Even Magnus Ofstad, Eivind Jørgen Gilje Dybvik

From Bare Metal to Scaling as a Service

Hovedoppgave i Drift av Datasystemer
Veileder: Tor Ivar Melling
Mai 2019

Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Even Magnus Ofstad, Eivind Jørgen Gilje Dybvik

From Bare Metal to Scaling as a Service

Hovedoppgave i Drift av Datasystemer
Veileder: Tor Ivar Melling
Mai 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

1 Abstract

This paper will go through the process of creating a cloud solution with on-premises bare-metal machines. It will show the process of getting Ubuntu MAAS, Canonical Juju, Ansible, and Kubernetes working on four HP workstation machines, with an L2 switch for networking. This paper will also show our process in documentation, including a cost benefit analysis, and a risk assessment analysis. It will show detailed information on how to install and use the system.

This paper was made as a part of a bachelors thesis written for SINTEF, as a way of supplementing their need for scaling computing power. The solution will provide a containerbased system orchestrated by Kubernetes, with TLS based authentication between each service that needs communication.

In this bachelors thesis, we will not be tackling the problems of backups, networking external from our cloud solution, or explicitly creating high availability through removing 'single points of failure'.

It is hoped that this paper will inform system-administrators on how to implement a bare-metal cloud solution, or inform system-administrators about how to operate the solution we have created.

Contents

2 Definisjoner

Navn	Forkortelse	Beskrivelse
Metal As A Service	MAAS	Et verktøy designet for å tilby maskinvare etter behov
Juju		Et program som hjelper med deployment av forskjellige tjenester
Ansible		En tjeneste som hjelper med å konfigurere flere servere
Kubernetes		En tjeneste som orkestrerer flere containere
Container		En container er et virtualisert miljø som er atskilt fra andre prosesser innen et OS
Docker		En tjeneste som setter opp containere
Blockchain		En blockchain er en distribuert database hvor hver node automatisk verifiserer endringer og tilføyelser som gjøres på noen av de andre nodene
DevOps		Et sett med utviklingspraksiser som kombinerer softwareutvikling og informasjonsteknologi-operasjoner.
Key Performance Indicator	KPI	En metrikk som forteller hvor effektivt en bedrift utfører oppgaver.
Preboot eXecution Environment	PXE	En funksjon som tillater oppstart av maskiner over nettverk
Basic Input-Output System	BIOS	Et program som er lagret å en brikke i en datamaskins hovedkort og kjøres når den blir slått på. BIOS sin primæroppgave er å forberede maskinvaren slik at programvae som kjører kan lastes. BIOS starter en maskin ved å lese de første sektorene på en disk.
Unified Extensible Firmware Interface	UEFI	Videreutviklet firmware som erstatter BIOS. I motsetning til BIOS som henter informasjon fra MBR, laster UEFI inn firmware fra en UEFI-partisjon på disken. Disken må da være partisjonert for GTP. BIOS er låst til 16 bit, mens UEFI kan benytte 32 og 64 bit kode.

Navn	Forkortelse	Beskrivelse
Baseboard Management Controller	BMC	En spesialisert mikrokontroller på et hovedkort. BMC håndterer arbeid mellom system-management software og platform hardware
Region Controller		En maskin som styrer flere region controllere
Rack Controller		En rack controller behandler flere noder i et MAAS cluster.
Network File System	NFS	En protokoll for distribuert filsystem. Tillater brukere å få tilgang til filer over et nettverk.
Pod		Et element som Kubernetes oprerer med. En pod inneholder en eller flere containere. Alle containere i poden deler IP med poden, men bruker særegne porter.
Persistent Volume	pv	et lagringsvolum for pods som trenger å lagre data
Persistent Volume Claim	pvc	Dette er et objekt Kubernetes bruker for å koble deployments opp mot persistent volumes.
Create, read, update and delete	CRUD	De fire grunnleggende funksjonene av lagringsområder.
Multitenancy		Referer til software som kjører på flere maskiner

3 Forstudierapport

4 Introduksjon

Hensikten med forstudierapporten er å legge til rette alle rammer som må på plass før et nytt prosjekt. Det vil så å kartlegge og dokumentere våre mål, rammer, suksesskriterier, estimerte kostnader og nytteverdier for prosjektet. I dette dokumentet skal vi bygge opp en forståelse over hva bachelorprosjektet skal innebære. Vi vil ta for oss bakgrunnen for oppgaven, hvorfor vi har blitt valgt til å opprette dette systemet, samt hvilke behov som skal oppfylles. Dermed skal vi kartlegge våre mål for oppgaven, som vil konkretisere hva vi skal oppnå.

- Bakgrunn for prosjektet
  - Beskrivelse av problemer og behov SINTEF har som denne bacheloroppgaven vil være et svar på.
- Prosjektmål
  - Mål for prosjektet med utgangspunkt i de problemer og behov som er kartlagt. Deles inn i effekt-, resultat-, og prosessmål
- Interessenter og rammebetingelser
  - Hvem som er involvert i noen grad i prosjektet. Hvilke roller disse involverte har, og hvem som er ansvarlig for hva.
  - Absolutte krav til prosjektet: ferdigdato, kostnadsramme, standarder o.l.
- Kristiske suksessfaktorer
  - Faktorer som tilsier om prosjektet har vært en suksess eller ikke. Disse faktorene er utarbeidet i samarbeid med oppdragsgiver.
- Risikoanalyse
  - Oversikt over forskjellige faktorer med tilhørende konsekvenser og sannsynlighet for prosjektet; både menneskelige- og teknologiske faktorer. Her vil det være et vedlegg som viser analysen.
- Kost/Nytte-analyse
  - Her finnes estimater på nytteverdi og kostnader for prosjektet, implementering og videre drift.
- Retningslinjer og standarder
  - Her finnes de retningslinjene og standardene som prosjektet må forholde seg til. Krav til dokumentasjon, kvalitetsgjennomganger, standarder, metoder, og endringshåndtering.

- Prosjektorganisering
  - Hvordan prosjektet blir organisert av arbeidsgruppen; arbeidsfordeling, roller, og ansvar.
- Anbefaling for videre arbeid
  - Oppsummering av resultatet av forstudiet, og en videre anbefaling utarbeidet fra resultatet.

5 Bakgrunn for prosjektet

SINTEF er et av Europas største uavhengige forskningsinstitutt som hvert år utfører flere tusen oppdrag for kundene sine. Med rundt 2000 ansatte har de kompetanse og ekspertise innen det meste av nyskapende teknologi. SINTEF ønsker i denne oppgaven å opprette en sky man kan installere og konfigurere ulike programmer på, og hvor man skal få tilgang til skalerbar datakraft. Deriblant skal Blockchain as a Service bli tatt i bruk for sikre handshake deals, samt effektivisering av oppgaver.

Selve bakgrunnen for prosjektet kommer fra en samtale en av oss hadde med Haro fra SINTEF på deres stand tidligere i høst. Her ble det ble diskutert ulike temaer innenfor AI (var på AI-konferanse), samt temaer vi senere skulle jobbe med i prosjektet. Diskusjonen gikk videre til en mulig bacheloroppgave, og i den anledning ble mailadressen til Haro utvekslet. Vi holdt kontakt via mail, og snart ble den ene personen til to studenter fra NTNU, Informatikk, Drift av Datasystemer. Etter noen møter og utveksling av informasjon ble vi til slutt enige om en spennende oppgave med vår veileder fra SINTEF.

5.1 Beskrivelse av problemer og behov

SINTEF trenger i dag en løsning som gir kunder mulighet til å utføre arbeidsoppgaver som er avhengig av blockchainteknologi. SINTEF jobber med store matvareprodusenter. som skal få igang en løsning som sporer matvarer gjennom matvarens 'livssyklus'. Det samme gjelder for fiskerinæringen i Norge som ønsker å implementere en ny måte å spore fiskestimer. SINTEF har notert et behov for å få i gang en skalerbar løsning som kan gi datakraft til disse blockchaintjenestene etter behov.

I denne bacheloroppgaven vil vi få operasjonsatt en løsning for skybasert blockchainteknologi. Dette gjøres via å produsere en fungerende sky med Ubuntu MAAS, for å tilrettelegge autoskalering. Juju kommer til å bli brukt til autodeploy av diverse tjenester vi vil tilby. For å utnytte datakraften som er tilgjengelig på best mulig måte tar vi i bruk Kubernetes, som er en container kontroller. Kubernetes kan automatisk utplassere containere ut i fra spesifikasjoner som vi setter, og vil oppdage hvis det er problemer med noen av de.

Dette vil være grunnlaget for oppgaven som går ut på å lage et effektivt rammeverk for tjenestehåndtering i en skybasert løsning hvor flere sammenstilte maskiner skal tilby automatisk robust virtualisering og dataflyt håndtering med støtte for "multitenancy". Oppgaven skal løse behov for "metal-as-a-service" hvor maskiner automatisk skalarer seg iht. pågang og ressursbehov slikt at man oppnår en devops flyt som ikke krever manuelle steg for automatisering av oppgaver.

5.2 Kort om dagens systemer og rutiner

Dagens systemer består av uavhengige maskinvare som hver virtualiserer diverse tjenester med statisk lastbalanse for å håndtere nettverkstrafikk og sørge

for tilstrekkelig ruting. Denne prosessen krever flere manuelle steg når nye tjenester skal settes i produksjon og klarer ikke tilfredstillende automatisere ressursallokering på tvers av servere.

6 Prosjektmål

Hovedmålet med prosjektet er å implementere et IT-system basert på en skalerbar sky som vil hjelpe SINTEF å sette til livs blockchainteknologien som skal hjelpe dem å overføre data sikkert og kryptert. Målet for SINTEF vil da være å kunne forhandle med bedrifter som krever en sikker overføring av data over nett.

6.1 Effektmål

Effekten SINTEF ønsker å oppnå med denne oppgaven er å redusere KPI mengden manuelle steg som må gjøres på serveresiden, og redusere dette med 90%. Dette vil i praksis si at vi vil redusere arbeidsmengden de ansatte bruker på oppgaver som skal erstattes av dette systemet med 90%.

6.2 Resultatmål

- Fullføre dokumentasjon
- Opprette en skybasert løsning med Kubernetes for bruk av SINTEF innen 20. Mai 2019
- Opprette et virtualiseringssystem som bruker containere fra Docker og bruker Kubernetes til å forenkle prosessen.
- Ved ferdigstilling av oppgaven, ha et system som skal være skalerbart for brukere, samtidig som det skal ha en oppetid på 99,99%
- Utføre prosjektet med et samlet forbruk på 1000 arbeidstimer. Et slingringsrom på +- 5% er akseptabelt
- Oppnå en kvalitet på prosjektet som tilsier at det kan produksjonssettes
- Automatisk skalering av fysisk hardware ved hjelp av Ubuntu MAAS
- Automatisering av serverressurser via Juju
- Automatisering av applikasjoner og konfigurasjon ved hjelp av Ansible
- Sette opp og konfigurere Docker og Kubernetes for mulighet til DevOps.

6.3 Prosessmål

- Bli kjent med og lære om hvordan en skalerbar skybasert løsning blir satt i drift
- Erfaring i form av praktisk oppgave hos en bedrift hvor oppgaven skal ut i næringslivet
- Ferdigstille forstudierapport innen tre uker.

- Ferdigstille systemkravrapport de to påfølgende ukene.
- Ferdigstille Driftsdokumentet innen de påfølgende 12 ukene.
- Ferdigstille sluttrapporten uken etter.
- Fullføre og bestå bacheloroppgaven med vurdering begge i prosjektgruppen er fornøyd med; B eller bedre
- Jobbe jevnt og systematisk fra start, slik at fremtidig stress blir minst mulig, samt at arbeidsmoralen holder seg på topp.

