandoen virker ikke å være idempotent, kan være noe med at vi har feil versjoner eller lignende. Første gang kommandoen kjøres kommer den opp til 36 OK-statuser men kjøres kommandoen igjen stopper den på seks OK-statuser fordi den har gjort endringer på nodene.

Problemet med bootstrap-kommandoen er at vi hadde allerede installert programvare på maskinene vi prøvde å bootstrappe. Det førte til kollisjoner mellom versjoner, pakker og lignende. Hvis alle nodene er helt tomme før bootstrappen, vil den sette opp alle avhengigheter på riktig måte.

A.3 kolla-ansible -i multinode prechecks

Ved installering av Ansible vil versjon 2.0.0.2 bli installert automatisk, selv om denne er utdatert. Man må manuelt gi versjons nummer eller installere ny versjon ettersom Kolla-ansible er avhengig av Ansible 2.4 eller nyere. Dette kan bli fikset med:

```
sudo -H pip install ansible==2.7
```

Det er også viktig å ha en oppdatert version av Ansible, standard versjon er (2.0.4)

```
sudo apt install ansible==2.4
```

Etter dette problemet ble fikset, gikk testen litt videre og stoppet med *Checking Docker Version*.

```
TASK [prechecks : Checking Docker Version] ****************************
*************************************************************************
***

fatal: [192.168.51.208]: FAILED! => {"changed": false, "cmd": "docker
--version", "failed": true, ‘‘failed_when_result": true, "msg": "[Errno 2]
```

```
No such file or directory", "rc": 2}
fatal: [192.168.51.203]: FAILED! => {"changed": false, "cmd": "docker
--version", "failed": true, ‘‘failed_when_result": true, "msg": "[Errno 2]
No such file or directory", "rc": 2}
to retry, use: --limit @/usr/local/share/kolla-ansible/ansible/site
.retry

PLAY RECAP *********************************************************
********************************************************************
***

192.168.51.203      : ok=4  changed=0   unreachable=0   failed=1
192.168.51.208      : ok=4  changed=0   unreachable=0   failed=1
```

Feilmeldingen viser til at kommandoen *docker –version* feiler. Det kommer av at Docker er nødt til å være installert på alle nodene før prechecken går igjennom. Dette ble gjort ved kommandoene:

```
curl -sSL https://get.docker.io | bash
sudo apt-get update
sudo apt-get upgrade -y docker
```

Etter kjørt kommandoer ble precheck kjørt igjen med følgende feil:

```
TASK [prechecks : Checking Docker Version] *****************************
********************************************************************
***

fatal: [192.168.51.208]: FAILED! => {"changed": false, "cmd": ["/usr/bin/
python", "-c", ‘‘import docker; print Docker. version "J, "delta": "0:00:0
0.007776", "end": "2019-02-11 09:54:54.920592", "failed": true, ‘‘failed_w
hen_result": true, "msg": "non-zero return code", ‘‘rc": 1, ‘‘start": "2019
-02-11 09:54:54.912816", ‘‘stderr": Traceback (most recent call last):\n F
ile \"<string>\", line 1, in <module>\nImportError: No module named docke
r", ‘‘stderr_lines": [‘‘Traceback (most recent call last):", " File \"<stri
ng>\", line 1, in <module>", "‘‘ImportError: No module named docker"], "st
dout": "", ‘‘stdout_lines": {]}
fatal: [192.168.51.203]: FAILED! => {"changed": false, "cmd": ["/usr/bin/
python", "-c", ‘‘import docker; print Docker. version "J, "delta": "0:00:0
0.007977", "end": "2019-02-11 09:54:54.912885", "failed": true, ‘‘failed_w
hen_result": true, "msg": "non-zero return code", ‘‘rc": 1, ‘‘start": "2019
-02-11 09:54:54.904908", ‘‘stderr": Traceback (most recent call last):\n F
ile \"<string>\", line 1, in <module>\nImportError: No module named docke
r", ‘‘stderr_lines": [‘‘Traceback (most recent call last):", " File \"<stri
ng>\", line 1, in <module>", "‘‘ImportError: No module named docker"], "st
dout": "", ‘‘stdout_lines": {]}
to retry, use: --limit @/usr/local/share/kolla-ansible/ansible/site.retry
```

```
PLAY RECAP *********************************************************
*********************************************************************
***

192.168.51.203      : ok=7  changed=0   unreachable=0   failed=1
192.168.51.208      : ok=8  changed=0   unreachable=0   failed=1

Command failed ansible-playbook -i multinode.ini -e @/etc/kolla/globals.yml
-e @/etc/kolla/passwords.yml -e CONFIG_DIR=/etc/kolla -e kolla_action=prech
eck /usr/local/share/kolla-ansible/ansible/site.yml
```

Problemet ble feilsøkt og oppstod ved at kolla ikke hadde aktivert shared mounts. Det ble gjort med følgende kommandoer:

```
sudo mkdir -p /etc/systemd/system/docker.service.d
cat <<EOF > /etc/systemd/system/docker.service.d/kolla.conf
[Service]
MountFlags=shared
EOF
```

Docker-tjenesten ble restartet og docker ble installert via pip:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
pip install -U docker
```

Problemet ble fikset og precheck forsøkt kjørt igjen, denne gangen med følgende feil:

```
TASK [rabbitmq : Check if all rabbit hostnames are resolvable] **********
*********************************************************************
***

failed: [192.168.51.208] (item=192.168.51.208) => {"changed": false, ‘‘cmd
": ["getent", "ahostsv4, ‘‘keystone"], "delta": "0:00:00.002294", "end": "
2019-02-11 10:03:07.617613", "failed": true, "item": "192.168.51.208", "m
sg": "non-zero return code", "rc": 2, "start": "2019-02-11 10:03:07.61
5319", "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines": []}
to retry, use: --limit @/usr/local/share/kolla-ansible/ansible/site.retry

PLAY RECAP *********************************************************
*********************************************************************
***

192.168.51.203 : ok=13 changed=0 unreachable=0 failed=0
192.168.51.208 : ok=30 changed=0 unreachable=0 failed=1

Command failed ansible-playbook -i multinode.ini -e @/etc/kolla/globals.y
```

```
ml -e @/etc/kolla/passwords.yml -e CONFIG DIR=/etc/kolla -e kolla_action=
precheck /usr/local/share/kolla-ansible/ansible/site.yml
```

Feilen her kom av at vi ikke hadde lagt inn nodene i etc/hosts. Følgende linjer ble lagt inn i filen:

```
192.168.51.203 nova
192.168.51.208 keystone
```

A.4 kolla-ansible -i multinode deploy

Deploy-kommandoen virket å gå bra helt til vi la merke til at RabbitMQ-konteineren gikk i konstant restartings-loop. Etter feilsøking kom vi frem til at det kunne ha noe med at konteiner-imaget var ødelagt [77] [78]. Feilmeldingen vi fikk når vi listet opp kjørende konteinere:

```
Mariadb container existed with non zero return code.
Waiting for master Mariadb
```

Konteineren utfører ti forsøk helt til den feiler. Dette er på grunnen av at mariadb-konteinere allerede er oppe å kjører men går i en evig restart loop. Problemet virket å komme fra *kolla-ansible -i multinode.ini bootstrap-servers* kommandoen. Det virket å være en bug med bootstrappen som gjør at den setter opp ting på feil måte og gjør sånn at mariadb går i uendelig restarting loop. Dette stemte ikke, da vi satt opp stacken på nytt, uten å kjøre bootstrap-kommandoen. Problemet oppstod fortsatt og RabbitMQ-konteineren fortsatte å gå i evig loop. Vi inspiserte logfilen i mariadb og fant kun følgende nytting informasjon:

```
Neither host 'keystone' nor 'localhost' could be looked up with
'/usr/sbin/resolveip'
Please configure the 'hostname' command to return a correct hostname.
```

Utifra denne informasjonen virket problemet å komme fra DNS. Men vi hadde allerede lagt inn korrekte navn og tilsvarende ip-adresser og verifisert at dette virket.

På dette tidspunktet satt vi lenge fast på feilsøking av hvorfor konteineren ikke ville starte. Det ble forsøkt løst på fler forskjellige måter, men uten hell. Litt tilfeldig tenkte vi at det kunne være feil med Ubuntu-imaget vi brukte uten at vi fant noe treff på det spesifikke problemet. Vi skiftet konteiner-imagene fra Ubuntu til CentOS og det løste problemet. Endelig hadde vi fått en deployment igjennom med Kolla-Ansible og vi kunne begynne med testing, rutinearbeid og vurderinger.

A.5 Kolla-ansible -i multinode deploy med ekstern konteiner database

```
TASK [keystone : Creating default user role] ***************************
**********************************************************************
***
fatal: [agent1]: FAILED! => {"changed": false, "failed": true, "module_st
derr": "No handlers could be found for logger \"keystoneauth.identity.gen
```

```
eric.base\"\nTraceback (most recent call last):\n  File \"/tmp/ansible_lk
pV57/ansible_module_os_keystone_role.py\", line 136, in <module>\n    mai
n()\n  File \"/tmp/ansible_lkpV57/ansible_module_os_keystone_role.py\", l
ine 107, in main\n    role = cloud.get_role(name)\n  File \"</opt/ansible
/lib/python2.7/site-packages/decorator.pyc:decorator-gen-68>\", line 2, i
n get_role\n  File \"/opt/ansible/lib/python2.7/site-packages/shade/_util
s.py\", line 410, in func_wrapper\n    return func(*args, **kwargs)\n  Fi
le \"/opt/ansible/lib/python2.7/site-packages/shade/openstackcloud.py\",
line 10379, in get_role\n
...
...
...
\n  File \"/opt/ansible/lib/python2.7/site-packages/keystoneauth1/identit
y/generic/base.py\", line 161, in _do_create_plugin\n    'auth_url is cor
rect. %s' % e)\nkeystoneauth1.exceptions.discovery.DiscoveryFailure: Coul
d not find versioned identity endpoints when attempting to authenticate.
Please check that your auth_url is correct. Unable to establish connectio
n to http://192.168.51.208:35357: HTTPConnectionPool(host='192.168.51.208
', port=35357): Max retries exceeded with url: / (Caused by NewConnection
Error('<urllib3.connection.HTTPConnection object at 0x7f58014048d0>: Fail
ed to establish a new connection: [Errno 111] Connection refused',))\n",
"module_stdout": "", "msg": "MODULE FAILURE"}
```

Feilen kommer av at keystone og mariadb kjører på forskjellige instanser. En måte å fikse det på er ved å spesifisere at keystone og mariadb bruker control-gruppen slik at keystone og mariadb ligger på samme node. Vi ønsker derimot å ha Keystone og Mariadb på seperate noder for realismens skyld.

Feilen ble fikset ved at vi hadde satt feil VIP adresse i globals.yaml. Den måtte endres til adressen keystone ligger på, ikke databasen. Med den endringen fikk vi satt opp en keystone-konteiner som kommuniserte med en separat mariadb-konteiner.

A.6 kolla-ansible -i multinode deploy med ekstern ikke-konteiner database

Vi ville få keystone til å kommunisere med en database instanse og ikke bare en database-konteiner for å replikere arkitekturen til SkyHiGh.

```
TASK [keystone : Creating admin project, user, role, service, and endpoin
t] ************************************************************************
*************************************
fatal: [agent1]: FAILED! => {"failed": true, "msg": "The conditional chec
k '(keystone_bootstrap.stdout | from_json).changed' failed. The error was
: Expecting ',' delimiter: line 1 column 150 (char 149)"}
```

Denne feilen kom av at databasen allerede eksisterte fra før av når vi prøvde å deploye. Vet ikke helt hva feilmeldingen har med problemet å gjøre. Det ble fikset ved å slette databasen helt og kjøre uten database gjorde slik av feilmeldingen gikk vekk.

B Feilsøking av OpenStack-Helm

Som nevnt tidligere i rapporten er OpenStack-Helm det mest komplekse av de vurderte teknologiene. Prosjektgruppen merket fort mangel på grunnlegende kunnskap når det kom til Kubernetes, det førte til store vanskeligheter med feilsøking og implementering av OpenStack-Helm. Oppsett av Helm ble forsøkt på litt forskjellige måter, blant annet Ubuntu 18.04 og Ubuntu 16.04. I tilegg til det ble det forsøkt satt opp en AllInOne deployment og en multinode deployment. Instanser ble og satt opp i fler forskjellige størrelser for å sørge for at CPU kraft ikke var problemet.

B.1 AllInOne

Et AllInOne oppsett var aldri svært relevant for et faktisk proof-of-concept ettersom det ikke vil være ønskelig oppsett for arbeidsgiver. Det ble forsøkt implementert på grunnlag av at gruppen kunne enkelt få litt erfaring og resultater ved oppsett av AllInOne. Det ville også være mulig å leke litt med oppsettet uten å være redd for å ødelegge noe.
Fulgte guide for all-in-one: https://careyscloud.ie/openstack_kube

Fikk problemer med at br-ex ikke finnes.

To gruppemedlemmer prøvde oppsett av AllInOne samtidig, ved bruk av gitt guide. Kun en av gruppemeldemmene fikk problemet. Da ble det bare forsøkt å starte ny stack og prøve igjen. Ved nytt forsøk fikk begge gruppemedlemmene AllInOne-oppsettet til å virke og feilen klarte ikke å bli gjenskapt så det var ikke mye feilsøking på problemet.

Et annet problem som dukket opp i ett AllInOne-oppsett var da det ble forsøkt å settes opp på Ubuntu 18.04. Den samme guiden som tidligere nevnt ble fulgt men med følgende feil:

```
failed: [local] (item=[u'docker.io=18.06.1-0ubuntu1.2~16.04.1']) =>
{"cache_update_time": 1552570956, "cache_updated": false, "changed":
false, "item": ["docker.io=18.06.1-0ubuntu1.2~16.04.1"], "msg": "'/usr/bin
/apt-get -y -o \"Dpkg::Options::=--force-confdef\" -o \"Dpkg::Options::=-
-force-confold\"
install 'docker.io=18.06.1-0ubuntu1.2~16.04.1'' failed: E: Version
'18.06.1-0ubuntu1.2~16.04.1' for 'docker.io' was not found\n", "rc": 100,
"stderr": "E: Version '18.06.1-0ubuntu1.2~16.04.1' for 'docker.io' was not
found\n", "stderr_lines": ["E: Version '18.06.1-0ubuntu1.2~16.04.1' for
'docker.io' was not found"], "stdout": "
Reading package lists...\nBuilding dependency tree...\nReading state
information...\n", "stdout_lines": ["Reading package lists...", "Building
dependency tree...", "Reading state information..."]}
```

Det viktigiste utifra denne feilmeldingen som gav oss nyttig informasjon vi kunne tyde:

```
install 'docker.io=18.06.1-0ubuntu1.2~16.04.1'' failed: E: Version
'18.06.1-0ubuntu1.2~16.04.1' for 'docker.io' was not found
```

Meldingen tyder på at Helm får docker til å installere en docker versjon som kun er støttet av Ubuntu 16.04 eller eldre. Stack ble slettet og satt opp på ny med versjon 16.04, da gikk oppsettet smertefritt.

B.2 Multinode

OpenStack Helm i et multinode oppsett var fra start sett på som god løsning. Multinode passer arbeidsgivers arkitektur og Openstack-Helm er bevist god løsning på konteinerisering av OpenStack fra større bedrifter. Problemet er kompleksiteten.

Prosjektgruppen har nærmest ingen tidligere erfaring med Kubernetes, kun veldig grunnleggende. Derfor har OpenStack-Helm vært svært utfordrende, men det har vært forsøkt å gjøre systematisk feilsøking for å komme seg forbi problemene men det har vist seg krevende.

Initielt oppsett av Kubernetes gjort ved følgende guide: https://docs.openstack.org/openstack-helm/latest/install/kubernetes-gate.html

Guiden er en slags forutsetnings guide før man setter opp og tar i bruk Helm. Det har ikke oppstått noe særlig problemer i denne guiden, bare noe småproblemer som at ssh-copy-id ikke virker, og at openstack-helm katalogen naturligvis ikke finnes i etc, og må opprettes manuelt uten at det står i guiden. Vi kom oss fint forbi ssh-copy-id feilen ved å bare kopiere nøklene over manuelt over ssh. Følgende pods kjører etter guiden er ferdig.

```
NAMESPACE     NAME                                        READY   STATUS
RESTARTS   AGE     IP               NODE      NOMINATED NODE
READINESS GATES
kube-system   calico-etcd-rk7tb                           1/1     Running
0          8m36s   192.168.51.213   kriger1   <none>
<none>
kube-system   calico-kube-controllers-6b87b685c6-7ngx7    1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   calico-node-8qnrg                           1/1     Running
2          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   calico-node-dfnzv                           1/1     Running
0          4m55s   192.168.51.209   kriger2   <none>
<none>
kube-system   calico-settings-dxslp                       0/1     Completed
3          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   etcd-kriger1                                1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
```

```
kube-system   kube-apiserver-kriger1                    1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   kube-controller-manager-kriger1           1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   kube-dns-7448997df8-t57m6                 3/3     Running
0          6m56s   192.168.35.65    kriger1   <none>
<none>
kube-system   kube-proxy-kl725                          1/1     Running
0          4m55s   192.168.51.209   kriger2   <none>
<none>
kube-system   kube-proxy-ntx7d                          1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   kube-scheduler-kriger1                    1/1     Running
0          10m     192.168.51.213   kriger1   <none>
<none>
kube-system   tiller-deploy-8585cdc7c9-5lw7p            1/1     Running
0          6m28s   192.168.35.66    kriger1   <none>
<none>
```

Etter forutsetningsguiden er satt opp blir selve OpenStack-Helm installert etter følgende guide: https://docs.openstack.org/openstack-helm/latest/install/multinode.html

Her vil man bare steg for steg kjøre scripts som setter opp de forskjellige tjenestene og avhengigheter. Problemet med guiden er at det står veldig lite hva som faktisk skjer og ingenting om hva som kan gå galt. Ved første script som blir kjørt:

```
./tools/deployment/multinode/010-setup-client.sh
```

Det har normalt ikke oppstått noen feil ved dette scriptet. Forventet pods blir startet og kjører. Det er først ved neste script problemene oppstår.

```
./tools/deployment/multinode/020-ingress.sh

Containers failed to start after 900 seconds

NAME                                    READY     STATUS
RESTARTS   AGE       IP               NODE
ingress-5cd9655495-ptbd6                0/1       CrashLoopBackOff
9          15m       192.168.227.194  kriger2
ingress-5cd9655495-zglt8                1/1       Running
0          15m       192.168.35.69    kriger1
ingress-error-pages-7f574d9cd7-2x896    1/1       Running
0          15m       192.168.227.195  kriger2
ingress-error-pages-7f574d9cd7-fb4c5    1/1       Running
```

```
0          15m       192.168.35.68     kriger1

Some pods are not ready
```

Ved nærmere inspeksjon ved bruk av kommandoen:

```
kubectl describe pods PODNAVN -n NAMESPACE
```

Får man følgende output:

```
Events:
  Type     Reason             Age                From
  Message
  ----     ------             ----               ----
  -------
  Normal   Scheduled          27m                default-scheduler
  Successfully assigned openstack/ingress-748786cdfc-4npgt to kriger2
  Normal   Pulled             27m                kubelet, kriger2
  Container image "quay.io/stackanetes/kubernetes-entrypoint:v0.3.1"
  already present on machine
  Normal   Created            27m                kubelet, kriger2
  Created container
  Normal   Started            27m                kubelet, kriger2
  Started container
  Normal   Pulled             27m (x2 over 27m)  kubelet, kriger2
  Container image "quay.io/kubernetes-ingress-controller/nginx-ingress-
  controller:0.23.0" already present on machine
  Normal   Created            27m (x2 over 27m)  kubelet, kriger2
  Created container
  Normal   Started            27m (x2 over 27m)  kubelet, kriger2
  Started container
  Normal   Killing            27m                kubelet, kriger2
  Killing container with id docker://ingress:Container failed liveness
  probe.. Container will be killed and recreated.
  Warning  Unhealthy          26m (x6 over 27m)  kubelet, kriger2
  Liveness probe failed: Get http://192.168.227.195:10254/healthz:
  dial tcp 192.168.227.195:10254: connect: connection refused
  Warning  FailedPreStopHook  26m (x2 over 27m)  kubelet, kriger2
  Exec lifecycle hook ([/tmp/ingress-controller.sh stop]) for Container
  "ingress" in Pod "ingress-748786cdfc-4npgt_openstack(13923d88-4bc0-11
  e9-a4be-fa163e3e1bbf)" failed - error: command '/tmp/ingress-controll
  er.sh stop' exited with 1: + COMMAND=stop
+ stop
+ kill -TERM 1
/tmp/ingress-controller.sh: line 26: kill: (1) - Operation not permitted
, message: "+ COMMAND=stop\n+ stop\n+ kill -TERM 1\n/tmp/ingress-
controller.sh: line 26: kill: (1) - Operation not permitted\n"
  Warning  Unhealthy  12m (x35 over 27m)    kubelet, kriger2  Readiness
```

```
probe failed: Get http://192.168.227.195:10254/healthz: dial tcp
192.168.227.195:10254: connect: connection refused
Warning  BackOff    2m51s (x85 over 24m)  kubelet, kriger2  Back-off
restarting failed container
```

Problemet med at ingress konteinerene står i status crashloopbackoff virker å komme fra at instansen ikke har nok ressurser til å starte podden innen gitt tidsramme. Readiness checken blir rett og slett ikke gitt nok tid til å starte i følge [79][80][81]. Det kan fikses på to forskjellige måter. Enten ved å øke cpu-kraften til instansen, eller ved å øke tiden den venter på pods slik at den ikke går tom for tid. Forsøkte å endre på initialDelaySeconds under livenessProbe i deployment-ingress.yaml og deployment-error.yaml i /ingress/templates katalogen, uten hell. Eneste utvei virker da å gi mer kraft til instansene. Etter økt kapasitet på instansene fant vi også ut at vi får samme problem med m1.large flavor, isteden for m1.medium.

Det ble forsøkt å gå videre uten hensyn til feilmeldingene for å se om resultater var oppnåelig. Følgende scripts fra guiden ble kjørt:

```
./tools/deployment/multinode/050-mariadb.sh
./tools/deployment/multinode/070-memcached.sh
./tools/deployment/multinode/080-keystone.sh
```

Scriptene fullføres uten feilmeldinger men ved å liste opp podsene er det mange av dem som ikke kjører. Enten dem sitter fast i pending eller i crashloopback. Her var det ingenting av relevante treff fra nettet som kunne hjelpe oss. Eneste måten å feilsøke på var å sjekke podsene, logger og konteinere. Feilsøking ble utført utifra følgende dokument [82]. Dette er følgende 'kubectl describe' feilmeldinger for de nyeste kjørte scriptene:

openstack ingress:

```
Warning  Unhealthy  23m (x2744 over 2d)  kubelet, kriger2  Liveness probe
failed: Get http://192.168.227.194:10254/healthz: dial tcp
192.168.227.194:10254: getsockopt: connection refused

Warning  BackOff    3m (x16455 over 2d)  kubelet, kriger2  Back-off
restarting failed container
```

openstack keystone-api, ingress, mariadb-ingress:

```
State:          Waiting
Reason:       PodInitializing
Ready:          False

Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
```

openstack mariadb-server:

```
Events:
  Type     Reason            Age                  From               Message
  ----     ------            ----                 ----               -------
  Warning  FailedScheduling  1m (x13812 over 2d)  default-scheduler  pod has
  unbound PersistentVolumeClaims (repeated 2 times)
```

openstack rabbitmq-rabbitmq:

```
Events:
  Type     Reason            Age                  From               Message
  ----     ------            ----                 ----               -------
  Warning  FailedScheduling  1m (x24186 over 4d)  default-scheduler  pod has
  unbound PersistentVolumeClaims (repeated 2 times)
```

Videre problemer funnet i docker ingress-kontainerene med docker logs kommando:

```
W0316 10:25:45.001059      30 controller.go:846] Service "openstack/rabbitmq"
does not have any active Endpoint.
W0316 10:25:45.001093      30 controller.go:846] Service
"openstack/keystone-api" does not have any active Endpoint.
W0316 10:25:48.334490      30 controller.go:846] Service "openstack/rabbitmq"
does not have any active Endpoint.
W0316 10:25:48.334522      30 controller.go:846] Service
"openstack/keystone-api" does not have any active Endpoint.
W0316 10:25:51.667828      30 controller.go:846] Service "openstack/rabbitmq"
does not have any active Endpoint.
W0316 10:25:51.667856      30 controller.go:846] Service
"openstack/keystone-api" does not have any active Endpoint.
W0316 10:47:14.380408      30 reflector.go:270] k8s.io/ingress-nginx/internal
/ingress/controller/store/store.go:152: watch of *v1.Endpoints ended with:
too old resource version: 118248 (11$
720)

W0315 14:47:00.913854      26 configmap.go:121]  is not a valid textual
representation of an IP address

E0315 14:47:02.972585      26 checker.go:57] healthcheck error: 500
```

På dette tidspunktet stod vi ovenfår en god del problemer, så vi måtte systematisk feilsøke hver og en av dem utifra feilmeldingene vi fikk fra podsene og konteinerne. Roten av problemene kommer av at ingress konteineren sitter fast i restart. Det ble kun funnet to løsninger på problemet som ble nevnt tidligere, begge ble forsøkt og ingen virket. Når de eneste to løsningene som blir funnet på nettet ikke har noe effekt på problemet og prosjektgruppen har begrenset med bakgrunnskunnskaper med Kubernetes, oppstår det en vanskelig situasjon som etter vurdering ikke er hensiktsmessig å gå videre på når vi allerede arbeider med to andre løsninger.

B.3 Eget oppsett av Kubernetes

Når multinode-oppsettet ikke virket satte vi opp Kubernetes kluster på eget vis uten bruk av guider fra OpenStack. Dette oppsettet ble forsøkt for å verifisere at det skyldes kluster vi hadde satt opp ved bruk OpenStack-guiden https://docs.openstack.org/openstack-helm/latest/install/kubernetes-gate.html. Derfor satte gruppen opp et kluster-oppsett i henhold til Kubernetes sin dokumentasjon: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/
Gruppen testet at deployment fungerte for å bekrefte at klusteret var riktig satt opp. Dette ble gjort ved bruk denne guiden: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Her deployet gruppen fungerende Nginx. Dette klusteret besto av en mester- og to workernoder. Gruppen fikk samme problemer som med forrige klusteret (OpenStacks oppsett) B.2 når vi forsøkte å deploye OpenStack komponentene ved bruk av skriptene til OpenStack-Helm.