6.4 Prosjektets omfang

I dette prosjektet skal vi bygge opp en skalerbar skyløsning slik at brukere kan få tilgang til datakraft fra flere GPU-racks etter behov. Vi skal lage denne skyløsningen via Ubuntu MAAS (Metal As A Service), Docker, Kubernetes, Juju og Ansible.

- Prosjektet skal opprette et nytt system for skalerbare løsninger for SINTEFs ansatte og kunder.
- Utførelsen av løsningen skal bli gjort ved hjelp av Linux, men selve løsningen skal være kompatibel med alle salgs operativsystemer
- Om kjøp av hardware eller software trengs skal dette tas opp med veileder på SINTEF; Peter Haro
- Prosjektet skal ikke ha krav om brukeropplæring

6.5 Prosjektets milepæler og hovedaktiviteter

se gantt-diagram i vedlegg

7 Interessenter og rammebetingelser

7.1 Interessentanalyse

Interessenter:

7.1.1 Oppgavestiller

- SINTEF, representert ved Peter Haro
- Oppdragsgiveren skal gi rammer til oppgaven, kontrollere utførelsen underveis og skaffe til veie infromasjon om nødvendig
- Suksesskriterier for oppgavestiller er et fungerende skalerbart skybasert system, som har kvalitet på utførelsen og et godt dokumentert sluttprodukt
- Resultatet skal godkjennes av oppgavestiller

7.1.2 Veileder ved NTNU

- Tor Ivar er veileder fra NTNU
- Suksesskriteriene her er et godt gjennomført prosjekt hvor deltakerne har fått økt faglig kompetanse samt erfaring innen prosjektarbeid hos en reel bedrift

7.1.3 Sluttbrukere

- Kunder som benytter seg av systemets applikasjoner, i dette tilfellet Blockchain As a Service
- Kunder her er store fiskerederier, matprodusenter o.l.
- Suksesskriterier for kundene er at sikkerheten bedres og at systemet er til å stole på i form av oppetid

7.1.4 SINTEF

- Ansatte over flere avdelinger fra SINTEF som får bruk for et skalerbart skybasert system med mye datakraft
- Suksesskriterier for brukerne fra SINTEF vil være at systemet er lett å bruke og at den gir dem merkverdig forskjell i form av datakraft og scaling etter behov fra hva de bruker fra før

7.1.5 Prosjektgruppen

- Prosjektgruppen består av Eivind Dybvik og Even Ofstad, BADR16H
- Levere prosjektet med bidrag fra sine veiledere
- Arbeidet utføres av prosjektgruppen
- Prosjektgruppens suksesskriterie er å levere en løsning som oppfyller SINTEF sine behov og forventninger, samt oppfylle veilederens krav når det kommer til dokumentasjon og timer, men også oppfylle våres personlige mål; å få B eller bedre i karakter.

Offentlige aktører som trenger datasammenstilling

- Offentlige aktører som trenger data fra systemet og applikasjoner for å analysere og sammenstille denne dataen ved lignende systemer for å kvantifisere nytten den gir/ikke gir. Spesielt aktuelt i dette prosjektet, da vi her utvikler en teknologi som ikke brukes mye i det norske næringslivet.
- Suksesskriterier til slike aktører vil være et system som kan kvantifiseres, altså at man får tilgang til informasjon om systemet og at dataen kan analyseres. Det vil også være viktig at kundene får systemet og applikasjonen til å fungere.

Interessent	Suksesskriterier	Bidrag til prosjektet
Interne		
Oppdragsgiver	Systemet fungerer, kvalitet på utførelse	Kunnskap om problemområdet
Veileder fra NTNU	Positiv erfaring for studenter, økt kompetanse	Beslutninger, hjelp med flyt
Prosjektgruppe	Vellykket system og prosjektrapporter	Utførelse, ansvar
Ansatte ved SINTEF	Skalerbar med mye datakraft, merverdi av bruk	Eventuelle testere
Eksterne		
Kunder	Sikkert system, fungerende system	Motivasjon for utførelsen
Offentlige aktører	Kvantifiserbart system for datahenting, lansert og fungerende system for kundene	Analyse av data for videreutvikling

7.2 Rammebetingelser

Listing og beskrivelse av absolutte krav som stilles til prosjektets gjennomføring som vil ha en avgjørende innflytelse på planer og valg i prosjektet.

- Endelig dato for oppsett og ferdigstilling av det nye systemet er satt til 20.05.2019.
- Alle tjenester og systemer skal være kompatibel med linux.
- Prosjektet og systemet skal dokumenteres i form av fem rapporter:
    1. Forstudierapport
    2. Systemkravrapport
    3. Driftsrapport
    4. Sluttrapport
    5. Refleksjonsnotat

8 Kritiske suksessfaktorer

8.1 Suksessfaktorer

Suksessfaktor	**Beskrivelse**
Lite arbeidsfravær	Vår gruppe består bare av 2 personer. På et prosjekt som går over lengre tid som dette, vil mye sykefravær eller annet fravær fra arbeid ha negativ innvirkning på prosjektets gjennomføring.
Tilstrekkelige spesifikasjoner på servere	Hele prosjektet omhandler arbeid på servere, og alt av arbeid henger på om servermaskinene greier å kjøre programvaren til en tilstrekkelig grad.
Gode muligheter for high availability og backup	Siden alt vi gjør er avhengig av oppetiden til serverene, vil dette være kritisk for at prosjektet blir en suksess
God samarbeidsvilje internt i prosjektgruppen	Vår gruppe har ikke jobbet sammen før med en praktisk oppgave, dette gjør at alle medlemmene i gruppa må uttrykke meningene sine i planleggingsfasen av prosjektet.
God dokumentasjon gjennom prosjektet	Vår gruppe kommer til å produsere flere dokumenter, som beskrevet i punkt 8.1 "Krav til dokumentasjon". Dokumentasjon som omhandler drift av systemet vil være vesentlig for ansatte i SINTEF som skal ta over driften etter prosjektet er ferdig. Dokumentasjonen må beskrive generell bruk, hvordan løsningen er satt opp, og tilganger som trengs av den driftsansvarlige.

8.2 Informasjonsbehov

Formålet her er å fastlegge behovet for informasjonsformidling i prosjektet, altså hvilken informasjon har vi bruk for, og hvilken informasjon har de ulike interessentene behov for.

Oppgavestiller

Prosjektgruppen vil ha behov for informasjon om deler av SINTEF til bruk i kost/nytte-analysen, samt deres spesifikke krav og tanker til prosjektets løsning og omfang. For oppgavestiller vil tilegning til informasjon komme gjennom den kontinuerlige prosessen ved å veilede oss gjennom prosjektet. Den endelige informasjonen vil komme gjennom dokumentasjonen.

Veileder på NTNU

Vil ha behov for statusoppdateringer på prosjektet enten ukentlig eller annen hver uke gjennom møter. Det skal sendes ut møteinnkallelse minst 2 dager i forveien, med saksliste, samt relevante dokumenter for møtet. Veileder vil også i disse møtene stille til disposisjon for faglig veiledelse om nødvendig, enten det skulle være hjelp til dokumentasjon eller tips til forbedring av systemet. Denne informasjonsformidlingen er viktig og dersom noen skulle være utilgjengelig over flere uker, da skal man ta møtet over Skype for Business eller lignende.

Offentlige aktører

Ønsker informasjon om data på systemet og applikasjoner som skal i drift. Til gjengjeld får vi kvantifiserbar dataanalyse over systemet, gjerne også sammenlignet med andre lignende systemer og applikasjoner.

Resterende interessenter vil ikke ha behov for informasjon gjennom prosjektperioden, men vil heller få ønsket informasjon senere gjennom ferdig dokumentasjon eller muntlig beskrivelse av systemet.

9 Risikoanalyse

I Denne risikoanalysen skal vi se på hva som kan gå galt, og hvordan man skal håndtere og/eller forebygge disse problemene. Vi vil sette opp denne informasjonen i et separat skjema for å holde informasjonen oversiktlig. I vedlegget har vi tatt med de risikoene vi mener har en sannsynlighet for at vi møter på. Disse risikoene kan være både tekniske og sosiale.
Se vedlegg
Risikoanalyse

10 Kost/Nytte-analyse

10.1 Alternativet

- I dag bruker SINTEF ca. et halvt årsverk pr år på å manuelt sette opp diverse løsninger for kunder og ansatte som trenger tilgang til datakraft.
- Vi antar en årslønn på ca.800,000NOK for en senior nettverksingeniør, og vil derfor si at det halve årsverket vil bli ca. 400,000NOK pr år for den manuelle jobben.
- Vår kontakt hos SINTEF sier de sparer 200,000NOK på at vi i bachelorprosjektet utfører dette arbeidet istedet for de. Dette skal regnes med i vår analyse.
- Vi bruker bare open-source software, og vil dermed ikke ha noen utgifter for lisenser eller kjøp av software.
- Hardware vi bruker eksisterer allerede hos SINTEF, eller er såpass billig at det ikke utgjør noen merkbar forskjell på budsjettet, det blir derfor ikke regnet med i denne analysen.

10.2 Kvantifiserbar og ikke-kvantifiserbar nytte

Kvantifiserbar nytte:

- Med denne løsningen kommer vi til å redusere arbeidsmengden senioringeniører vil bruke på manuelt oppsett av maskinvare og software til diverse prosjekter som dette systemet kommer til å kjøre. Denne arbeidsmengden vil reduseres med 90%.

Ikke-kvantifiserbar nytte:

- En automatisert løsning vil redusere behovet for gjentakende arbeidsoppgaver knyttet til manuelt oppsett av servere. Dette vil gi ansatte bedre tid og redusere deres arbeidsbelastning knyttet til oppgaven. Bedre tid kan føre til at ansatte får utført andre arbeidsoppgaver.

Vedlegg:
Kost/Nytte-analyse

11 Retningslinjer og standarder

11.1 Krav til dokumentasjon

- Systemkravrapport: Denne rapporten skal dokumentere hva vi trenger av hardware og software for at prosjektet skal kunne gjennomføres. Denne rapporten skal være ferdig innen 06.02.2019
- Driftsdokument: Her blir systemet vi lager dokumentert. Dette dokumentet vil beskrive hvordan systemet er satt opp, og hvordan man skal drifte systemet. Dette dokumentet skal være ferdig 01.04.2019
- Sluttrapport: Dette blir en avsluttende rapport hvor vi skal evaluere prosjektarbeidet vårt.

11.2 Krav til kvalitetsgjennomganger

- Stresstest av system. Man må teste hvor mye belastning tjenesten takler før den blir produksjonssatt, da systemet kan bli dyrt hvis kundene ikke får tilgang til systemet under perioder med mye pågang.
- Brukertester. For å få systemet til å fungere som oppdragsgiver vil, må vi få brukere til å teste systemet før produksjonssetting, da det er de som til sist skal ha kontinuerlig kontakt med systemet.
- Resilienstest. Resilienstesten skal si oss om systemet takler feilkonfigurerte tjenester, og om det kan automatisk opprette tapte forbindelser, og ikke krasje når det oppstår feil.

11.3 Krav til standarder og metoder

Passordstandard:
Vi vil ha en passordstandard av høyeste kvalitet. Alle passord skal være sammensatt av en tilfeldig setning, hvor tall, store bokstaver og spesialtegn er med. Ved generering av passord vil bokstaver som egner seg som tall bli byttet (f.eks. "e" blir til "3" og "i" blir til "1". Alle passord skal lagres i Passpack.
Brukernavnstandard:
Ved opprettelse av brukernavn skal man ha de tre første bokstavene i fornavnet etterfulgt av de tre første bokstavene i etternavnet. Om noen vil få like brukernavn skal nummerering i påstigende rekkefølge gjelde. Da systemene og programmene vi bruker er utenlandske skal man oversette de norske tegnene "Æ, Ø og Å" til "AE, O og AA".