```
    ===== Processing [neutron] chart =====
make[1]: Entering directory '/opt/openstack-helm'
if [ -f neutron/Makefile ]; then make -C neutron; fi
if [ -f neutron/requirements.yaml ]; then helm dep up neutron; fi
/bin/bash: helm: command not found
Makefile:34: recipe for target 'init-neutron' failed
make[1]: *** [init-neutron] Error 127
make[1]: Leaving directory '/opt/openstack-helm'
Makefile:27: recipe for target 'neutron' failed
make: *** [neutron] Error 2
```

C Kode

C.1 Puppet-kode for oppsett av Kolla-Ansible

```
#
# profile::base base klassen for felles konfigurasjoner
#
class profile::base {

  # the base profile should include component modules that will be on all nodes

  # installing python and pip
  class { 'python' :
    version    => 'system',
    pip        => 'latest',
  }

  # pip docker package
  python::pip { 'docker':
    ensure    => 'latest',
    require   => Class['python'],
  }

  # docker latest
  class { 'docker':
    version => 'latest',
    before  => File['/etc/systemd/system/docker.service.d/kolla.conf'],
  }

  # ntp
  class { 'ntp':

  }

  # Setting the timezone
  class { 'timezone':
    timezone => 'Europe/Oslo',
  }

  # tee /etc/systemd/system/docker.service.d/kolla.conf >> [Service] MountFlags=shared
  file { '/etc/systemd/system/docker.service.d/kolla.conf':
    ensure => file,
    path => '/etc/systemd/system/docker.service.d/kolla.conf',
```

```
    replace => 'no',
    content => "[Service]\n MountFlags=shared",
  }

  package {'mariadb-client-core-10.1':
     ensure    => present,
  }
}

#
# The manager profile class
#
class profile::manager {

  $ansible_config = lookup('ansible_config')
  $globals_config = lookup('globals_config')
  $multinode_config = lookup('multinode_config')

   package {'ansible':
     ensure   => 'latest',
   }

   python::pip { 'ansible':
     ensure   => '2.4',
   }

   python::pip { 'kolla-ansible':
     ensure   => 'latest',
     before   => File['kolla', 'inventory', '/root/kolla-ansible/etc/kolla'],
   }

   # copying some folders
  file {'kolla':
    ensure   => 'directory',
    source   => '/usr/local/share/kolla-ansible/etc_examples/kolla',
    recurse  => 'remote',
    path     => '/etc/',
  }

  file {'inventory':
    ensure   => 'directory',
    source   => '/usr/local/share/kolla-ansible/ansible/inventory/',
    recurse  => 'true',
    path     => '/root/'
   }
```

```
file {'/etc/ansible/ansible.cfg':
  ensure   => 'present',
  content  => $ansible_config,
  require  => Package['ansible'],
}

vcsrepo { '/root/kolla':
  ensure   => 'present',
  provider => 'git',
  source   => 'https://github.com/openstack/kolla.git',
}

vcsrepo { '/root/kolla-ansible':
  ensure   => 'present',
  provider => 'git',
  source   => 'https://github.com/openstack/kolla-ansible.git',
}


file {'/root/kolla-ansible/etc/kolla':
   ensure   => 'directory',
   source   => '/root/kolla-ansible/etc/kolla/',
   recurse  => 'true',
   path     => '/etc/kolla/',
   before   => File['/etc/kolla/globals.yml'],
}

# globals_config
file {'/etc/kolla/globals.yml':
  ensure   => 'present',
  content  => $globals_config,
}

# multinode_config
file {'/root/multinode.ini':
  ensure   => 'present',
  content  => $multinode_config,
}

# apt-get install -y python-dev libffi-dev gcc libssl-dev python-selinux python-setuptools
package {['libffi-dev', 'gcc', 'libssl-dev', 'python-selinux', 'python-setuptools']:
  ensure    =>  'present',
  provider  =>  'apt',
}

# Installing requirements for kolla and kolla-ansible
```

```
  python::requirements { '/root/kolla/requirements.txt':
    require    => Vcsrepo['/root/kolla'],
  }

  python::requirements { '/root/kolla-ansible/requirements.txt':
    require    => Vcsrepo['/root/kolla-ansible'],
    before     => Exec['generate pwds'],
  }
  # Creating ssh key
  ssh_keygen { 'root':
     filename  => '/root/.ssh/ssh_host_rsa_key',
    # before   => Exec['copy file'],
  }

  # Generating pwd for /etc/kolla/passwords.yml
  exec {'generate pwds':
     command   => 'kolla-genpwd',
     path      => ['/bin/bash -c', '/sbin/bash -c', '/usr/bin/which',
     '/usr/local/bin/kolla-genpwd'],
     onlyif    => 'which kolla-genpwd',
  }

  exec {'prechecks':
     command   => 'kolla-ansible -i /root/multinode.ini prechecks -v',
     path     => ['/usr/local/bin/kolla-ansible', '/usr/bin/which'],
     onlyif    => 'which kolla-ansible',
  }

}
```

D Møtereferat

Mandagsmøte 14.01.19 - kl 12:00

Første møtet med Eigil, Lars Erik og Hjelmås.

Hva vi har gjort:
-

Hva er neste:
- Komme ordentlig igang, skrevet ned regler, starte på prosjektplan, bestemme metodologi, faste møter.

Utfordringer vi har hatt:
-

Spørsmål:

Selve møtet:
- Se Cern- installasjonen av openstack (hj)
- Sjekk først kolla om den fungerer som forventet og om den løser utfordringene (Eigil)
- Sett opp prosjektplan
- Start med en komponent i openstack (keystone)

Vi hadde møte med veilederen vår Erik Hjelmås, og oppdragsgiverne Eigil Obrestad og Lars Erik Pedersen. Erik anbefalte oss å ta en kikk på CERN sin OpenStack-implementasjon, da han mente denne kunne vært interessant mtp. containerbruk. Han nevnte også en gruppe som hadde en bacheloroppgave i fjor som muligens kunne være interessant for oss. Erik poengterte at oppgaven er en utredningsoppgave, og at det derfor er viktig at vi har gode vurderinger og begrunnelser for valgene vi tar og konklusjonene/påstandene vi kommer med. Eigil og Lars Erik fortalte litt mer om oppgaven og sine forventninger og målsettinger for prosjektet, men vi må fortsatt få ned det litt mer detaljert senere. Vi fikk et lynkurs av Eigil og Lars Erik i hvordan Skyhigh/Skylow fungerer, og diskuterte litt hva der som ville være relevant for oss. De mente at Keystone mest sannsynlig ville være den enkleste prosessen å starte med.

Etter møtet satt vi oss ned for å sette agenda for uken. Vi tok en kikk på hva vi må ha med i prosjektplan og hva vi burde starte med. Vi skrev ned grupperegler og bestemte arbeidstider. Over de neste dagene skal vi hver enkelt tenke over våre målsettinger og læringsmål ved prosjektet, og skrive de ned mot slutten av uka. Vi skal lese oss opp på den relevante teknologien så vi har et godt utgangspunkt da prosjektplanen er ferdig. Vi bestemte oss for å bruke Scrum med sprinter på 1 eller 2 uker, hvor møtene kan være på mandag/fredag. Per dags dato ser vi ikke noe stort behov for en gruppeleder, da vi alle har

arbeidet på flere prosjekter sammen før ser vi sannsynligheten for større uenigheter som redusert, og siden gruppen har 3 medlemmer vil flertallet alltid kunne bestemme.

Mandagsmøte 21.01.19 - 12:00

Hva vi har gjort:
- Researchet tidligere prosjekter.
- Laget første utkast for forprosjektet.
- Satt oss litt inn i OpenStack Kolla.

Hva er neste:
- Ferdigstille forprosjektet.
- Sette oss bedre inn i Kolla og gjøre tester av dette i skyhigh.

Utfordringer vi har hatt:
-

Spørsmål:
- Mer definert oppgave? Hva er sluttmålet vårt?
- Hvor legger vi grunnlaget for bruk av kilde?
- Har du sett på utkastet av forprosjektet vårt?

Selve møtet:
- nytt møte tirsdag 22.01.19 angående prosjektplan kl 14:00

- matrialkurs og pensum på blackboard finner vi eksempler på relevante oppgaver.

- mange gode videoer forelesninger osv. anbefalt av hjelmås.

- **spørsmål 1, Mer definert oppgave? Hva er sluttmålet vårt?**
  Definert oppgave er hovedsakelig en teknisk anbefaling av en prototype implementasjon. Kanskje ikke bare implementeringen men hvordan vil det virke i oppgraderinger osv.

  Har Keystone i seg selv database? Alle tjenester har vær sin database. Alle databaser kjører i et mysql database-cluster. Starter med å kun gjøre Keystone-tjenesten konteinerisert, deretter kanskje gjør hva som er bak og konteinerisert. Det er ønskelig at isteden for 2 dedikerte VM-er som kjører keystone, så kan det kjøres 5 konteinere som er lett å skalere. Noen bruker og memcache og er kanskje innom rabbitMQ.

  Helm er en måte å beskrive en samling av konteinere på.

Prototype = implementering i skylow så er det opp til Lars Erik og Eigil og ta det inn i produksjon.

Ikke krise om man gjør det den ene eller andre veien så lenge det er godt argumentert for. Høy-nivå mål er å få redusert størrelsen på Puppet-repoet, kan det forenkles med å ha egne konteinere i stedet for at det er så tungt med puppet-konfigurasjonsstyring.

- **spørsmål 2, Hvor legger vi grunnlaget for bruk av kilde?**
  Alle steder der vi skriver om noe som vi har lært et eller annet sted er det greit å sitere kilden. kan og godt sitere videoer. Hver referanse skal det svares på, hvem, hva, hvor, når? ACM digital library for å finne bibtex stuff

- **spontan spørsmål, skal vi bruke template i den endelige rapporten?**
  Se på blackboard. Eigil sier det ligger et sted på github, finner vi ikke den send mail til Tonje.

- **Spørsmål 3, Har du sett på utkastet av forprosjektet vårt?**
  Nei :-(

Mandagsmøte 28.01.19 - 12:00

Hva vi har gjort:
- Videre arbeid med prosjektplan
- Startet med prøving og feiling i Kolla
- Startet med litt generell research

Hva er neste:
- Levere forprosjektet
- Mer prøving mindre feiling
- Videre research på Kolla
- Prøve å få til en Kolla-deployment som virker i skyhigh.

Utfordringer vi har hatt:
- Dårlig dokumentasjon for basic Kolla setup
- Vanskeligheter med å sette seg inn i Kolla
- Mye forskjellige måter og prosjekter litt om hverandre, det fører til mye rot for oss og det er mye bare “halvveis” dokumentasjoner.

Spørsmål:
- Har veileder fått sett på forprosjektet?
- Hva trenger vi for å få integrert en av tjenestene i SkyLow?
- Hva er den enkleste metoden for å verifisere at f.eks keystone faktisk virker?

Selve møtet:

- Sørge for at det vi skriver er rettet mot eget prosjekt (i hovedprosjekt, om teknologier etc)
- Oppgavebeskrivelse så bra ut
- Bare gjøre mer research for å skjønne arkitekturen bedre
- Snakke med Eigil og Lars Erik for mer detaljert om hva som går inn/ut etc
- Spørsmål og tour av serverrom med Eigil:
    - Compute på egne servere
    - Storage på egne servere
        - Databaser i cluster som oppdaterer hverandre
    - Redundancy i de fleste ledd
    - Tjenester på flere servere
        - Eks: Infra 1 / Infra 2 / Infra 3 - 1 Keystone på hver
    - Keystone snakker ikke så mye selv, andre snakker til den
    - Ikke livsfarlig å bruke hiera
    - Ser ut som er ca. samme konfigfiler
    - Sjekk Github ntnusky/ntnuopenstack - roller/profiler etc
    - Verifisere funksjonalitet med OpenStack CLI - openstack catalog list
    - Finne ut av databaser - hvor / hvordan

Mandagsmøte 04.12.19 - 12:00

Hva vi har gjort:

- Levert prosjektplan
- Videre arbeid med kolla hands on
- Videre fordypning i container teknologier

Hva er neste:

- Arbeide mer med kolla både hands on men kanskje litt mer fokus på research
- Starte på hovedrapporten

Utfordringer vi har hatt:

- Sette seg inn i OpenStack Kollas måte å orkestrere containere.
- Deloye containere med Kolla. Dette har vært utfordrende å sette seg inn i. Det er utdaterte dokumentasjoner fører til vanskeligheter for oss.
- Noen av feilmeldingene er vanskelige å sette fingeren på.

Spørsmål:

- Ikke noe møte, veileder fraværende.

Selve møtet:

- Ikke noe møte, veileder fraværende.

Retrospektivt møte:

Hva gikk bra i forrige sprint som vi kan ta med oss videre?

- Arbeidsmengden var bra, møter 4 dager i uken fra 9 til 17 har virket som en solid måte for gruppen å jobbe på.
- Arbeidet bra med prosjektplanen og fikk et resultat vi ble fornøyd med.
- Klart å holde oss foran skjema med god planlegging og kommet tidlig i gang med første oppgave av prosjektet.
- Fulgt Scrum modellen bra som gitt har oss ønsket progresjon.

Hva gikk dårlig i forrige sprint som vi kan gjøre forbedringer på?

- Vi burde gjøre mer research iht. kolla ting som er relatert til den.
- morten må bli mer venn med tmux. Tmux må bli mer venn med **meg!**

Mandagsmøte 11.02.19 - 12:00

Hva vi har gjort:

- Deployet keystone containere tirsdag 05.02. Etter det slettet vi stacken for å gjøre det på en mer strukturert måte igjen men har slitt med å få ting til å virke igjen, akkurat nå har vi fortsatt ikke fått deployet.

Hva er neste:

- Få deployet keystone.
- Begynne å skrive om teknologi og videre på innledning

Utfordringer vi har hatt:

- Deploye på en strukturert måte.
- Lite dokumentasjon på feilmeldinger.

Spørsmål:

- Hva burde være detaljnivået på beskrivelse av teknologier og lignende? Skal vi gå ut ifra at leseren er kjent eller ukjent med innholdet. Skal vi skrive mye om f.eks container teknologier? Skal vi skrive mye om OpenStack og dens arkitektur?
- Hvordan virker kommunikasjonen innad i OpenStack?
- Hvorfor klarer jeg ikke å starte stacken i SkyLow?
- Hvordan skal vi logge oss inn på api serverne?
- Kan Eigil forklare litt rundt OpenStack? Registrere endpoints?

Selve møtet:

- Tegne opp litt angeånde feilen med hostname. Lag hendelseskart elns. Feilsøk systematisk
- 11-12 på tirsdag har hjelmås litt tid og på fredag.
- hvordan distrubueres og hvordan lagres konteinere?
- Se på hvordan man tar i bruk spesifikke tjenster som f.eks keystone bruker en spesifikk database server

Mandagsmøte 18.02.19 - 12:00

Hva vi har gjort:

- Fikk endelig deployet konteinere igjen med Kolla med noen forrutsetninger, vi har ikke helt fått til å få keystone til å bruke ekstern database noe som oppdragsgiver vil ha. Grunnen til vi ikke fikk deployet var pga. MariaDB konteineren som ble laget av Kolla satt fast i en restart loop, det ble fikset med å bytte ut konteiner os-et fra Ubuntu til CentOS.
- Det samme problemet oppstår nå med RabbitMQ konteineren, men vi får fortsatt deployet selv om den sitter fast i en restart loop. Dette kan ha noe med at Keystone er den eneste tjenesten vi har deployet og RabbitMQ som en meldingskø-tjeneste kanskje ikke skjønner hva den skal gjør ved oppstart så den klarer ikke starte. Vi skulle gjerne fjernet RabbitMQ akkurat nå fordi vi har ingen bruk for den men for å deploye via kolla er Keystone avhengig av den og vil ikke lage konteinere om den ikke har tilgang til RabbitMQ.

Hva er neste:

- Sette oss mer inn i hvordan man skriver god rapport, så skrive litt god rapport.
- Sette opp OpenStack Kolla ved bruk av Puppet.

Utfordringer vi har hatt:

- Slitt mye med å få deployet, brukte en hel uke på kun noen små feil som har vært veldig vanskelig å komme seg forbi pga. litt dårlig dokumentasjon, lite community og få treff ved innhenting av informasjon fra feilmeldingene.
- Kommet ordentlig i gang med skriving.

Spørsmål:

-

Selve møtet:

- Ingen møte denne uka

Mandagsmøte 25.02.19 - 12:00

Hva vi har gjort:

- Nesten skrevet ferdig chapter 1 i rapporten og begynt smått med chapter 2 og chapter 3.
- Satt opp Kolla med Puppet og nesten-deployet.
- Deployet Kolla uten Puppet men med ekstern database. slik et realistisk scenario med database vil være i forhold til IIK.

Hva er neste:

- Arbeide videre med rapporten, hovedsakelig utfylling chapter 2.
- Arbeide videre med Kolla, trenger forsatt gå litt mer i dybden på konfigurering i henhold til oppdragsgiveren.
- Deploye keystone i SkyLow med eller uten Puppet.

Utfordringer vi har hatt:

- Få ekstern database til å virke med Kolla.
- RabbitMQ konteiner som nekter å starte.
- Forskjellige DNS problemer som har ført til mye feilsøking.

Spørsmål:

- Hvordan vurderer vi egenbygde konteinere? Egenbygde via Kolla? Egenbygde via Docker, styrt av Puppet? Da må man enten hente lage helt egne images eller hente samme images vi allerede har brukt. OpenStack Helm?
- Hvordan gjøre en best mulig vurdering av teknologiene? Noen nøkkelpunkter å vurdere?
- Hva er scopet vårt for vurdering av konteinerløsning? Skal vi inn i dybden på kubernetes, Helm?
- Hvis vi mener Kolla ikke er 100% egnet, kan vi se på andre alternativer til kolla isteden for å bygge selv, dette mtp. scope og tidsbegrensninger, sånn som helm?