11.4 Endringshåndtering

Ved en endringsanmodning vil vi ta for oss disse stegene for å bestemme om endringen er kost-effektiv, og verd å implementere i prosjektet. Vi mener denne rutinen vil kunne holde kvaliteten vi ønsker fra dette prosjektet.

1. Dokumenter endringens innhold
2. Analyser konsekvensene for prosjektet
3. Beregn eventuell kost/nytte
4. Godkjennelse og aksept
5. Logg endringen
6. Juster planene
7. Informer aktuelle interessenter
8. Gjennomfør endringen

12 Prosjektorganisering

I dette prosjektet har Even Ofstad og Eivind Dybvik tatt på seg en arbeidsmetode som går ut på å jobbe parallelt om oppgavene vi har påtatt oss. Det vil si å skrive dokumentasjon mens man utfører oppgaven. Dette er for å utnytte ressurser på en mest mulig effektiv måte, da man alltid vil ha arbeid pågående i arbeidstid. Arbeid utenom vanlig arbeidstid skal meldes inn og skrives ned i vår timeliste, slik at alt arbeid blir regnet med i vår sluttrapport.

13 Anbefaling om videre arbeid

Vi har i denne rapporten beskrevet omfanget av oppgaven, satt kvantifserbare og ikke-kvantifesrbare mål som vi mener vil oppnås gjennom kontinuerlig og seriøst arbeid. Vi har også gjenomført ulike analyser om riskoer, suksessfaktorer og de ulike interessene som vil være påvirket av prosjektet. Gjennom våre analyser og vurderinger av prosjektets omfang vil vi anbefale at prosjektet videreføres med de rammene og planene vi har lagt frem i forstudierapporten. Oppgaver som bør ses mer på etter oppgaven er ferdigstilt:

- Backupløsning.
- Fjerning av single points of failure.
- Oppsett av blockchain-rammeverk.
- Omdirigere nett-trafikk
- Bedre utnyttelse av hver node
- Oppsett av ekstern Power Supply Unit
- Monitorering av tjenestene
- Dedikert brannmur

14 Systemkravrapport

15 Hensikt med dokumentet

Dette dokumentet skal gi en innsikt i hva man trenger av software og hardware for at prosjektet kan anses som en suksess.

- Overordnet prosjektbeskrivelse/systembeskrivelse

  Her finner man en oversikt over systemet og litt om hvordan det henger sammen

- Detaljerte krav til egenskaper

  Her beskrives kravene man har til systemets egenskaper. Hva må systemet oppnå for at det skal kunne brukes?

- Spesifikasjon av systemets funksjonelle egenskaper

  Her blir systemets funksjon illustrert og beskrevet på en oversiktlig måte.

- Databeskrivelse

  Her vil hvert komponent i systemet bli beskrevet, og et Entity-Relationship-diagram vist.

- Krav til systemdokumentasjon

  Her spesifiseres krav til maskinvare, service, maskiner som må kjøpes, programvare som brukes.

- Krav til dokumentasjon

  Her finner man brukerveiledning, driftsdokumentasjon, forvaltningsdokumentasjon.

- Krav til manuelle arbeidsrutiner

  Her beskrives rutiner som skal til for å vedlikeholde systemet.

- Regler for godkjenningsprøve

  Her skal godkjenningsprøven beskrives.

- Reviderte resultater fra forstudiet

  Her viser vi forandringer siden forstudiet.

16 Overordnet prosjektbeskrivelse/systembeskrivelse

SINTEF er en forskningsbasert bedrift som utvikler og tilbyr den nyeste teknologien som finnes til sine kunder og ansatte. I den siste tiden har flere kunder etterspurt en blockchainteknologi som kan verifisere data i handshake deals. Kombinert med dette ville SINTEF ha en bedre metode for virtualisering enn hva de har i dag. Prosessen som blir gjort i dag på egne arbeidsstasjoner som hver virtualiserer diverse tjenester med statisk lastbalanse for å håndtere nettverkstrafikk og sørge for tilstrekkelig ruting. Løsningen for disse behovene ble å lage en bare-metal sky med mye datakraft som kan autoskalere etter behov, og som har mulighet for opprettelse av containere.
Rutinene som da blir dekket er opprettelse av virtuelle tjenester som ansatte kan ta i bruk. Disse tjenestene vil være i form av Kubernetes som orkestrerer containere. Prosessen ved å kjøre noe virtuelt på containere vil være mye enklere og vil kunne støtte et større behov, samt at nedetiden vil være tilnærmet null. Disse gjør det mulig å kjøre ut det man har lyst til i et virtuelt miljø, ofte en applikasjon som skal testes eller rulles ut til kunder. Den automatiske skaleringen Kubernetes har, sammen med automatiseringen Ansible tilbyr, vil gjøre de nye rutinene for de ansatte merkbart bedre og spare dem masse tid. Systemet har innebygd High Availability for containerene, og dermed tjenestene som kjører på de. Dette fører til høy oppetid som eventuelle kunder kommer til å kreve.
Utstyret vi har fått til disposisjon for å realisere denne skyen er fire kraftige servere, en switch og en ruter, med mulighet til å skaffe mer om nødvendig. Det er prosjektgruppen som har hovedansvaret for å installere og konfigurere den skalerbare skyen, med veiledning fra nevnt veileder på SINTEF om nødvendig. Selve programvaren vi bruker har åpen kildekode og er dermed gratis å ta i bruk. Selve blockchainteknologien vil kjøre på skyen vi implementerer, men utviklingen av den vil ansatte på SINTEF ta seg av. Vi skal dog i samarbeid med dem konfigurere det riktig.

16.1 Kort overordnet beskrivelse av det nye systemet

I dette prosjektet vil vi få operasjonsatt en løsning for skybasert blockchainteknologi som skal autoskaleres etter behov. Dette gjøres via å produsere en fungerende sky med Ubuntu MAAS, og programvaren Juju som vil tilby autodeploy av diverse tjenester vi tilbyr. For best utnyttelse av skalerbar datakraft ved virtualisering har vi valgt å ta ta i bruk Docker sammen med Kubernetes. Docker er plattformen som vil stå for bygging, distribuering og drift av containere, mens Kubernetes vil effektivisere bruken av mange containere samtidig, samt gjøre skalering av containere enklere. Dette vil være grunnlaget for oppgaven som skal lage et effektivt rammeverk for tjenestehåndtering i en skybasert løsning. Flere sammenstilte maskiner skal tilby automatisk robust virtualisering og dataflyt håndtering med støtte for "multitenancy". Oppgaven skal løse be-

hov for "metal-as-a-service" hvor maskiner automatisk skalarer seg iht. pågang og ressursbehov slikt at man oppnår en devops flyt som ikke krever manuelle steg for automatisering av oppgaver.

16.2 Organisatoriske og personellmessige konsekvenser

Som nevnt i beskrivelse av systemet skal vi lage en skalerbar sky som vil bli implementert i SINTEF. Ved opprettelse av denne skyen vil personale i bedriften merke en forskjell ved at oppgaver som krever en virtuell maskin eller skalerbar datakraft vil bli mye mer automatisert og enklere å ta i bruk. Det vil bli mindre ansvarsforhold da de manuelle rutinene som var på plass før har blitt byttet ut med et mer automatisert system hvor det eneste som trengs er en drift-ansvarlig som kan vedlikeholde og fikse eventuelle problemer og bugs som skulle oppstå. For SINTEF vil det nye systemet skape flere prosjekter og utvikling av ny type programmer som blir lettere å implementere. Et eksempel på dette er blockchain-teknologien som skal bli implementert ved hjelp av skyen og som allerede har avtaler om prosjekter med implementasjon av denne teknologien som hovedelement. Som nevnt i vår kost/nytte analyse vil organisasjonen også tjene på implementasjon av skyen i bedriften etter tre år, noe som er veldig positivt for et system som gjør et eldre system enklere.

17 Krav til egenskaper

1. **Funksjonalitet** - Hvilke funksjoner og behov som skal være på plass
    - Enkelt system å ta i bruk for sluttbrukere
    - For opprettelse av containere og virtuelle maskiner
    - En nærmest automatisk prosess
    - Skalerbar sky som opprettholder kapasiteten
    - Skal kjøres på Ubuntu, men kan tas i bruk av alle operativsystem
2. **Pålitelighet** - Hvor lenge klarer systemet å opprettholde sitt ytelsesnivå under ulike forhold under en angitt tidsperiode
    - Systemet skal ha kraftig maskinvare som skal kunnes skaleres etter behov, påliteligheten til dette ytelsesnivået under ulike forhold skal være svært høy. I og med at dette blir en skalerbar skyløsning må systemet ha 99.99% oppetid
3. **Effektivitet** - Effektiviteten av systemet
    - Effektivt system med en merkbar forskjell til det forrige
4. **Vedlikeholdbart** - Hvor ofte må det vedlikeholdes, mye bugs, bra oppetid?
    - Svært sjelden vedlikehold
5. **Brukervennlighet**
    - Systemet skal være meget enkelt og brukervennlig å ta i bruk for sluttbrukerene. Det vil si at det eneste sluttbrukerne skal trenge å gjøre er å bestemme hvor mye datakraft som trengs for programmet sluttbrukeren ønsker å kjøre og trykke på en knapp for opprettelse av container som innfrir dette behovet. Ved Ansible skal de bare trenge å ha frem koden de vil kjøre og til hvilke maskiner, så vil Ansible gjøre resten.

6. **Krav til dokumentert system**

   - Skal inneholde minst følgende dokumentasjon: Forstudierapport Systemkravrapport Driftsdokument Sluttrapport. Disse skal beskrive følgende
   - Hvordan løsningen fungerer
   - Hvilke metode, programmeringsspråk og software som har blitt brukt og hvorfor.
   - Ytelsen på systemet
   - Brukerveiledning til systemet
   - Steg for opprettelse av systemet
   - Videre utvikling
   - Begrensninger og muligheter ved systemet

18 Spesifikasjon av systemets funksjonelle egenskaper

Her beskrives hvordan systemet vil operere når det er ferdig implementert. Det skal beskrives hvordan diverse prosesser utføres.