- Hvis vi annser Kolla som optimal løsning, skal vi i det hele tatt vurdere andre løsninger?

Selve møtet:

- Spørsmål1: Les rundt på forumer, hva gjør andre? hvem stiller spørsmålene? kanskje sende en hendvendelse via twitter til Jan Ivar Beddari. profesjonell openstacker, implementerte uninett openstack. Si at spørsmålet kommer fra hjelmås. hva er en god løsning på implementeasjon av konteinere i teknologi?

- Spørsmål 2: Noter kvalitativt, kolla vs helm, skriv ned foreløpige plusser og minuser. Finner man ingen konkrete svakheter og styrker mellom dem, skriv kvalitativt om dem, så det på en måte kan bli målt opp. Iso25010 handler litt om sammenligning av

teknologier og kan være nyttig for å ta i bruk når dere skal sammenligner kolla med andre løsninger.

- Helm er en sentral fil(??) til å beskrive kubernetes cluster

- Viktig å hente inn andres erfaringer og best practices.

Mandagsmøte 04.03.19 - 12:00

Hva vi har gjort:
- Så godt som ferdigstilt kapittel 1, arbeidet videre med kapittel 2, 3 og 4.
- Deployet kolla keystone koneinere ved bruk av puppet installasjon (intern database)
- Deployet en pakke av tjenster ved bruk av Helm (all-in-one)

Hva er neste:
- Deploye kolla keystone ved bruk av puppet og ekstern database eller integrere det rett i skylow om det viser
- Deploye multinode helm

Utfordringer vi har hatt:
- Få ekstern database til å virke med Kolla.
- RabbitMQ konteiner som nekter å starte.

Spørsmål:
- Hva verdsetter oppdragsgiver mest når det kommer til vår oppgave? konfigurasjonsstyring? Hastighet? Kompleksitet?

Selve møtet:
- Spørsmål 1: hva er det dere anser som praktisk i bruk

Mandagsmøte 11.03.19 - 12:00

Hva vi har gjort:
- Satt oss inn i OpenStack Helm og prøvd å fått deployet både multinode og AllInOne.
- “Suksessfullt” deployet AllInOne med unntak av akkurat Keystone for konteinerene vil ikke kjøre.
- Skrevet en god del på Teori og vurderings delen av rapporten.

Hva er neste:
- Arbeide litt videre med OpenStack Helm men skifte litt mer fokus over på selvbygde konteinere.
- Vurdere om selvbygde konteinere er et alternativ og hvor komplisert det er å sette opp og hvor effektivt det er å drifte.

- Begynne å sette opp Kolla i serverene i SkyLow.

Utfordringer vi har hatt:
- OpenStack Helm har vært veldig komplisert å sette seg inn i. Det er ganske rett frem og sette opp ettersom det er noen step-by-step guider fra folk eller OpenStack selv, men når disse ikke virker er det veldig utfordrende og feilsøke. Det er masse forskjellige komponenter som vi har forsøkt og sette oss inn i, og Kubernetes i seg selv er et veldig kompleks system som krever ekspertise som vi kanskje ikke har for å forstå teknologien på en god måte.

Spørsmål:
- Er strukturen på rapporten vår grei?
- Hva tenker Hjelmås om vurderingspunktene vi har valgt? Litt for generelle kanskje?
- Hvor mye i praktisk må settes opp for å gjøre en helhetlig vurdering?

Selve møtet:

- Prøv å kjør noen enkle tester med når Kubernetes etter at clusteret er satt opp for å få litt kontroll og se at ting faktisk virker og at ting skjer som det skal, referer gjerne til IMT3005 dokumentet hvor Hjelmås har beskrevet tester og sånn.

- Spørsmål 1: Virker greit, ikke krise om det må omstruktureres litt

- Spørsmål 2: Virker greit men vær presise med vurderingskriteriene, hva menes med sikkerhet? Hva er effektivitet? osv.

- Spørsmål 3: Beskriv hvordan på papiret hvordan det skal fungere, og argumenter teoretisk hva som er bra dårlig, og beskriv forsøket vi har gjort og hvor det gikk galt og hvordan vi systematisk feilsøkte og hvorfor det ikke lot seg gjøre.

Mandagsmøte 18.03.19 - 12:00

Hva vi har gjort:
- Systematisk feilsøking av openstack helm. forsøkt og komme videre uten hell
- Startet med forsøk av egenbygde konteinere, fått keystone konteinere til å kjøre og verifisert at det virker.

Hva er neste:
- Gi opp openstack helm og heller bare beskrive hvordan det kan bli tatt i bruk og hvorfor vi ikke klarte. Kompleksitet, mye og sette seg inn i, etc.
- Komme tilbake til kolla, prøve å få fungerende ekstern database eller prøve å bruke databasen i SkyLow
- Jobbe videre med egenbygde konteinere

Utfordringer vi har hatt:

- Veldig utfordrende å arbeide med openstack helm, ettersom vi ikke har særlig bakgrunnsteori og helm ligger oppå kubernetes som i seg selv er et veldig kompleks system.
- feilsøking i helm, ikke så alt for mye å finne på nettet, vanskelig å forstå feil og i det hele tatt vite hvordan man skal gjøre endringer.

Spørsmål:

- Hva gjør vi om vi ikke klarer å komme forbi de hindringene vi har nå?

- Til Eigil:
    - Er det farlig at ting blir endret i databasen i SkyLow?
    - Keystone db connection

Selve møtet:

- Sjekk hvordan Eigil og Lars-Erik har satt opp keystone og lag dockerfile ut av det.
- Vær kritisk til måten de har satt det opp, kan det gjøres bedre? Ta inspirasjon fra hvordan andre gjør det.

Mandagsmøte 25.03.19 - 12:00

Hva vi har gjort:

- Lagt Helm på is, prøvd så langt det lot seg gjøre, deployet både AllInOne og Multinode, gjort systematisk feilsøking og rapportert forsøkene. Ikke allverdens med hjelp å få.
- Hvorfor ikke bruke helm:
    - Trenger bredere kompetanse, stor kompleksitet. Egner seg kanskje ikke til Eigil og Lars Erik da
    - Fikk kun til å sette opp AllInOne, noe som ikke oppdragsgiver vil få verdi ifra
    - Multinode oppsettet var fylt med feilmeldinger men ikke mye svar. for det meste konteinere som satt fast. Delay second, mer ressurser etc.
- Fått opp keystone og horizon i egenbygde konteinere.

Hva er neste:

- Jobbe med kolla igjen. Finne ut hvordan man integrerer det med SkyLow.
- Jobbe videre med egenbygde konteinere.
- Rapportere videre

Utfordringer vi har hatt:

- Helm.

Spørsmål:

- Kan vi gjøre endringer på databasen i SkyLow uten at det crasher noe?

Selve møtet:

- Fikk presentert hvor langt vi hadde kommet på våre 3 vurderte løsninger. Oppdragsgiver virker interessert i de forskjellige løsningen og virket enige i våre vurderingskriterium. Må huske å definere hva f.eks effektivitet betyr.
- Fikk fortalt at Helm ikke lenge var med pga. kompleksitet.

Mandagsmøte 01.04.19 - 16:00

Hva vi har gjort:

- Fått opp igjen fungerende kolla hvor tjenesten keystone kjører i konteiner og kommuniserer med ekstern database, testet og verifisert at man kan legge inn brukere.
- Kjørt opp egenbygde konteinere og fått det til å kjøre keystone.

Hva er neste:

- Forsøke å integrere noe i SkyLow (som vanlig)
- For alvor begynne å fokusere mer på rapporten og gjøre den klar til innlevering av første utkast 12.04.19.

Utfordringer vi har hatt:

- Brukte litt for mye tid på å sette opp puppet agent og master.

Spørsmål:

-

Selve møtet:

- Ingen møte med veileder.

Mandagsmøte 08.04.19 - 12:00

Hva vi har gjort:

- Skrevet videre på rapporten.

Hva er neste:

- Prøve å få rapporten opp til 30 sider før innlevering av 1. utkast innen 12.04

Utfordringer vi har hatt:

- Sliter med å fylle ut sider, og generelt skrive rapporten på en så god måte som mulig

Spørsmål:

- Hvor mange sider er forventet av en gruppe på 3 i vårt type scenario?
- Vet du om noen spesifikke kriterier for hva som skiller fra f.eks B fra en C?

-

Selve møtet:

- Snakk litt med Mats Oves gruppe om sammenheng mellom oppgaver vi har gjort. Vi har foreslått endringer til arkitektur mens de driver med testing av nåværende arkitektur.
- Presentasjon kan telle positivt, blir ikke vektlagt på nervøsitet eller presentasjonsteknikk og lignende.
- Ved rapporten må det komme tydelig frem hva vi har gjort, tydelig presentert alternativene vi har, har figurer som viser oppsett, kan fint tegne use-caser, viser underveis at vi gjør vurderinger, skriver objektivt og godt. Fin balanse med tekst og figurer, det må komme tydelig frem at vi har god teknisk forståelse. Kommer det tydelig frem at vi har stor arbeidsmengde, at vi har fullført det vi skal og litt til + en velskrevet rapport er vi fort på en B.
- Vedlegg går og på rapport kvalitet og vil telle på karakter men ikke være sentralt.
- Skriv så mye vi kan på rapporten før påska.
- Kan bruke kombinasjon av metrikker og beskrivelse, som regel er ren kvalitativ (tekst) det beste.
- Cirka 50-80 sider med ren rapport. Vertfall 50 sider med ren rapport.

Mandagsmøte 29.04.19 - 12:00

Hva vi har gjort:

- Jobbet med rapporten

Hva er neste:

- Jobbe med rapporten, Skrive mer på kap 6, 7 og begynne på 9

Utfordringer vi har hatt:

-

Spørsmål:

- Noen tips til hvordan skal vi gjøre kravspek?
- Vector grafikk? f.eks på 6.2.2 skal vi lage vår egen verjson av figuren i vector grafikk også hendvise til den faktiske figuren?
- 5.2.2 er hentet direkte fra video, må vi fortsatt fjerne da, eller må vi quote f.eks?
- Er det noe du føler vi mangler eller kan utfylle mer på?

Selve møtet:

- Det burde innledningsvis stå hvilken krav og kriterier som er viktig for kontenerisering av tjenester. Beskrive hvorfor vi har valgt de kriteriene vi har valgt. Hva er kravene fra Eigil og Lars Erik. Et type kravspek kapittel som beskriver rammene. Trenger ikke flytte noe, bare skrive litt anderledes hvis vi skal inkludere det. Kan bruke use-case

ved oppgradering f.eks. Definerer use-cases på kravspek delen. Høre med Lars erik og eigil om type use-cases.
- Alle egnelagde figurer skal være vector grafikk, figurene hentet fra nett kan ikke gjøres noem med.
- Skummelt å nevne open-source til bruk av egne formål pga lisenser. Kan verdt å sjekke hvilken lisens Helm og ansible og OpenStack er. OpenStacks open-source lisens er apachelisens. Hvis et prosjekt er open-source hva har de som forker lov til å gjøre.
- Mtp. på nedkorting, det er litt unødvendig langt, men la det stå foreløpig å se ann hvor lang rapporten blir. Alle setninger skal hovedsakelig være på plass for å gi prosjektet verdi. Burde gå utifra å skrive kun om hva som er direkte viktig for prosjektet.
- hovedinndelingen kan egentlig ligge som kun en paragraf i rapportens innhold.
- Ha en god blandnig av kvantitativ og kvalitativ vurdering.
- Prøv å kom med kun EN anbefaling men i dialog med oppdragsgiver. Konklusjonen burde ha et balansert syn, det er viktig å ha fler synspunkt.
- Kan bruke enkelt personer (anonyme kilder) som kilder og kan hendvise til det.

- Dokumentasjonskvalitet KAN AV OG TIL være VELDIG VIKTIG!

Mandagsmøte 06.05.19 - 12:00

Hva vi har gjort:
- Skrevet på rapport. Fylt ut alt vi mangler utenom avslutningskapittel. Laget liste over gjøremål for innlevering.

Hva er neste:
- Finpusse, skrive om, ferdigstille rapport.

Utfordringer vi har hatt:
-

Spørsmål:
- Hva på lista over gjøremål bør prioriteres å bli ferdig med før vi leverer 2. utkast?
-

Selve møtet:
- Fiks det som skal skrives, husk at alle figurer skal ha figurtekst og referes til i teksten
- Konklusjon-kap ca 3 sider - skal også ha med drøfting om løsningen, men det kan også ha blitt gjort på slutten av forrige kapittel.

Møte med pipeline gutta 13.05.19 kl 11:00

Lagd tester via git commit

bruker puppet, beeker, her gikk det fort i svingene gitt.

pipeline gutta sier det hadde vært lettere om tjenestene hadde kjørt i konteiner spesielt kubernetes.

Mandagsmøte 13.05.19 - 14:00

Hva vi har gjort:
- Fått tilbakemelding på 2. utkast av rapporten
- Jobbet videre med korrekturlesing og retting av feil

Hva er neste:
- Gå gjennom tilbakemeldinger fra veileder
- Levere rapporten

Utfordringer vi har hatt:
- Slitt litt med siteringsregler og best-practices.
- Vanskelig å vite hva av rapporten som er av svakest kvalitet, hva vi burde bruke tid og energi på å forbedre.

Spørsmål:
- Skulle vi skrive om Puppet og Ansible?
- Skal vi kutte noe mer på teori?
- Hva måtte gjøres for å fikse siteringene våre?
- Hva kan vi gjøre for å fikse figur/tabell-mangel fra side 40 til 54?

Selve møtet:
- Ikke nødvendig
- Er litt unødvendig informasjon men rapporten er ikke så lang at det gjør så veldig mye
- Legge inn år og besøkt dato
- Kanskje finne noen figurer om trender og popularitet på de forskjellige vurderingene.

E Timelister

Timelister

	Total tid	Hamse	Morten	Fredrik	Sammenlagt
	Timer	512:25:00	520:40:00	500:00:00	1533:05:00

Uke 2	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse					3:00:00			3:00:00
Morten								0:00:00
Fredrik		1:00:00	2:00:00		1:00:00			4:00:00
Total	0:00:00	1:00:00	2:00:00	0:00:00	4:00:00	0:00:00	0:00:00	7:00:00

Uke 3	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	5:45:00	8:00:00			7:00:00			20:45:00
Morten	5:45:00	7:30:00	7:55:00	8:20:00	7:00:00			36:30:00
Fredrik	5:45:00	3:30:00	8:30:00	7:30:00	7:00:00			32:15:00
Total	17:15:00	19:00:00	16:25:00	15:50:00	21:00:00	0:00:00	0:00:00	89:30:00

Uke 4	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00	0:30:00	7:15:00	8:00:00	8:15:00			32:00:00
Morten	8:00:00	0:30:00	8:00:00	8:00:00	8:15:00			32:45:00
Fredrik	7:45:00	0:30:00	8:00:00	8:00:00	8:15:00			32:30:00
Total	23:45:00	1:30:00	23:15:00	24:00:00	24:45:00	0:00:00	0:00:00	97:15:00

Uke 5	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:00:00	8:00:00	6:00:00			30:00:00
Morten	8:00:00		8:00:00	8:00:00	7:00:00			31:00:00
Fredrik	8:00:00		4:00:00	6:00:00	3:00:00			21:00:00
Total	24:00:00	0:00:00	20:00:00	22:00:00	16:00:00	0:00:00	0:00:00	82:00:00

Uke 6	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:30:00		8:00:00	8:45:00	9:00:00			34:15:00
Morten	8:30:00		8:00:00	8:00:00	9:00:00			33:30:00
Fredrik	8:30:00		8:00:00	8:00:00	7:00:00		1:30:00	33:00:00
Total	25:30:00	0:00:00	24:00:00	24:45:00	25:00:00	0:00:00	1:30:00	100:45:00

Uke 7	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:40:00		8:00:00	6:00:00	8:00:00			30:40:00
Morten	8:40:00		9:30:00	7:00:00	7:45:00			32:55:00
Fredrik	8:00:00		8:00:00	7:30:00	7:45:00			31:15:00
Total	25:20:00	0:00:00	25:30:00	20:30:00	23:30:00	0:00:00	0:00:00	94:50:00

Uke 8	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:30:00	8:00:00	8:00:00			32:30:00
Morten	8:00:00		8:00:00	8:00:00	8:00:00			32:00:00
Fredrik	9:00:00		6:00:00	5:00:00	1:00:00			21:00:00
Total	25:00:00	0:00:00	22:30:00	21:00:00	17:00:00	0:00:00	0:00:00	85:30:00

Uke 9	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	7:00:00		7:00:00	8:00:00	11:00:00			33:00:00
Morten	7:30:00		7:00:00	8:00:00				22:30:00
Fredrik			7:00:00	8:00:00	8:00:00			23:00:00
Total	14:30:00	0:00:00	21:00:00	24:00:00	19:00:00	0:00:00	0:00:00	78:30:00

Uke 10	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:00:00	8:00:00	6:00:00			30:00:00
Morten	8:00:00		8:00:00	8:00:00	6:00:00			30:00:00
Fredrik	6:00:00		7:00:00	8:00:00	6:00:00			27:00:00
Total	22:00:00	0:00:00	23:00:00	24:00:00	18:00:00	0:00:00	0:00:00	87:00:00

Uke 11	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		6:00:00	8:00:00	8:00:00	4:00:00		34:00:00
Morten	8:00:00		8:00:00	8:00:00	8:00:00			32:00:00
Fredrik			3:00:00	8:00:00	7:00:00			18:00:00
Total	16:00:00	0:00:00	17:00:00	24:00:00	23:00:00	4:00:00	0:00:00	84:00:00

Uke 12	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:00:00	8:00:00	7:00:00			31:00:00
Morten	5:00:00		8:00:00	8:00:00	8:00:00			29:00:00
Fredrik	8:00:00		8:00:00	8:00:00	8:00:00			32:00:00
Total	21:00:00	0:00:00	24:00:00	24:00:00	23:00:00	0:00:00	0:00:00	92:00:00

Uke 13	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	9:00:00		8:00:00	9:00:00				26:00:00
Morten	8:00:00		8:00:00	8:00:00	6:00:00			30:00:00
Fredrik	9:00:00	2:00:00	4:00:00		6:00:00		1:00:00	22:00:00
Total	26:00:00	2:00:00	20:00:00	17:00:00	12:00:00	0:00:00	1:00:00	78:00:00

Uke 14	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:00:00	4:00:00	7:00:00			27:00:00
Morten	8:00:00		4:00:00	Hyttetur	Hyttetur			12:00:00
Fredrik	8:00:00		8:00:00	1:00:00	7:00:00			24:00:00
Total	24:00:00	0:00:00	20:00:00	5:00:00	14:00:00	0:00:00	0:00:00	63:00:00

Uke 15	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:00:00	6:00:00	7:30:00			29:30:00
Morten	9:00:00		9:00:00	8:00:00	7:30:00			33:30:00
Fredrik	8:00:00	1:00:00	8:00:00	6:00:00	7:30:00			30:30:00
Total	25:00:00	1:00:00	25:00:00	20:00:00	22:30:00	0:00:00	0:00:00	93:30:00

Uke 16	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	Påske	Påske	4:00:00	Påske	Påske			4:00:00
Morten	4:00:00	Påske	4:00:00	Påske	Påske			8:00:00
Fredrik	2:00:00	2:00:00	Påske	Påske	Påske	2:00:00	1:00:00	7:00:00
Total	6:00:00	2:00:00	8:00:00	0:00:00	0:00:00	2:00:00	1:00:00	19:00:00

Uke 17	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	Påske		6:00:00	8:00:00	7:00:00			21:00:00
Morten	6:00:00		8:00:00	8:00:00	7:30:00			29:30:00
Fredrik	4:00:00	1:00:00	8:00:00	9:00:00	7:30:00	01:00.00		29:30:00
Total	10:00:00	1:00:00	22:00:00	25:00:00	22:00:00	0:00:00	0:00:00	80:00:00

Uke 18	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	7:00:00		8:00:00	8:00:00	6:30:00			29:30:00
Morten	8:00:00		8:00:00	8:00:00	6:30:00			30:30:00
Fredrik	10:00:00		7:30:00	8:00:00	7:30:00	2:00:00		35:00:00
Total	25:00:00	0:00:00	23:30:00	24:00:00	20:30:00	2:00:00	0:00:00	95:00:00

Uke 19	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	8:00:00		8:15:00	9:30:00	14:00:00			39:45:00
Morten	4:00:00		8:30:00	10:00:00	14:00:00			36:30:00
Fredrik	8:00:00		10:00:00	10:00:00	14:00:00			42:00:00
Total	20:00:00	0:00:00	26:45:00	29:30:00	42:00:00	0:00:00	0:00:00	118:15:00

Uke 20	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Total
Hamse	7:00:00		7:00:00	3:30:00		4:00:00	3:00:00	24:30:00
Morten	8:00:00		9:00:00	6:30:00		1:00:00	4:00:00	28:30:00
Fredrik	5:00:00	1:30:00	10:00:00	8:00:00	1:00:00	5:00:00	4:30:00	35:00:00
Total	20:00:00	1:30:00	26:00:00	18:00:00	1:00:00	10:00:00	11:30:00	88:00:00

F Opprinnelig oppgavebeskrivelse

Container-tjenester for OpenStack

Eigil Obrestad & Lars Erik Pedersen

October 22, 2018

1 Bakgrunn

NTNU benytter openstack som en privat skyløsning. Openstack gir studenter og ansatte en plattform hvor en kan lage virtuelle maskiner og nettverk. Plattformen benyttes av studenter som skal lære seg systemadministrasjon, forskere som har behov for å kjøre simuleringer, samt mye annet. Det er IIK som er ansvarlig for arkitekturen bak de fleste openstack skyene på NTNU.

1.1 Arkitektur

Selve openstack prosjektet består av en rekke tjenester. Disse tjenestene kan for eksempel være en autentiseringskomponent (keystone) eller et web-basert brukergrensesnitt (horizon). En openstack tjeneste er i praksis et web-grensesnitt, et REST-API, eller en tjenerprosess som kommuniserer med andre komponenter. Hver tjeneste kjører i dag som 2 eller flere virtuelle maskiner bak en lastbalanserer.

Hver slik virtuell maskin gjør i all hovedsak kun en tjeneste. Dette gjør at det er ganske mange virtuelle maskiner i plattformen, som alle har et operativsystem, krever overvåking, oppdateringer og lignende.

2 Oppgaven

Oppgaven skal ta for seg alternative arkitekturer til hva som finnes i dag. Hovedmålet må være å finne en arkitektur som er mer effektiv å drifte, både i det daglige, men kanskje spesielt i forbindelse med oppgraderinger. En ide til en slik arkitektur er å se på muligheten til å lage containere som er ansvarlige for de ulike tjenestene, og det er det denne oppgaven skal handle om. Oppgaven vil da bestå av to deler.

2.1 Containere for openstack

Skal man kjøre openstack tjenester i containere må man lage disse containerene, og i denne forbindelse finnes det minst to alternativer:

- **Openstack Kolla**: Openstack har allerede et prosjekt som omhandler tjenester i containere. Oppgaven bør vurdere om dette prosjektet er hensiktsmessig for skyhighs arkitektur, og med det se på hvilken arbeidsflyt som kreves for å kjøre Kolla, samt hvordan oppgraderinger bør foregå.

- **Bygge containere selv**: Om det viser seg at Kolla er uegnet er det et alternativ å lage rutiner på å bygge containere selv. Da må man se på hvordan arbeidsflyt det så kreves

Om det finnes andre alternativer som er aktuelle er det mulig å utforske disse og.

2.2 Container infrastruktur

Etter å ha bygget containere vil man ha behov for en platform å kjøre disse på. Da det i skyhigh ikke er benyttet containere før er det behov for å bygge en passende plattform for å kjøre containerene på. Denne delen av oppgaven vil innebære å idenitifisere passende verktøy for å kjøre kontainere, samt finne gode verktøy for administrasjon og vedlikehold.

Alle maskiner i skyhigh i dag er administrert av puppet, så en eventuell løsning kan med fordel implementeres i puppet for automatisk installasjon.

G Prosjektplan

Prosjektplan for bacheloroppgave 2019

Morten Bjerke, Fredrik Røstad, Hamse Hashi

January 2019

1

Contents

1 Mål og rammer **2**
- 1.1 Bakgrunn . . . 2
- 1.2 Prosjektmål . . . 2
  - 1.2.1 Resultatmål . . . 2
  - 1.2.2 Effektmål . . . 2
  - 1.2.3 Læringsmål . . . 2
- 1.3 Rammer . . . 3
  - 1.3.1 Tidsmessige rammer . . . 3
  - 1.3.2 Økonomiske rammer . . . 3
  - 1.3.3 Kravspesifikke rammer . . . 3

2 Omfang **3**
- 2.1 Fagområde . . . 3
- 2.2 Avgrensning . . . 4
- 2.3 Oppgavebeskrivelse / problemstilling . . . 4
  - 2.3.1 Konteinere for OpenStack . . . 4
  - 2.3.2 Konteiner-infrastruktur . . . 5

3 Prosjektorganisering **5**
- 3.1 Ansvarsforhold og roller . . . 5
  - 3.1.1 Prosjektgruppemedlemmer . . . 5
  - 3.1.2 Produkteiere . . . 5
  - 3.1.3 Veileder . . . 5
- 3.2 Grupperegler og rutiner . . . 6
  - 3.2.1 Grupperegler . . . 6
  - 3.2.2 Arbeidsrutiner . . . 7
- 3.3 Verktøy . . . 8

4 Planlegging, oppfølgning og rapportering **8**
- 4.1 Hovedinndeling av prosjektet . . . 8
  - 4.1.1 Prosjektplan . . . 8
  - 4.1.2 Vurdering av best egnet teknologier . . . 8
  - 4.1.3 Implementering og testing . . . 9
  - 4.1.4 Ferdigstille rapport . . . 9
- 4.2 Valg av SU-modell/prosessrammeverk med argumentasjon . . . 9
- 4.3 Valg av metode og tilnærming (avklare Teori- og Metodebruk . . 9
- 4.4 Plan for gjennomføring . . . 10

5 Organisering av kvalitetssikring **11**
- 5.1 Dokumentasjon, standardbruk, og kildekode . . . 11
- 5.2 Risikoanalyse . . . 11
  - 5.2.1 Risikoer tatt i betraktning . . . 12
  - 5.2.2 Risikomatrise . . . 13
  - 5.2.3 Tiltak til risikoer . . . 13

5.2.4 Risikomatrise etter tiltak 15

References **15**

1 Mål og rammer

1.1 Bakgrunn

I de senere årene har skyløsninger blitt meget populært innen IT-bransjen. Private skyløsninger er skyer satt opp internt for en organisasjon. OpenStack er en av de store aktørene innen private skyløsninger. Ved NTNU er det satt opp en implementasjon av OpenStack kalt SkyHiGh. Tjenesten brukes av både studenter og lærere ved NTNU som har behov for å arbeide med virtuelle maskiner, sette opp egne infrastrukturer, eller sette opp test-miljøer og lignende type formål. SkyHiGh driftes av NTNUs institutt for informasjonssikkerhet og kommunikasjonsteknologi, av Eigil Obrestad og Lars Erik Pedersen. Vårt prosjekt går ut på å gjøre en akademisk utredning av konteiner-tjenester på vegne av NTNU IIK ved NTNU Gjøvik.

1.2 Prosjektmål

Målene med dette prosjektet er følgende:

1.2.1 Resultatmål

Vurdere om OpenStack Kolla eller selvbygde konteinere er hensiktsmessig å bruke for SkyHiGh, implementere den ønskede løsningen i testmiljøet SkyLow, og dokumentere rutiner og arbeidsflyt for den valgte løsningen. I den valgte løsningen skal en teknisk anbefaling supplementeres med en praktisk løsning hvor OpenStack komponenten Keystone er plassert i konteinere, med dokumentasjon for driftsrutiner vedlagt.

1.2.2 Effektmål

Målet er at kompleksitet og tidsbruk ved oppgraderinger eller endringer i OpenStack skal reduseres. For øyeblikket er det en tidkrevende og kompleks prosess som hovedsaklig improviseres.

- Løsningen skal være enkel og effektiv å drifte i det daglige og ved oppdateringer.
- Løsningen skal redusere tidsforbruk for driftsrutiner.
- Løsningen skal ta i betraktning for det som blir ansett som beste sikkerhets praksis i forhold til konteinere.
- Det er ønskelig om løsningen reduserer kodebasen til SkyHiGh.

1.2.3 Læringsmål

Ved slutten av prosjektet håper vi å ha lært om hvordan det er å jobbe på et ordentlig driftprosjekt. Vi ønsker å teste ut nye rammeverk og verktøy, og å lære

bedre å arbeide i team. Vi ønsker og å få bedre forståelse for den fremtidsrettede teknologien konteinere og konteiner-orkestrering, da dette er relevant teknologi vi håper å kunne ta med oss videre inn i arbeidslivet. I tillegg ønsker vi å lære å jobbe med den smidige utviklingsmodellen Scrum i et realistisk scenario.

1.3 Rammer

Dette er rammene satt for prosjektarbeidet.

1.3.1 Tidsmessige rammer

Gruppen er nødt til å holde seg innenfor gitt tidsfrist av universitet, det vil si en total arbeidstid på rundt fire og en halv måned. Forventet arbeidsinnsats er fra 1500 til 1800 timer fordelt på de 3 gruppemedlemmene. Rapportskrivingen skal arbeides med parallelt med del én av oppgaven; vurdering av teknologier, samt del to; implementering av eventuell løsning. Det vil da ikke gi gruppen full frihet til arbeid med selve løsningen på fulltid. Den praktiske delen av oppgaven skal være ferdig innen påskeferien, 12. april, da resten av tiden skal brukes på rapportskriving. Rapporten skal leveres inn 20. mai.

1.3.2 Økonomiske rammer

Slik vi vurderer så trenger ikke prosjektet noe økonomisk budsjett da alle verktøyene og ressursene vi tar i bruk er enten lisensiert via universitetet eller kostnadsfrie. Om det det blir behov for å kjøpe noe eller om det dukker opp noen utgifter vil gruppen ta ansvar for å finansiere det selv så lenge det er innenfor rimelig grense.

1.3.3 Kravspesifikke rammer

Vi skal også følge en teknologisk ramme ved at arbeidsgiver hovedsakelig ser etter effektivisering av nåværende tjeneste ved bruk av konteiner-teknologi, og ønsker derfor om ønskelig å fortsette bruk av noen utvalgte og spesifikke teknologier, hovedsaklig Puppet. Løsningen skal kun utredes for den private skyløsningen OpenStack, spesifikt NTNU sitt eget prosjekt SkyHiGh. Etter samtale med vår oppdragsgiver har vi kommet frem til at vi først og fremst skal levere tjenesten Keystone i konteinere på en god måte, heller enn å fokusere på å få alle tjenestene inn i konteinere. Hvis tiden tillater det kan vi vurdere å sette inn flere tjenester i konteinere, men dette er ikke prioritert.

2 Omfang

2.1 Fagområde

Sky-løsninger blir stadig mer populært blant bedrifter. Flere og flere tjenester, komponenter, servere osv. blir digitalisert, og et hav av bedrifter er allerede

godt inne i denne mer digitaliserte kulturen. I denne oppgaven skal det arbeides med den private skyløsningen OpenStack. OpenStack er en plattform for å lage infrastrukturen til private eller offentlige skyer(*OpenStack*, 2019). NTNU benytter seg av OpenStack og har sin egen sky i OpenStack med navnet SkyHiGh(*Openstack documentation*, 2018). Denne brukes av IT-studenter og ansatte for faglige gjøremål. SkyHiGh driftes for tiden av to ansatte og hele prosjektet har totalt åtte bidragsytere(*NTNUSky*, 2019). Tjenestene som blir levert er kjørt i virtuelle maskiner men teknologien oppgaven omhandler vil være rundt konteinere og orkestrering av de. Vi kommer naturligvis til å ta i bruk Docker for denne oppgaven. Docker er den største innen konteiner-teknologi og har liten konkurranse i markedet. Det har også i de senere årene vokst frem bruk av Kubernetes sammen med OpenStack. Kubernetes (*What is Kubernetes?*, 2018) er et rammeverk for å orkestrere konteinere. OpenStack har allerede et prosjekt der de benytter seg av Docker-konteinere for å levere tjenester mer effektivt og med redusert ressursbruk.

2.2 Avgrensning

Mange bedrifter benytter seg av OpenStack men problemet som skal løses i denne oppgaven vil være spesifikt knyttet til NTNU sin egen sky, SkyHiGh. Prosjektets mål er å finne best mulig løsning for at tjenester i SkyHiGh kan blir mer effektive og bruke mindre ressurser ved at tjenestene blir kjørt i konteinere i stedet for virtuelle maskiner. Gruppen vil kun gjøre vurdering av mest relevante teknologier og det vil gjøres implementasjon ut fra gruppens beslutning. Sluttproduktet skal være et testeksemplar som kan testes i SkyLow. Dette skal fungere som et proof of concept.

2.3 Oppgavebeskrivelse / problemstilling

Oppgaven skal ta for seg alternative arkitekturer til hva som finnes i dag. Hovedmålet er å finne en arkitektur som er mer effektiv å drifte i det daglige, men også kanskje spesielt i forbindelse med oppgraderinger. En idé til en slik arkitektur er å se på muligheten til å lage konteinere som er ansvarlige for de ulike tjenestene, og det er det denne oppgaven skal handle om. Vår problemstilling er derfor: *Hva er den mest effektive og hensiktsfulle metoden for å kjøre NTNUs SkyHiGh-tjenester i konteinere?*

2.3.1 Konteinere for OpenStack

Oppgaven vil da bestå av to deler. Skal man kjøre OpenStack tjenester i konteinere må man lage disse konteinerne, og i denne forbindelse finnes det minst to alternativer:

- OpenStack Kolla: OpenStack har allerede et prosjekt som omhandler tjenester i konteinere. Oppgaven bør vurdere om dette prosjektet er hensiktsmessig for SkyHiGh sin arkitektur, og med det se på hvilken arbeidsflyt som kreves for å kjøre Kolla, samt hvordan oppgraderinger bør foregå.

- Bygge konteinere selv: Om det viser seg at Kolla er uegnet er det et alternativ å lage rutiner på å bygge konteinere selv. Da må man se på hvordan arbeidsflyt det så kreves.

Om det finnes andre alternativer som er aktuelle er det mulig å utforske disse og.

2.3.2 Konteiner-infrastruktur

Etter å ha bygget konteinere vil man ha behov for en platform å kjøre disse på. Da det i SkyHiGh ikke er benyttet konteinere fra før av, er det behov for å bygge en passende plattform for å kjøre konteinere på. Ideen om å kjøre OpenStack sine skytjenester i konteinere er relativt nytt, derfor skal vi gjøre en vurdering på hvordan andre har implementert lignende løsning. Vi har sett at blant annet CERN(Bell, 2017), og RedHat(Jameson, 2015) har implementert OpenStacks skytjenester som konteinere. Vi skal også se om Mirantis har gjort noe tilsvarende. Det vil gjøres grundige undersøkelser av hvordan disse gigantene bruker teknologien. Hovedsakelig vil denne delen av oppgaven innebære å identifisere passende verktøy for å kjøre konteinere, samt finne gode verktøy for administrasjon og vedlikehold. Alle maskiner i SkyHiGh er i dag administrert av Puppet, så en eventuell løsning kan med fordel implementeres i Puppet for automatisk installasjon.

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

3.1.1 Prosjektgruppemedlemmer

- Morten Bjerke (Gruppeleder).
- Fredrik Røstad.
- Hamse Hashi (Scrum master).

3.1.2 Produkteiere

- Lars Erik Pedersen og Eigil Obrestad - Oppdragsgivere. Begge er ansatte ved NTNUs IIK og tidligere studenter i Gjøvik.

3.1.3 Veileder

- Erik Hjelmås - Førsteamanuensis og Programansvarlig for BITSEC ved NTNU Gjøvik.

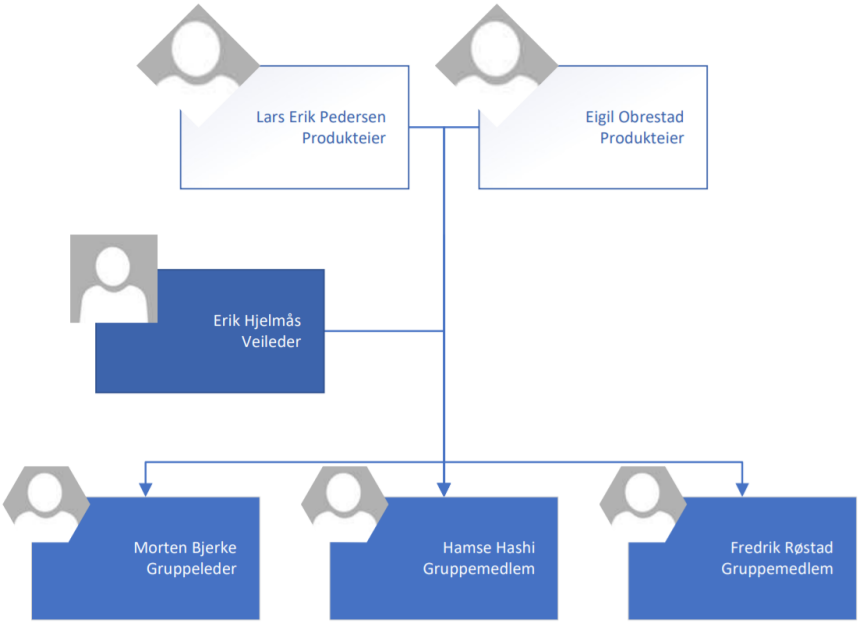


Figure 1: Organisasjonskart for prosjektgruppen

3.2 Grupperegler og rutiner

3.2.1 Grupperegler

- Det forventes en innsats på 30 timer pr uke.
- Møteplikt til faste møter.
- Aktiv deltagelse i gruppemøter.
- Møtereferat skrives i alle faste møter, sekretær roteres.
- Møtetider:
  - Mandag 09.00-17.00
  - Onsdag 09.00-17.00
  - Torsdag 09.00-17.00
  - Fredag 09.00-17.00
- Være tilgjengelig på valgt kommunikasjonsplattform, om dette ikke er mulig skal det forsøkes å gi beskjed til resten av gruppen.

- Om et gruppemedlem ikke kan møte til avtalt tid skal gruppen informeres senest dagen før.
- Ved eventuelle uenigheter bestemmer flertallet i gruppen.
- Sanksjon på 50 kroner for å være mer enn 15 minutter for sen. Pengene skal gå til en felleskasse som kan bli brukt av gruppen til felles aktiviteter eller lignende.
- Utgifter til prosjektet skal deles likt på alle gruppens medlemmer.
- Rutiner ved gjentatte brudd på regler:
  - Starter med første sanksjon og eskalerer ved behov.
  - Samtale mellom alle gruppedeltakerne om problemet.
  - Skriftlig advarsel med konkrete krav om hva som forventes endret eller forbedret.
  - Samtale mellom gruppen og veileder.
  - Ekskludering fra gruppen. Ingen ekskludering siste 30 dager av prosjektet.

3.2.2 Arbeidsrutiner

- Annenhver mandag skal vi ha "sprint review meeting" og "sprint retrospective event" der vi diskuterer hva som gikk bra og hva som kunne blitt gjort bedre i forhold til forrige sprint. I disse møtene vil vi se på hvordan vi ligger an i forhold til planlagt progresjon. Dette skal foregå før møtet med veileder.
- Annenhver mandag skal vi ha "sprint planning meeting" etter møtet med veileder og oppdragsgiver.
- Vi starter hver dag med et lite "daily scrum meeting" på 10-15 minutter der vi gjør klart hva som er dagens gjøremål. Vi skal også bestille rom under dette møtet.
- Vi skal ha et lite statusmøte hver mandag før møtet med veileder og eventuelt oppdragsgiver.
- I arbeidstiden skal det først og fremst arbeides med oppgaven, minimalt med personlige gjøremål.
- Ved behov vil også gruppa holde ekstra møter.

3.3 Verktøy

- Overleaf LaTeX- Prosjektplanen og prosjektrapporten skrives i Overleaf, en webbasert samarbeidsplattform for LaTeX.
- Trello - Sprint Planning Board er laget i Trello. Det er delt opp i backlog, sprint backlog, arbeides på, til vurdering, og ferdig.
- Google Drive (Docs, Sheets) - Møtereferater, notater, grupperegler, timelister, og andre administrative dokumenter skrives og lagres i Google Drive ved hjelp av Google Docs og Google Sheets. Prosjektarbeidene i Overleaf blir også kopiert over i Drive for backup.
- Microsoft Visio - Brukes til å lage grafikker som brukes i prosjektet.
- Microsoft Project - Gantt-diagram vil bli laget i MS Project.
- Git - Brukes til solid arbeidsflyt i infrastrukturkode, og sørger for versjonskontroll.

4 Planlegging, oppfølgning og rapportering

4.1 Hovedinndeling av prosjektet

Oppgaven vil bestå av to hovedoppgaver. Oppgaven en er å gjøre en utredning av konteiner-teknologier som kan være hensiktsmessig å benytte for OpenStack sine komponenter. Oppgave to vil i hovedsak være å lage et produkt som kan erstatte en av tjenestene SkyHiGh leverer med konteinere (for eksempel Keystone).

4.1.1 Prosjektplan

Starten av prosjektet. I starten av denne perioden vil gruppen etablere kontakt samt avtale faste ukentlige møter med veileder og arbeidsgiver. Prosjektplanen inneholder detaljerte beskrivelse av prosjektet, gruppens mål og rammer, planlagt tidsbruk, og annen relevant informasjon som vil kunne være til hjelp når hoveddelen av prosjektet begynner. Prosjektplanen skal leveres før avtalt tid som er 1. Februar 2019.