18.1 Funksjonell beskrivelse basert på DFD-diagrammer

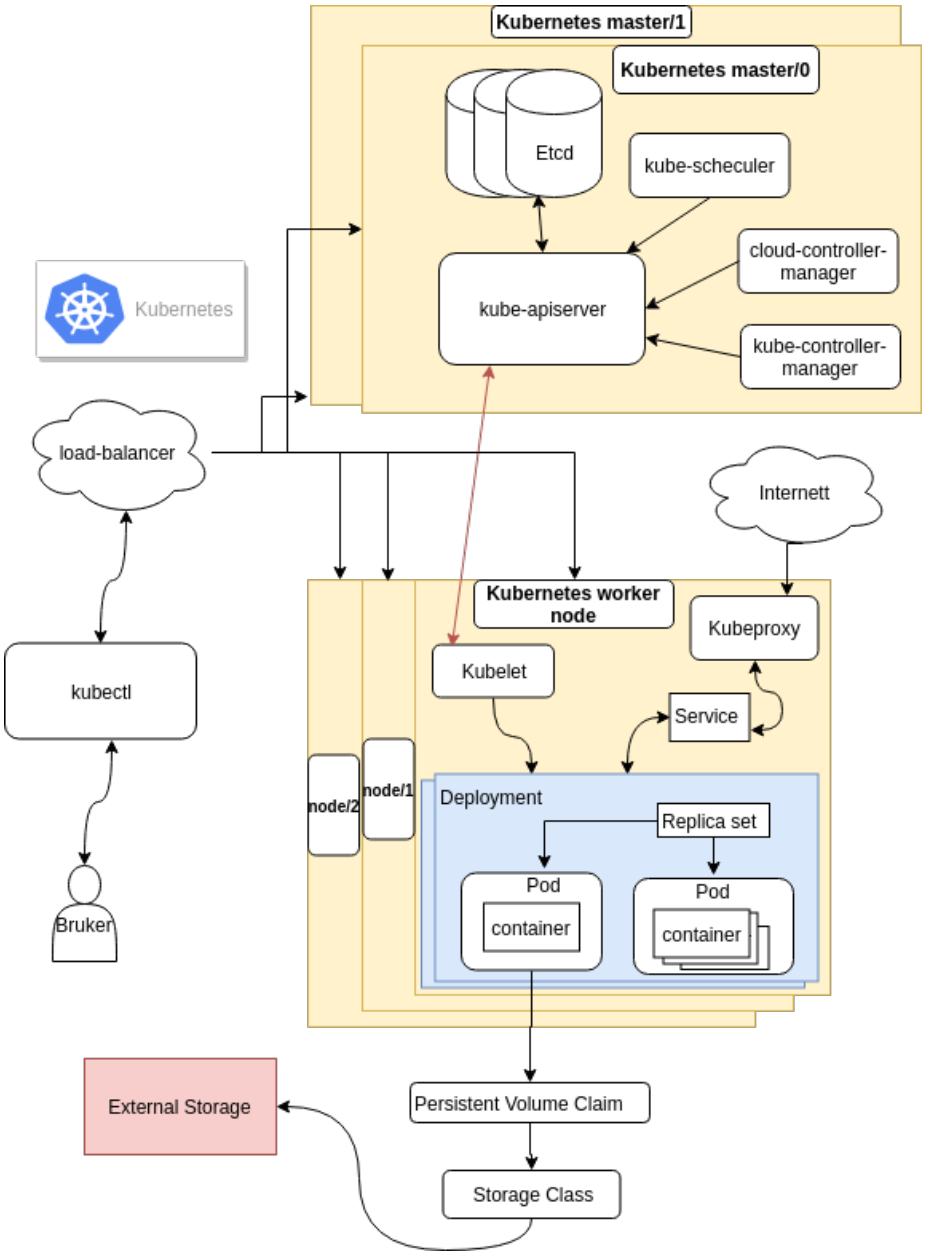


Figure 1: oversikt over Kubernetes

18.1.1 Beskrivelse av funksjonene

[18]

- **Master-komponentene:**
  Master node er ansvarlig for styringen av Kubernetes-clusteret. Dette er selve inngangspunktet til alle de administrative oppgavene Kuberentes tar seg av. Master noden tar seg også av orkestrering av worker-nodene, hvor de faktiske tjenestene kjører. Vi har to master-noder for å gi systemet High Availability (HA). Her vil vi se nærmere på hver av master-komponentene som master-noden består av, og som gir clusterets kontrollpanel.

- API server
  Kompenent på masteren som eksponerer Kubernetes API-et. Det er front-end for Kubernetes kontrollpanelet. API-serveren er inngangspunkter for alle REST-kommandoene som brukes til å styre klassen. Den behandler REST-forespørsler, validerer dem og utfører den begrensede forretningslogikken. Resultatstatusen må fortsette et sted, og det bringer oss til neste komponent i master-noden.

- Etcd storage
  etcd er en enkel, distribuert, konsistent nøkkelverdibutikk. Den brukes hovedsakelig til delt konfigurasjon og serviceoppdagelse. Den gir en REST API for CRUD-operasjoner, samt et grensesnitt for å registrere overvåkere på spesifikke noder, noe som gjør det mulig å varsle resten av klassen om konfigurasjonsendringer.

  Et eksempel på data lagret av Kubernetes i etcd er jobber som planlegges, opprettes og distribueres, pod / service-detaljer og state, namespaces og replikasjonsinformasjon, osv.

- Flannel [17]
  Flannel er en tjeneste som ligger hos hver Kubernetes node. Denne tjenesten lager et logisk nettverk som Kubernetes bruker til intern kommunikasjon i clusteret.

- Scheduler
  Selve deploymenten av konfigurerte pods og services på nodene skjer takket være scheduler-komponenten. Scheduleren har informasjon om ressursene som er tilgjengelig på medlemmene av clusteret, samt de som kreves for at den konfigurerte tjenesten skal kjøre, og dermed kan bestemme hvor en bestemt tjeneste skal distribueres.

- Control-manager
  Det er vanlig å kjøre flere typer kontrollere inni master-noden. Kontrollstyreren er en tjeneste som integrerer disse. Logisk sett er hver kontroller en separat prosess, men for å redusere kompleksiteten, er alle samlet og kjøres som en prosess. En kontroller bruker API-serveren til å se over statusen til det delte clusteret og gjør korrigerende endringer i gjeldende tilstand for å endre den til ønsket tilstand. Et eksempel på en slik kontroller

er Replikasjonskontrolleren, som tar seg av antallet pods i systemet. Selve faktoren til replikasjonskontrolleren er konfigurert av brukeren, og det er kontrollerens ansvar å gjenopprette en mislykket pod eller fjerne en som ikke tas i bruk. Andre eksempler på kontrollere er endpoints-controller, namespace-controller og service-accounts-controller.

Node-komponentene
Node-komponentene kjører på enhver node, opprettholder kjørende pod-er og vedlikeholder dem.

- **Container**
  Containere er en fleksibel og lettvekts plattform. Det bruker det underliggende operativsystemet slik at den som tar i bruk containeren bestemmer akkurat hva som skal kjøres på den. Ofte vanlig å kjøre en tjeneste per container. Den fungerer altså som en lettversjon av en virtuell maskin.

- **Docker**
  Det er Docker (som er en programvareteknologi) som står bak denne virtualiseringen på operativsystemnivå, også kjent som containere

- **Kubelet**
  En agent som kjører på hver node i klusteret. Den sørger for at containere kjører i en pod. Kubelet tar et sett med PodSpecs (spesifikasjon av poddens ønskede oppførsel) som leveres gjennom ulike mekanismer og sikrer at beholderne beskrevet i disse PodSpecs-ene kjører og er healthy.

- **Pod**
  Det minste og enkleste Kubernetes-objektet. En Pod representerer et sett med en eller flere containere på clusteret.

- **Namespace**
  En funksjon fra Kubernetes som støtter flere virtuelle clustere på samme fysisk cluster. Namespaces brukes til å organisere objekter i et cluster og for å gi en måte å dele klusterressurser på.

- **Workload**
  Workloads er objekter du bruker til å styre, vedlikeholde og kjøre containere på klusteret. Kubernetes utfører distribusjonen og oppdaterer workloaden slik at den samsvarer med den nåværende tilstanden av applikasjonen. Workload inkluderer DaemonSet, Deployments, Jobs, Pods, ReplicaSet, ReplicationController, og StatefulSet objekter.

18.2 Funksjonell beskrivelse

18.2.1 Beskrivelse av use-case (bruksmønster)

Vårt system er grunnmuren som SINTEF kommer til å kjøre applikasjoner på. Eventuelle use-case kommer bare til å bli relevant etter SINTEF har satt opp disse applikasjonene. Derav har vi ingen use-case.

19 Databeskrivelse

19.1 Dataelementbeskrivelsen

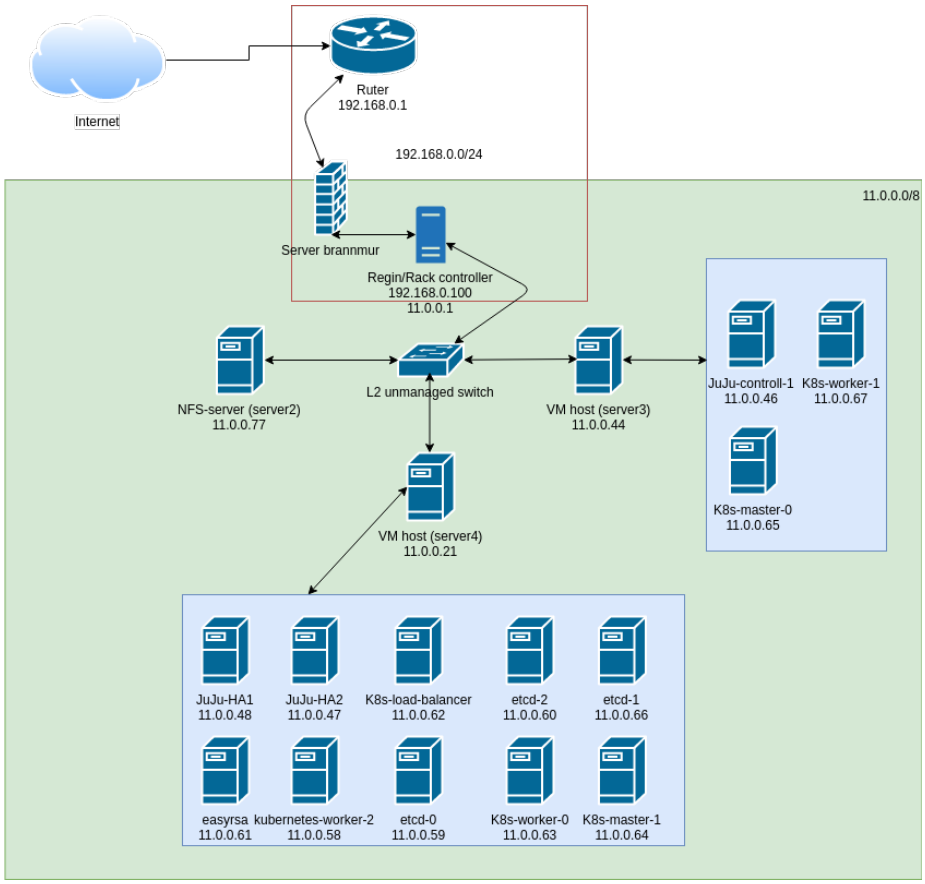


Figure 2: oversikt over systemet

Systemet
Systemet vårt består i praksis av to soner. En som består av fysiske maskiner og virtuelle maskiner via MAAS. Dette er bare-metal siden av systemet. Her har vi en Region/rack controller med IP 11.0.0.1, som styrer DCHP, PXE, DNS, MAAS, og har Juju administrator. På Figur 1 viser vi også et nett mellom Region/rack og ruter. Dette er fordi den ruteren vi har ikke er en optimal ruter til bruk direkte til vår arbeidssone. Som det opererer per nå går all trafikk gjennom denne serveren. Den andre sonen er et logisk nettverk som Kubernetes bruker til å kommunisere internt.

Vi har så en L2 switch hvor alt av maskiner er koblet til. Denne switchen er

fysisk koblet til alle våre bare-metal servere.
Server2:
Dette er vår NFS-server. Denne tilbyr lagring til hele systemet.
VM-host (server3):
På denne maskinen kjøres 5 virtuelle maskiner hvorav tre er i bruk.

- Juju-controll-1: Dette er vår Juju-controller.
- K8s-worker-1: Dette er en av våre Kubernetes worker-nodes
- K8s-master-0: Dette er en av våre Kubernetes master-nodes

VM-host (server4): På denne maskinen kjører vi 10 maskiner hvor alle er i bruk.

- Juju-HA1: Dette er en high-availability node for Juju.
- Juju-HA2: Dette er en high-availability node for Juju.
- K8s-load-balancer: Dette er en node som kjører Kubernetes-loadbalancer. Her blir trafikk i kubernetes-clusteret optimalisert.
- etcd-2: Dette er en node som kjører Etcd for Kubernetes.
- etcd-1: Dette er en node som kjører Etcd for Kubernetes.
- easyrsa: På denne noden kjøres Easyrsa, som sikrer TLS for kommunikasjon mellom Kubernetesobjekter.
- Kubernetes-worker-2: Dette er en av våre Kubernetes worker-nodes.
- etcd-0: Dette er en node som kjører Etcd for Kubernetes.
- K8s-worker-0: Dette er en av våre Kubernetes worker-nodes.
- K8s-master-1: dette er en av våre Kubernetes master-nodes.