4.1.2 Vurdering av best egnet teknologier

Den første oppgaven etter prosjektplanen. Oppgavene i prosjektet vil jobbes med litt om hverandre, men det vil være et hovedfokus som føles naturlig for gruppen og vil gi oss en solid struktur. Denne delen av oppgaven vil være vital for videre arbeid av prosjektet. Vi vil starte med å gjøre research på alle teknologier vi mener kan være relevante. Vi ønsker å finne ut av hva som er best practice da det gjelder konteiner-tjenester i OpenStack, og om vi mener det er plausibelt å bruke dette i løsningen som skal utvikles. Etter dette skal

vi se spesifikt på om OpenStack Kolla eller bygging av egne konteinere er mest hensiktsmessig for oss. På oppdragsgivers anbefaling starter vi med vurderingen av OpenStack Kolla.

4.1.3 Implementering og testing

Siste fasen av arbeid på problemstillingen. Ved valgt teknologi vil det gjøres videre utredning av denne teknologien og gruppen vil forsøke å samle så mye relevant informasjon som mulig for å gi best resultat. I denne fasen skal det da gjøres dypdykk fra hele gruppen på den valgte teknologien og samtidig skal det jobbes litt parallelt med testing og implementasjon for å holde oss på rett kurs. Etter valgt teknologi vil det forsøkes på best mulig måte å implementere løsningen, det vil i hovedsak gå ut på å erstatte en allerede fungerende tjeneste med vår egen løsning. SkyHiGh leverer fler tjenester, men gruppens hovedfokus vil være å erstatte kun en tjeneste med vår egen løsning på en så solid måte som mulig. Løsningen vi implementerer skal rulles ut i SkyLow.

4.1.4 Ferdigstille rapport

I tiden etter at løsningen er ferdigstilt skal rapporten fullføres. Rapporten vil bli jobbet med parallelt i alle tidligere faser, men vil ikke være hovedfokus før løsningen er ansett som ferdig. I rapporten vil alt oppgaven omhandlet fra start til slutt bli dokumentert, sammen med gruppens relevante vurderinger.

4.2 Valg av SU-modell/prosessrammeverk med argumentasjon

Oppgaven vår er av en slik natur at arbeidsoppgavene kan endres ut fra hvordan utredningene våre går. Det vil si at en mer statisk utviklingsmodell som for eksempel fossefall som egner seg mer for tydelige mål ikke vil være passende. Vi mener derfor at det er logisk å gå for en smidig utviklingsmodell, og har bestemt oss for å bruke Scrum. Scrum som utviklingsmodell er bra i en situasjon som vår, ettersom modellen er veldig tydelig og definert, og den er i tillegg lett å ta i bruk. Det er noe som gjør at vi ikke bruker overdrevent mye tid rundt utviklingsmodellen og heller kan fokusere mer på selve oppgaven. Alle på gruppen er kjent med Scrum fra tidligere semestre, da vi har jobbet med forskjellige varianter av den i flere tidligere prosjekter. Modellen er også passende for oss med tanke på antall medlemmer i gruppen.

4.3 Valg av metode og tilnærming (avklare Teori- og Metodebruk

Siden prosjektperioden er forholdsvis kort og krever klar progresjon på deloppgaver jevnlig, så har vi to ukers sprint om gangen. Vi kommer til å operere med daily scrum møter på starten av hver dag, hvor vi tar 10 minutter til å gå over hvordan vi ligger an og hva dagens gjøremål skal være. Det vil også bli

tatt i bruk sprint review og sprint planning møter. Disse møtene vil ledes av en Scrum master. Vi har valgt Hamse som Scrum master, da han er den i gruppen med mest tidligere erfaring i Scrum. Sprint review og Sprint planning vil begge foregå på mandager, da vi har møte med veileder og produkteiere denne dagen. Vi vil føre opp deloppgaver i en product backlog. Oppgavene i 'Backlog' vil flyttes over til 'Sprint backlog' da de velges for neste sprint. Da en oppgave blir tildelt et eller flere gruppemedlemmer blir den flyttet til 'Arbeides på'. Oppgaven blir flyttet over 'Til vurdering' da den/de som jobbet på oppgaven er fornøyd med utkastet, og 'Ferdig' da den er vurdert som godkjent av enten ett gruppemedlem som ikke deltok på oppgaven, eller hvis alle gruppemedlemmene er enige på oppgaver der alle deltok.

4.4 Plan for gjennomføring

ID	Task Mode	Task Name	Duration	Start	Finish
1		**Prosjektplan**	**20 days**	**Mon 07.01.1**	**Fri 01.02.19**
2		Research og litteratursøk	20 days	Mon 07.01.19	Fri 01.02.19
3		Prosjektstart og avtale med veileder	5 days	Mon 07.01.19	Fri 11.01.19
4		Sprint 1: Arbeid med prosjektplan	15 days	Mon 14.01.19	Fri 01.02.19
5		Innlevering av prosjektplan	0 days	Fri 01.02.19	Fri 01.02.19
6		**Utredning**	**20 days**	**Mon 04.02.1**	**Fri 01.03.19**
7		Sprint 2: Vurdere og teste OpenStack	10 days	Mon 04.02.19	Fri 15.02.19
8		Sprint 3: Vurdere og teste selvbygd konteinerløsning	10 days	Mon 18.02.19	Fri 01.03.19
9		Beslutningspunkt for løsning	0 days	Fri 01.03.19	Fri 01.03.19
10		**Utvikling**	**30 days**	**Mon 04.03.1**	**Fri 12.04.19**
11		Sprint 4: Videre arbeid med valgt	10 days	Mon 04.03.19	Fri 15.03.19
12		Sprint 5: Arbeid med PoC	10 days	Mon 18.03.19	Fri 29.03.19
13		Sprint 6: Ferdigstilling av PoC	10 days	Mon 01.04.19	Fri 12.04.19
14		PoC på løsning ferdig	0 days	Fri 12.04.19	Fri 12.04.19
15		**Rapportskriving**	**76 days**	**Mon 04.02.1**	**Mon 20.05.1**
16		Arbeide med rapport ved siden av annet arbeid	50 days	Mon 04.02.19	Fri 12.04.19
17		Sprint 7: Skrive prosjektrapport	10 days	Mon 15.04.19	Fri 26.04.19
18		Sprint 8: Skrive prosjektrapport	10 days	Mon 29.04.19	Fri 10.05.19
19		Sprint 9: Finpussing før sluttinnlevering	6 days	Mon 13.05.19	Mon 20.05.19
20		Innlevering av oppgaven	0 days	Mon 20.05.19	Mon 20.05.19

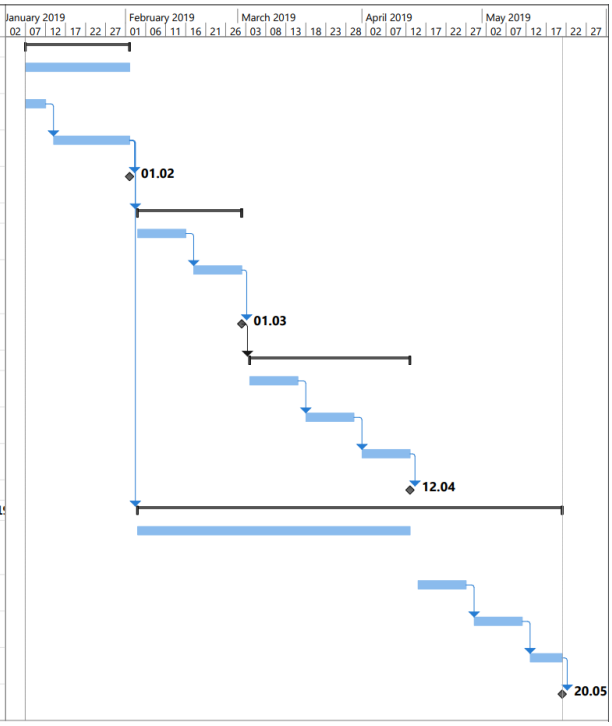


Figure 2: Fremdriftsplan

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk, og kildekode

Produkteierne har bedt om dokumentasjon på rutiner som behøves ved utrulling eller oppdatering av den valgte løsningen. Dette må derfor dokumenteres. Kildekode vil bli lagt på Github. Koden skal følge generelle best practices for kodestandard. Vi skal sikte mot selvdokumenterende kode. Dette anses som best practice i programmerbar infrastruktur alias "Infrastructure as Code"(Morris, 2016). Vi skal også følge security by design som et konsept(Wikipedia contributors, 2018). Kommentering av kode skal gjøres i eventuelle script etc., men deklarativ infrastrukturkode er selvdokumenterende og vil ikke bli videre dokumentert med mindre vi anser det som nødvendig i spesifikke tilfeller.

5.2 Risikoanalyse

Risikomatrisen viser konsekvens på x-aksen og sannsynlighet på y-aksen. Produktet av disse er risikoverdien ved en spesifikk risiko. Klassifisering av risikoverdiene finnes i risikoverdivurderingen under matrisen. Hver enkelt risiko er representert i matrisen med nummeret fra risikolisten.
Vi bruker en skala fra 1-4 på både sannsynlighet og konsekvens. Der 1 er lav sannsynlighet/ konsekvens, 2 under middels sannsynlighet/ nokså liten konsekvens, 3 over middels sannsynlighet/ ganske stor konsekvens og 4 meget sannsynlig/ stor konsekvens.

5.2.1 Risikoer tatt i betraktning

Teknologimessige risikoer				
Nr	Beskrivelse	Sannsynlighet	Konsekvens	Risikoverdi
1	Onlinetjenester (Overleaf, SkyHiGh/SkyLow, Trello osv) går ned og fører til at gruppen ikke får arbeidet.	2	2	4
2	Tap av dokument som fører til at gruppen mister arbeid gjort så langt.	2	4	8
3	Vi skaper nedetid i SkyLow når vi prøver å rulle ut løsninger i SkyLow	3	2	6
4	Vi får ikke de nødvendige ressursene og tilgangene i SkyHiGh/SkyLow for å implementere løsningen vår.	1	3	3

Forretningsmessige risikoer				
Nr	Beskrivelse	Sannsynlighet	Konsekvens	Risikoverdi
5	Dårlig kommunikasjon og samarbeid med oppdragsgiver.	2	4	8
6	Tap av sensitiv data fra oppdragsgivers systemer, slik som passord til VM o.l.	2	4	8
7	Arbeidet er ikke det produkteier ønsker	2	2	4

Projektgruppemessige risikoer				
Nr	Beskrivelse	Sannsynlighet	Konsekvens	Risikoverdi
8	Dårlig motivasjon i gruppen fører til redusert fremgang.	3	2	6
9	Sykdom fører til redusert fremgang.	3	2	6
10	Uenigheter innad i gruppen forsinker arbeidet.	2	2	4
11	Dårlig innsats av noen i gruppen fører til manglende progresjon.	2	2	4
12	Gruppemedlem ekskluderes fra eller forlater gruppen, som reduserer gruppens kapasitet.	1	4	4
13	Arbeidet havner ganske langt bak tidsskjema i progresjonsplanen(Ca 1-2 uker).	2	3	9
14	Gruppa støter på tekniske utfordringer/ problemer og sitter fast i arbeidet.	4	3	12

5.2.2 Risikomatrise

	Konsekvens			
Sannsynlighet			4	12
		1, 7, 10, 11	13	2, 5, 6
		3, 8, 9	14	

Risikoverdier:

- 1-3: Akseptabel risiko. Risikoer som ikke trenger tiltak.
- 4-7: Vurderes for hvert tilfelle. Risikoer som må vurderes i hvert tilfelle om de trenger tiltak.
- 8+: Uakseptabel risiko. Disse risikoene burde reduseres med tiltak.

5.2.3 Tiltak til risikoer

Teknologimessige risikoer				
Nr	Tiltak	Rest sannsynlighet	Rest konsekvens	Rest risiko
2	Ta regelmessig backup av dokumentet separat fra kilden.	2	1	2
3	Fokus på å tenke før man handler. Ikke gjør ting man ikke er sikre på.	2	2	4

Forretningsmessige risikoer				
Nr	Tiltak	Rest sannsynlighet	Rest konsekvens	Rest risiko
5	Ta tak i problemet tidlig og si ifra til veileder.	1	2	3
6	God rutine for behandling av sensitiv informasjon	1	4	4
7	Holde kontinuerlige dialog med oppdragsgiver	1	2	2

Projektgruppemessige risikoer				
Nr	Tiltak	Rest sannsynlighet	Rest konsekvens	Rest risiko
8	Gruppa har felles ansvar å motivere hverandre og snu trenden.	3	1	3
9	Rutine for at man jobber litt hjemmefra hvis mulig, resten av gruppa tar ansvar for arbeidsoppgavene til den som er syk.	3	1	3
10	Ved uenigheter skal gruppa ha dialog og raskt komme til mest mulig enighet som tilfredsstiller flertallet. Hvis flertallet ikke klarer å bli enige vil gruppeleder bestemme i henhold til gruppereglene.	2	1	2
12	Gi tidlig beskjed til veileder for å få endret forventninger og mål, samt endre plan om nødvendig.	1	3	3
13	Gruppemøte og sette tiltak for å ta igjen tapt arbeid ved å f.eks øke arbeidstidene.	2	2	4
14	Fokus på å gjøre solid research. Prøve å løse problemet sammen som en gruppe, deretter oppsøke hjelp fra veileder og andre.	3	2	6

5.2.4 Risikomatrise etter tiltak

	Konsekvens			
Sannsynlighet		5, 7	12	6
	2, 10	3, 13		
	8, 9	14		

Risikoverdier:

- 1-3: Akseptabel risiko. Risikoer som ikke trenger tiltak.
- 4-7: Vurderes for hvert tilfelle. Risikoer som må vurderes i hvert tilfelle om de trenger tiltak.
- 8+: Uakseptabel risiko. Disse risikoene burde reduseres med tiltak.

References

Bell, T. (2017). *Containers on the cern cloud.* Retrieved from `http://superuser.openstack.org/articles/containers-cern-cloud`

Jameson, J. (2015). *Containerize openstack with docker.* Retrieved from `https://redhatstackblog.redhat.com/2015/07/16/containerize-openstack-with-docker/`

Morris, K. (2016). *Infrastructure as code.* O'Reilly Media,inc.

Ntnusky. (2019). Retrieved from `https://github.com/ntnusky`

Openstack. (2019). Retrieved from `https://www.openstack.org/`

Openstack documentation. (2018). Retrieved from `https://www.ntnu.no/wiki/display/skyhigh/Openstack+documentation`

What is kubernetes? (2018). Retrieved from `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`

Wikipedia contributors. (2018). *Secure by design — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/w/index.php?title=Secure_by_design&oldid=852560060`

Konteinertjenester for OpenStack

NTNU
Kunnskap for en bedre verden