Cluster Selve clusteret er hele systemet i Kubernetes og består av et sett med maskiner (kalles noder), som kjører containerbaserte applikasjoner som blir styrt og vedlikeholdt av Kubernetes. Et cluster har flere worker-noder og minst en master-node (ofte flere for å oppnå High Availability)

20 Krav til systemkonstruksjon

Maskinutstyr som trengs i dette prosjektet på samme nettverk:

- Hovedserver med MAAS, Juju, Ansible, Kubernetes
- Maskiner med Baseboard Management Controller (BMC)
- Server(e) for testing
- To arbeidsstasjoner for testing og utvikling
- Skjerm, mus og tastaur

21 Krav til dokumentasjon

Følgende dokumentasjon skal leveres med systemet

- Forstudierapport
  En rapport som skal beskrive hva som må gjøres i prosjektet, samt diverse analyser, som risikoanalyse
- Systemkravrapport.
  Denne rapporten skal dokumentere hva vi trenger av hardware og software for at prosjektet skal kunne gjennomføres.
- Driftsdokument.
  Her blir systemet vi lager dokumentert. Dette dokumentet vil beskrive hvordan systemet er satt opp, og hvordan man skal drifte systemet.
- Sluttrapport.
  Dette blir en avsluttende rapport hvor vi skal evaluere prosjektarbeidet vårt.

22 Reviderte resultater fra forstudiet

22.1 Reviderte prosjektmål

Hovedmålet med prosjektet er å implementere et IT-system basert på en skalerbar sky som vil hjelpe SINTEF å sette til livs blockchainteknologien som skal hjelpe dem å overføre data sikkert og kryptert. Målet for SINTEF vil da være å kunne forhandle med bedrifter som krever en sikker overføring av data over nett.

22.2 Reviderte rammebetingelser

Listing og beskrivelse av absolutte krav som stilles til prosjektets gjennomføring som vil ha en avgjørende innflytelse på planer og valg i prosjektet.

- Endelig dato for oppsett og ferdigstilling av det nye systemet er satt til 20.05.2019.
- Alle tjenester og systemer skal være kompatibel med linux-miljøet.
- Prosjektet og systemet skal dokumenteres i form av fem rapporter: Forstudierapport, Systemkravrapport, Driftsrapport, sluttrapport og refleksjonsnotat.

22.3 Revidert risikoanalyse

Revidert risikoanalyse

23 Driftsdokument

24 Innledning

Dette dokumentet beskriver løsningen for vår problemstilling, som handler om å lage et skalerbart skybasert system hvor det skal opprettes containere for kjøring av programmer og diverse for kunder og ansatte av SINTEF. Dokumentet består av avgrensninger for oppgaven, konseptuell beskrivelse av løsningen og strategi og metode brukt.

25 Avgrensning

Hva som trengs for å få en overkommelig oppgave, hva som ikke skal være med i oppgaven. Handler om spesifikk beskrivelse av løsningen vi har tatt i bruk: SINTEF ønsker en løsning som kan automatisere oppretting av containere og autoskalere disse ved hvilket behov som trengs. Systemet skal ved hjelp av MAAS og Juju kjøres på en bare-metal sky, der bare-metal vil være SINTEF's maskinvare i form av servere. Kubernetes sammen med Ansible skal gjøre systemet mest mulig skalerbart og forenkle prosessen ved oppretting av containere, slik at de som skal kjøre programmene sine på skyen vi lager, ikke skal trenge å sette seg inn i noe om hva vi har gjort for å ta det i bruk. Selv om vi skal være med å kjøre programmer som Blockchain as a Service på skyen vår, skal vi ikke ha noe med selve utviklingen av slike programmer og heller ikke implementeringen av disse på systemet vårt.

26 Oversikt over innholdet

26.0.1 Begrunnelse av teknologivalg

I dette kapittelet vil vi få fram hvilke teknologier som ble valgt for å gjennomføre prosjektet, og hvorfor de ble valgt.

26.0.2 Beskrivelse av tekniske løsninger

Her beskriver vi en oversikt over systemet, og viser litt mer i dybden hva forskjellige teknologier gjør.

26.0.3 Strategi og metode

En beskrivelse av vår prosess, og informasjonssanking.

26.0.4 Løsningsbeskrivelse

Her beskrives i detalj hvordan systemet ble satt opp.

27 Begrunnelse av teknologivalg

Utenom planlegging av oppgaven har vi ikke hatt mye valg når det kommer til hvilken type teknologi vi skulle benytte oss av. I planleggingsfasen ble vi fremstilt for en plan for et system som skulle ta i bruk mye ukjent teknologi. Hva som skulle opprettes var underforstått; en bare-metal sky der det skulle installeres og konfigureres programmer som gjør det mulig for oppretting av containere med autoskalering. Teknologier vi har tatt i bruk er:

- MAAS
- Juju
- Kubernetes
- Ansible

Grunnen for valget av disse teknologiene er mye på grunn av Canonical, som er de som laget Ubuntu. Det er også de som står for MAAS og Juju. Ved å ha en så stor bedrift som står for programmene som lager selve grunnmuren for systemet vårt, gjør at det finnes mye informasjon om vår situasjon på nettet. De har en solid dokumentasjon samt discourses hvor du kan stille spørsmål om problemer og få svar (ofte fra utviklere hos Canonical selv), ofte i løpet av noen dager. Videre har vi valgt Kubernetes da dette er et av kriteriene som ble satt til oss. Kubernetes passer perfekt i en sammenheng som dette, da det er et orkestreringsverktøy designet av Google for å organisere og kjøre store mengder med containere.

28 Beskrivelse av tekniske løsninger

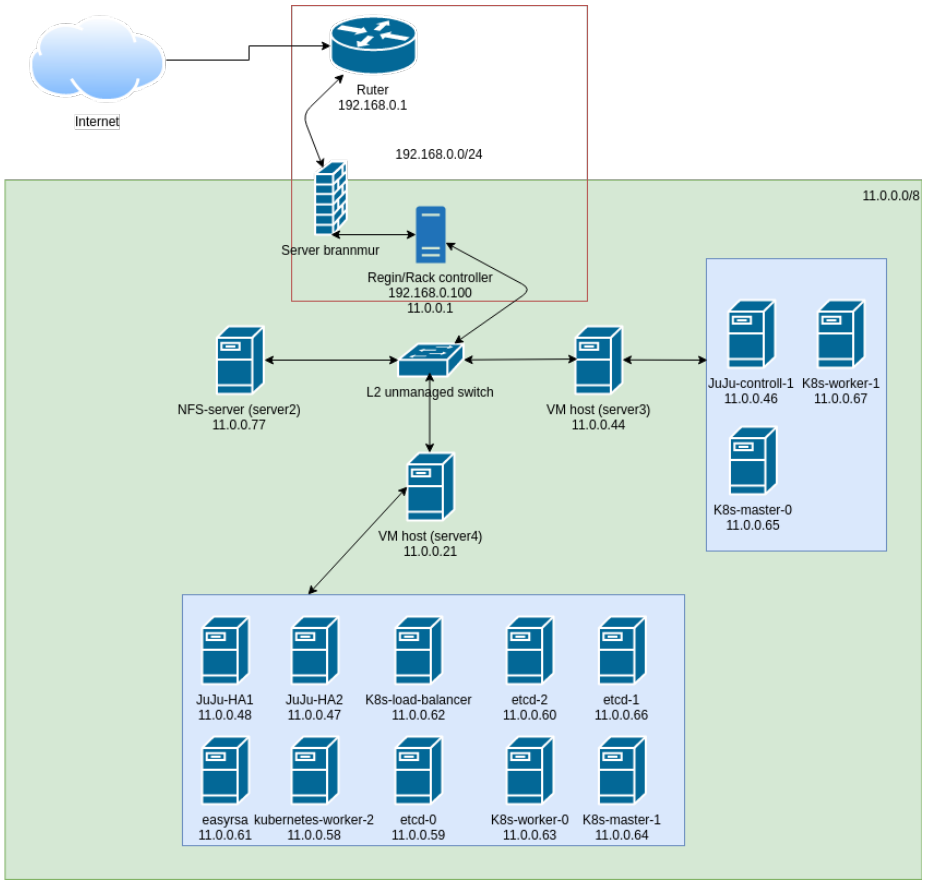


Figure 3: oversikt over systemet

Systemet

Systemet vårt består i praksis av to soner. En som består av fysiske og virtuelle maskiner via MAAS. Dette er bare-metal siden av systemet. Her har vi en Region/rack controller med IP 11.0.0.1, som styrer DCHP, PXE, DNS, MAAS, og har Juju administrator. På Figur 1 viser vi også et nett mellom Region/rack og ruter. Dette er fordi den ruteren vi har ikke er en optimal ruter til bruk direkte til vår arbeidssone. Som det opererer per nå går all trafikk gjennom denne serveren. Den andre sonen er et logisk nettverk som Kubernetes bruker til å kommunisere internt.

Vi har så en L2 switch hvor alt av maskiner er koblet til. Denne switchen er fysisk koblet til alle våre bare-metal servere.

Server2 Dette er vår NFS-server. Denne tilbyr lagring til hele systemet.

Vm host (server3) På denne maskinen kjøres 5 virtuelle maskiner hvorav tre er i bruk.

- Juju-controll-1: Dette er vår Juju-controller.
- K8s-worker-1: Dette er en av våre Kubernetes worker-nodes
- K8s-master-0: Dette er en av våre Kubernetes master-nodes

VM host (server4) På denne maskinen kjører vi 10 maskiner hvor alle er i bruk.

- Juju-HA1: Dette er en high-availability node for Juju.
- Juju-HA2: Dette er en high-availability node for Juju.
- K8s-load-balancer: Dette er en node som kjører Kubernetes-loadbalancer. Her blir trafikk i Kubernetes-clusteret optimalisert.
- etcd-2: Dette er en node som kjører Etcd for Kubernetes.
- etcd-1: Dette er en node som kjører Etcd for Kubernetes.
- easyrsa: På denne noden kjøres Easyrsa, som sikrer TLS for kommunikasjon mellom Kubernetesobjekter.
- Kubernetes-worker-2: Dette er en av våre Kubernetes worker-nodes.
- etcd-0: Dette er en node som kjører Etcd for Kubernetes.
- K8s-worker-0: Dette er en av våre Kubernetes worker-nodes.
- K8s-master-1: dette er en av våre Kubernetes master-nodes.

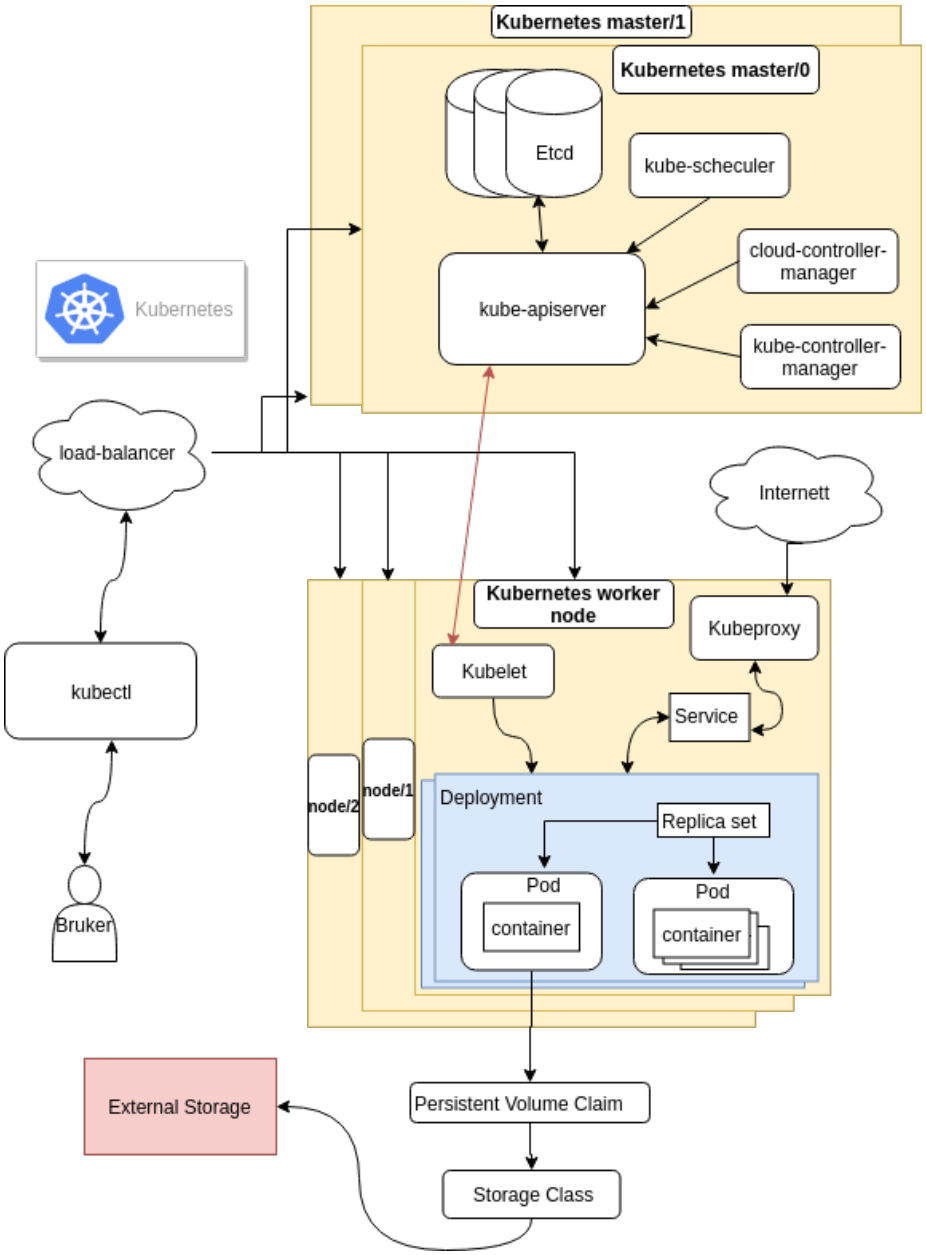


Figure 4: oversikt over Kubernetes

- **Cluster**
  Selve clusteret er hele systemet og består ev et sett med maskiner (kalles

noder), som kjører containerbaserte applikasjoner som blir styrt og vedlikeholdt av Kubernetes. Et cluster har flere worker-noder og minst en master-node (ofte fler for å oppnå High Availability)

- **Master-komponentene**
  Master node er ansvarlig for styringen av Kubernetes-clusteret. Dette er selve inngangspunktet til alle de administrative oppgavene Kuberentes tar seg av. Master noden tar seg også av orkestrering av worker-nodene, hvor de faktiske tjeneste kjører. Vi har to master-noder for å gi systemet High Availability (HA). La oss se nærmere på hver av master-komponentene som master-noden består av, og som gir clusterets kontrollpanel.

- **API server**
  Kompenent på masteren som eksponerer Kubernetes API-et. Det er front-end for Kubernetes kontrollpanelet. API-serveren er inngangspunkter for alle REST-kommandoene som brukes til å styre klassen. Den behandler REST-forespørsler, validerer dem og utfører den begrensede forretningslogikken. Resultatstatusen må fortsette et sted, og det bringer oss til neste komponent i hovedknutepunktet.

- **Etcd storage**
  etcd er en enkel, distribuert, konsistent API-server. Den brukes hovedsakelig til delt konfigurasjon og serviceoppdagelse. Den gir en REST API for CRUD-operasjoner, samt et grensesnitt for å registrere overvåkere på spesifikke noder, noe som gjør det mulig å varsle resten av klassen om konfigurasjonsendringer. Et eksempel på data lagret av Kubernetes i etcd er jobber som planlegges, opprettes og distribueres i pod-er, namespaces og replikasjonsinformasjon, osv.

- **EasyRSA**
  EasyRSA er en applikasjon som fungerer som en certificate authority og lager sertifikater for applikasjoner den er koblet opp mot. Applikasjonen er et kommandolinjeverktøy som lager og tar vare på Public Key Infrastructure (PKI). Dette systemet gjør at kommunikasjon i Kubernetes-clusteret bruker TLS-sertifikater. [16]

- **Scheduler**
  Selve deploymenten av konfigurerte pods og services på nodene skjer takket være scheduler-komponenten. Scheduleren har informasjon om ressursene som er tilgjengelige på medlemmene av clusteret, samt de som kreves for at den konfigurerte tjenesten skal kjøre, og dermed kan bestemme hvor en bestemt tjeneste skal distribueres. Faktorer tatt i betraktning for planlegging avgjørelser inkluderer individuelle og kollektive ressursbehov, maskinvare / programvare / policy begrensninger, affinitet og anti-affinitetsspesifikasjoner, datalokalitet, inter-arbeidsbelastning og tidsfrister.

- **Control-manager**
  Det er vanlig å kjøre flere typer kontrollere inni master-noden. Kontrollstyreren er en tjeneste som integrerer disse. Logisk sett er hver kontroller en separat prosess, men for å redusere kompleksiteten, er alle samlet og kjøres som en prosess. En kontroller bruker api-serveren til å se over statusen til det delte clusteret og gjør korrigerende endringer i gjeldende tilstand for å endre den til ønsket tilstand. Et eksempel på en slik kontroller er Replikasjonskontrolleren, som tar seg av antallet pods i systemet. Selve faktoren til replikasjonskontrolleren er konfigurert av brukeren, og det er kontrollerens ansvar å gjenopprette en mislykket pod eller fjerne en som ikke tas i bruk. Andre eksempler på kontrollere er endpoints controller, namespace controller og service accounts controller.

Node-komponentene
komponentene kjører på enhver node, opprettholder kjørende pod-er og vedlikeholder dem.

- **Container**
  Containere er en fleksibel og lettvekts plattform. Det bruker det underliggende operativsystemet slik at den som tar i bruk containeren bestemmer akkurat hva som skal kjøres på den. Ofte vanlig å kjøre en tjeneste per container. Den fungerer altså som en lightversjon av en virtuell maskin.

- **Docker**
  Det er Docker (som er en progrramvare-teknologi) som står bak denne virtualiseringen på operativsystemnivå, også kjent som containere

- **Kubelet**
  En agent som kjører på hver node i klusteret. Den sørger for at containere kjører i en pod. Kubelet tar et sett med PodSpecs (spesifikasjon av poddens ønskede oppførsel) som leveres gjennom ulike mekanismer og sikrer at beholderne beskrevet i disse PodSpecs-ene kjører og er healthy.

- **Pod**
  Det minste og enkleste Kubernetes-objektet. En Pod representerer et sett med en eller flere containere på klusteret.

- **Namespace**
  En funksjon fra Kubernetes som støtter flere virtuelle clustere på samme fysisk cluster. Namespaces brukes til å organisere objekter i et cluster og for å gi en måte å dele klusterressurser på.

- **Workload**
  Workloads er objekter du bruker til å styre, vedlikeholde og kjøre containere på klusteret. Kubernetes utfører distribusjonen og oppdaterer workloaden slik at den samsvarer med nåværende tilstanden av applikasjonen. Workload inkluderer DaemonSet, Deployments, Jobs, Pods, ReplicaSet, ReplicationController, og StatefulSet objekter.

29 Strategi og metode

29.1 Strategi

For å oppnå målene vi hadde satt for denne bacheloroppgaven har vi valgt en parallell arbeidsmetode som går ut på å jobbe med det tekniske apsektet ved oppgaven samtidig som vi skriver dokumentasjon. Dette har vi gjort ved å utføre teknisk arbeid, og deretter skrive ned relevant informasjon samt utarbeide dokumentere kilder.

29.2 Metode

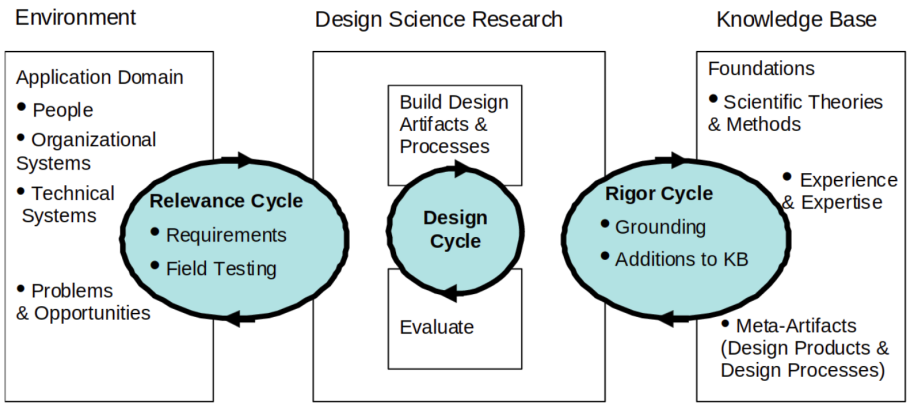


Figure 5: Three cycle view of design science research

I dette prosjektet har vi gått ut i fra Alan R. Hevners dokument kalt 'A Three Cycle View of Design Sience Research'. [19] Dette er en iterativ metode som innebærer at man skal bruke 3 sykluser innen designprosessen.

- Relevance Cycle
  Denne syklusen omhandler å finne et problem, og se etter en potensiell løsning. Her blir både informasjonssøk med vekt på kontekst av applikasjonen, og definisjoner av akseptkriterier satt opp. Dette gir en oversikt over hva som må gjøres, hva som er kritisk, og hva som kan ignoreres. Dette vil være en prosess hvor vi ser på allerede eksisterende system hos bedriften, og derfra utformer løsningsforslag. Dette vil gi oss det vi trenger av suksesskriterier, og systemkrav.

- Design Cycle
  Designsyklusen er hjertet i prosessen, da det er her man setter igang selve utviklingen av løsningen. Dette er en hurtig prosess hvor vi bygger et system, og etterpå evaluerer produktet. Dette blir gjennomført til vi kommer til et punkt vi er fornøyd med.

- Rigor Cycle
  I 'Rigor'-syklusen skal man tilegne seg informasjon og dokumentasjon om temaet man jobber med. Her finner man ekspertise, vitenskapelig litteratur, og diverse annen informasjon man trenger.

30 Løsningsbeskrivelse

Oversikt over vår løsning

Maskiner: 4x HP Z4 G4 Workstation

- Hovedkort: HP 81C5
- CPU: Intel Xeon W-2102
- RAM: 2x Hynix HMA84GR7AFR4N-VK 32GB ECC
- Nettverk: Intel Ethernet Connection (2) I219-LM
- Skjermkort: NVIDIA GP107GL
- Lagring: 2x Seagate ST4000NM0165 ATA
- Lagring: SAMSUNG MZVKW512HMJP-000H1 521GB

Nettverk:

- MAAS-Cluster: 11.0.0.0/8
- Management: 192.168.0.0/24
- Kubernetes / flannel: 10.0.0.0/24

Gateway	IP: 192.168.0.1	
Master Server	IP: 11.0.0.1	192.168.0.100
Juju controller	IP: 11.0.0.46	High Availability .47 .48
MAAS Server3	IP: 11.0.0.44	
MAAS Server4	IP: 11.0.0.21	
Kubernetes Master 0	IP: 11.0.0.65	
Kubernetes Master 1	IP: 11.0.0.64	
Kubernetes Load Balancer	IP: 11.0.0.62	
Easyrsa	IP: 11.0.0.61	
Etcd 0	IP: 11.0.0.59	
Etcd 1	IP: 11.0.0.66	
Etcd 2	IP: 11.0.0.60	
Kubernetes Worker 0	IP:11.0.0.63	
Kubernetes Worker 1	IP: 11.0.0.67	
Kubernetes Worker 2	IP: 11.0.0.58	
NFS-server	IP:11.0.0.77	

30.1 MAAS

Ubuntu MAAS er en opensource løsning laget av Canonical, designet for å orkestrere mengder av 'bare-metal'-maskiner. Dette leder til at man kan lett legge til maskiner til bruk i cluster, og redusere arbeidsmengden som skal til for å drifte clusteret.

30.1.1 Installasjon

Betingelser for installasjon av MAAS:

- Maskin som kan håndeteres via BMC [1]
- Ubuntu Server 18.04 LTS for nyeste versjon
- Tilgang til internett eller ferdig nedlastet software

For å starte installasjonen må man først legge til repository.

```
sudo apt-add-repository -yu ppa:maas/stable
```

Deretter installer maas

```
sudo apt install maas
```

Dette installerer både region og rack controller. Deretter bør man få tilgang på MAAS GUI. Da må man opprette et key-pair for brukeren som skal ha tilgang.

```
ssh-keygen -t rsa
```

Åpne en nettleser og fyll inn <MAAS server IP>:5240/MAAS. Her har man tilgang til det meste man skal trenge fra MAAS sin side.

Nettverk:
På MAAS har vi satt opp et clustersubnett (MAAS-cluster) som vi definerte til 11.0.0.0/8. På Dette nettet er MAAS satt opp som DHCP. Vi har også et subnett (192.168.0.0/24) som er ut mot ruter. På Master server har vi satt en av nettverksportene til å peke mot MAAS-cluster og en mot management. Dette fungerte best for oss, på grunn av vår ruter. På cluster-nettet er det viktig at man reserverer en dynamisk IP-range, da dette er påkrevd for å sette opp noder.[2]
Vi måtte også konfigurere brannmuren på region/rack-serveren til å aktivere ip forwarding mellom de to interne NIC-ene eno1 og eno2. Dette gjorde vi i iptables, kommandoene fant vi fra [2]. Dette fikk MAAS-serveren vår til å oppføre seg som router/firewall mellom de to interfacene for å tillate at maskiner som er koblet til det private interfacet vårt (eno1) får tilgang til Internett via det offentlige interfacet (eno2) som er koblet til ruteren vår.

På nodenes side:
Åpne BIOS/UEFI (obs, pass på å oppdatere firmware). Våre maskiner har Intel AMT som BMC, så dette slår vi på via UEFI. Her setter vi et passord og IP til

BMC slik at MAAS kan kontrollere maskinen. Deretter konfigurerer vi noden til PXE-boot slik at MAAS kan ta kontroll. Nå skal maskinene automatisk bli commissioned i MAAS. Ved deployment trengte våre maskiner en legacy-boot istedenfor UEFI, men dette er en bug.[3] Buggen sier at dersom man booter fra en disk som er større enn 2TB lagringskapasitet kan man støte på problemer. Nå skal maskinen starte opp og bli deployed.

30.1.2 Oversikt

På neste side kan du se MAAS-clusteret vårt i webgrensesnittet: Her kan man navigere seg rundt for å se hvordan selve clusteret er satt opp, hvilke maskiner vi har kjørende på clusteret, konfigurasjonen til rack- og region-kontrollerne vår, samt å se podsene vi bruker til virtualisering. Jeg skal rask gå igjennom de forskjellige delene av MAAS sitt webgrensesnitt vi har brukt gjennom bachelor-prosjektet.
Slik ser webgrensesnittet til MAAS ut. Øverst i url-en er IP-adressen til eno2-interfacet som går ut mot ruteren. Grensesnittet er meget brukervennlig og man kan lett navigere seg til de ulike konfigurasjonene vi opperer med. Her er vi i under machines, som viser oversikten over alle maskiner i MAAS-domenet, og statusen på disse. Vi ser nærmere på server3:

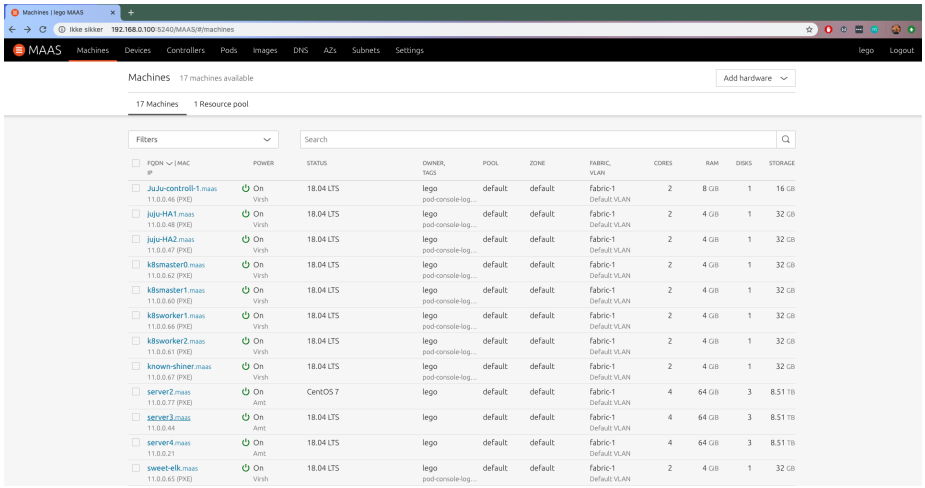


Figure 6: oversikt over maskinene i MAAS

Her gir MAAS oss informasjon om maskinen server3.

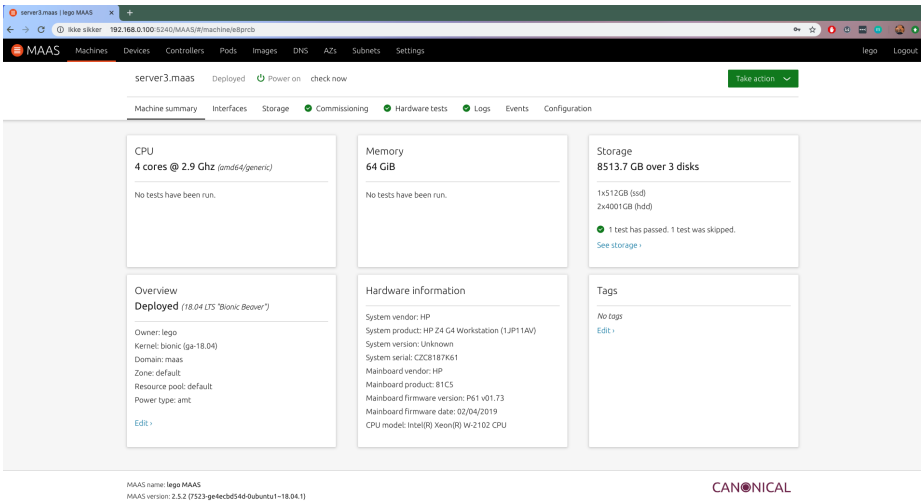


Figure 7: oversikt over server3

Server3 sitt interface:

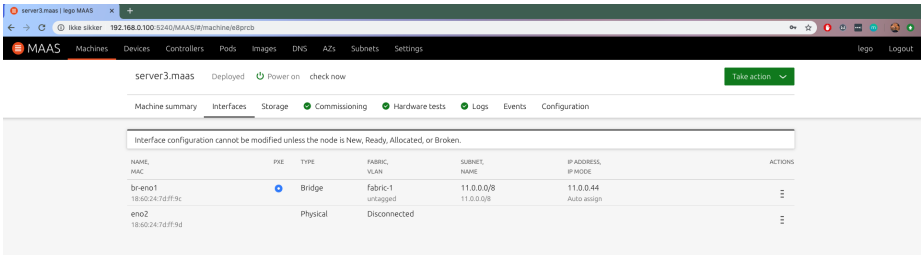


Figure 8: oversikt over interfacet til server3

Konfigurasjonsinnstillingene til Server3, Intel AMT er vår BMC som gjør at vi kan få kontakt med serveren over Internett. Vi konfigurerte det i BIOS til serveren og må ha passord for å autentisere.

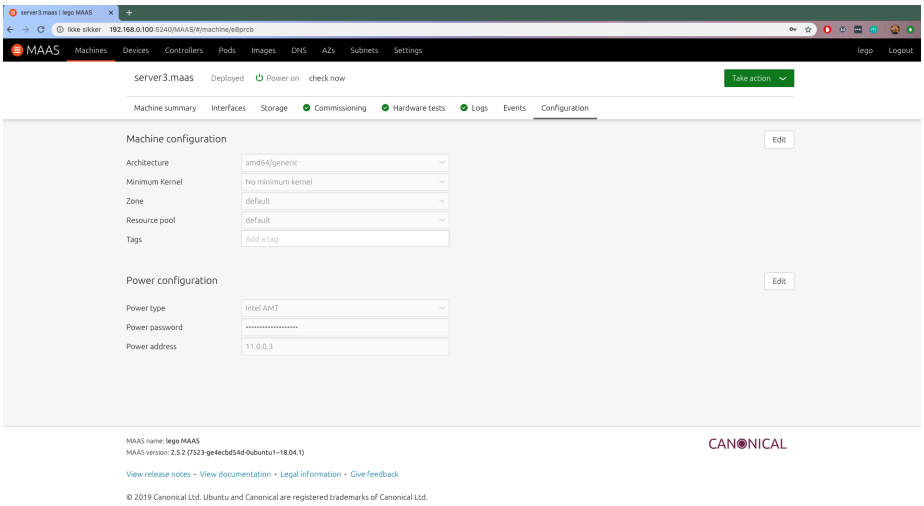


Figure 9: oversikt over konfigurasjonen til server3

Vi har dedikert Server2 og Server3 til virtuelle maskiner.

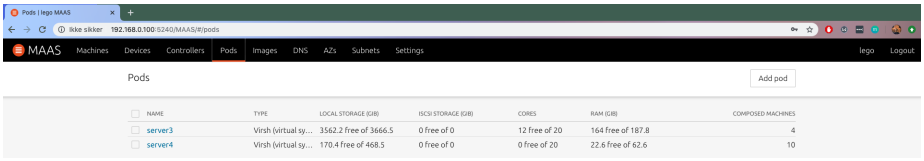


Figure 10: oversikt over podsene vi har laget i MAAS

Her ser vi oversikt over hvor mye Server3 har distribuert av sin datakraft og hvilke virtuelle maskiner den har laget.

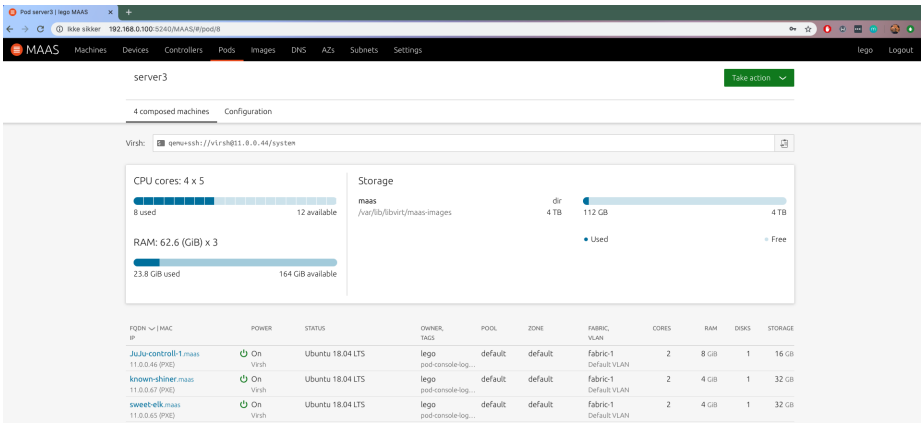


Figure 11: Oversikt over podden Server3 og dens virtuelle maskiner

Rack- og region-kontrolleren sin oversiktside.

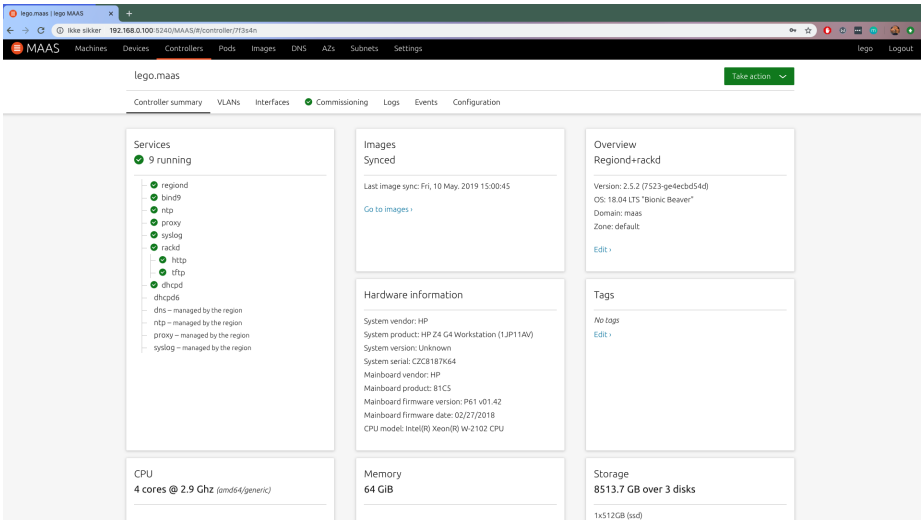


Figure 12: oversikt over region og rack-kontrolleren

Her ser vi hvilke public keys vi har lagt til i MAAS. Disse kan ssh-et seg inn til enhver maskin som er koblet til MAAS.

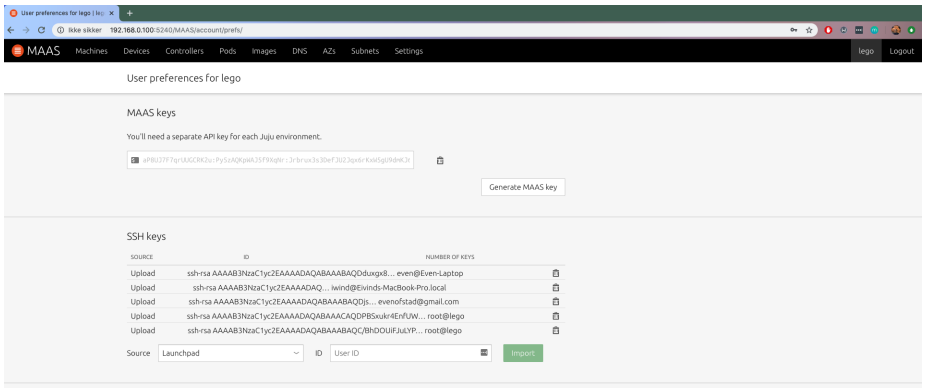


Figure 13: oversikt over bruker lego sine ssh tilkoblinger

30.2 Juju

Vi har tatt i bruk verktøyet Juju, da det er et verktøy som gjør det mye lettere å sette opp kompliserte systemer som kan ta veldig lang tid å konfigurere rett. Juju laster ned og installerer alt som trengs for de systemene man vil installere, samt oppretter koblingene mellom de forskjellige delene som blir satt opp. Juju bruker automatisk ressursene som er tilgjengelig i skyen den er tildelt for å deploye de systemene man installerer. For å se en oversikt over skyen kan man bruke kommandoen 'juju status'.

30.2.1 Installasjon

Siden ubuntu 14.04 LTS er det anbefalt å installere Juju via snaps. Denne kan installeres på samme server som MAAS.

```
sudo apt install snapd
```

```
sudo snap install juju --classic
juju add-cloud
```

under add-cloud må vi velge MAAS, gi navn til skyen (vi har valgt MAAS-cloud1). Legg deretter inn API endpoint url. Dette er <MAAS server IP>:5240/MAAS/ legg så til credentials:

```
juju add-credential maas-cloud1
```

auth-type her er automatisk, og MAAS-oauth er tilsvarende API-nøkkelen man finner på MAAS-webgrensesnittet under <bruker>. For å lage en Juju kontroller må man først opprette den. Her bruker vi Juju bootstrap.

```
juju bootstrap <sky-navn> --to <MAAS-maskin>.<domene>
```

Vårt skynavn er MAAS-cloud1, maskinen vår er kalt Juju-controll-1, og domenet som brukes er MAAS. Dette fører til at vår kommando ble som følger:

```
juju bootstrap maas-cloud1 --to Juju-controll-1.MAAS
```

Nå skal man kunne bruke Juju.
OBS! Vi bootet til en 512GB nvme disk, men hvis disken man skal boote til er større enn 2TB kan man støte på en kjent bug.[20] Denne buggen er fra 2016 og skal være fikset, men vi møtte på den i 2019.

30.2.2 Oversikt

I vårt Juju-miljø har vi tre redundante kontrollermaskiner som holder styr på systemet. Dette gjør at Juju ikke henger på en spesifikk node. Administratorbrukeren for Juju er root på Region/rack controller, bare fra denne brukeren kan man kontrollere MAAS-cloud1. For å kunne se alt som foregår på MAAS-cloud1 kan man bruke kommandoen:

```
juju status
```

Da får man dette:

```
root@lego:~# juju status
Model  Controller   Cloud/Region  Version  SLA          Timestamp
k8s    maas-cloud1  maas-cloud1   2.5.3    unsupported  14:35:36+02:00

App                    Version  Status  Scale  Charm                  Store       Rev  OS      Notes
easyrsa                3.0.1    active      1  easyrsa                jujucharms  231  ubuntu
etcd                   3.2.10   active      3  etcd                   jujucharms  411  ubuntu
flannel                0.10.0   active      5  flannel                jujucharms  398  ubuntu
kubeapi-load-balancer  1.14.0   active      1  kubeapi-load-balancer  jujucharms  614  ubuntu  exposed
kubernetes-master      1.14.1   active      2  kubernetes-master      jujucharms  638  ubuntu
kubernetes-worker      1.14.1   active      3  kubernetes-worker      jujucharms  504  ubuntu  exposed

Unit                      Workload  Agent  Machine  Public address  Ports           Message
easyrsa/0*                active    idle   9        11.0.0.61                       Certificate Authority connected.
etcd/0                    active    idle   1        11.0.0.59       2379/tcp        Healthy with 3 known peers
etcd/1                    active    idle   4        11.0.0.66       2379/tcp        Healthy with 3 known peers
etcd/2*                   active    idle   7        11.0.0.60       2379/tcp        Healthy with 3 known peers
kubeapi-load-balancer/0*  active    idle   0        11.0.0.62       443/tcp         Loadbalancer ready.
kubernetes-master/0*      active    idle   5        11.0.0.65       6443/tcp        Kubernetes master running.
  flannel/0*              active    idle            11.0.0.65                       Flannel subnet 10.1.28.1/24
kubernetes-master/1       active    idle   3        11.0.0.64       6443/tcp        Kubernetes master running.
  flannel/4               active    idle            11.0.0.64                       Flannel subnet 10.1.48.1/24
kubernetes-worker/0       active    idle   8        11.0.0.63       80/tcp,443/tcp  Kubernetes worker running.
  flannel/3               active    idle            11.0.0.63                       Flannel subnet 10.1.93.1/24
kubernetes-worker/1*      active    idle   2        11.0.0.67       80/tcp,443/tcp  Kubernetes worker running.
  flannel/1               active    idle            11.0.0.67                       Flannel subnet 10.1.16.1/24
kubernetes-worker/2       active    idle   6        11.0.0.58       80/tcp,443/tcp  Kubernetes worker running.
  flannel/2               active    idle            11.0.0.58                       Flannel subnet 10.1.4.1/24

Machine  State    DNS         Inst id        Series  AZ       Message
0        started  11.0.0.62   k8smaster0     bionic  default  Deployed
1        started  11.0.0.59   virt05         bionic  default  Deployed
2        started  11.0.0.67   known-shiner   bionic  default  Deployed
3        started  11.0.0.64   virt09         bionic  default  Deployed
4        started  11.0.0.66   k8sworker1     bionic  default  Deployed
5        started  11.0.0.65   sweet-elk      bionic  default  Deployed
6        started  11.0.0.58   virt03         bionic  default  Deployed
7        started  11.0.0.60   k8smaster1     bionic  default  Deployed
8        started  11.0.0.63   virt08         bionic  default  Deployed
9        started  11.0.0.61   k8sworker2     bionic  default  Deployed
```

Figure 14: juju status

30.3 Ansible

Ansible er et enkelt lesbart automatiseringsverktøy som automatiserer ressursdelegering i sky, konfigurasjons administrasjon, applikasjonsutplassering, orkestrering av komplekse oppgaver, distribuering av programvare og mange andre IT-behov. Alt dette kan Ansible automatisere på så mange noder og maskiner man vil. Disse maskinene må dog være koblet til serveren Ansible kjører på via SSH. Ansible har et stort fokus på enkelhet og bruker derfor et veldig simpelt språk kalt YAML. YAML brukes i Ansible Playbooks, som er filene man kjører i Ansible. Ved å ha en slik simpel tilnærming blir det enkelt å forstå hva som skal gjøres i hver playbook. YAML vil bare det absolutt nødvendige, noe som også gjør at oppsettet i filen skal være perfekt. Dette har resultert i mye feilmeldinger hvor man deretter prøver å finne hvor i filen man har feilet. Det kan være smart å bruke 'yamllint.com' samtidig for å sjekke om bruken av YAML er korrekt. Ansible fungerer ved at den kobler seg til våre noder og kjører ut små programmer kalt Ansible modules til dem. Disse programmene er skrevet for å være ressursmodeller av ønsket tilstand i systemet. Ansible utfører da disse modellene (over SSH), og fjerner dem når de er ferdige. Root login er ikke nødvendig, man kan bruke hvilken bruker man vil på serveren, men må da spesifisere hvilken bruker som har sudo-tilgang i konfigurasjonsfilene til Ansible. Som standard bruker Ansible en enkel tekstfil (kalt hosts) for å representere hvilke maskiner den administrerer. Maskinene kan legges inn i egne grupper for lettere å rulle ut lik informasjon til dem som trenger det. Når hosts-filen er oppført, kan variabler tilordnes dem enkelt ved å kalle på dem i YAML-filene ved å skrive hosts: <node-gruppe>. Ellers er det to måter man kan skrive en Ansible playbook på, enten kan man skrive rett inn i playbook-en hva man vil gjøre under 'tasks:', eller så kan man lage ulike roles som beskriver en enkel task, og kalle på disse med 'roles:' og rollene man har laget. [15]

30.3.1 Installasjon

Selve installasjonen av Ansible var rett frem og enkel da den besto av noen enkle kommandoer. Starte med å oppdatere:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Installerer så de nødvendige repository-ene Ansible trenger:

```
sudo apt-add-repository ppa:ansible/ansible
```

Må oppdatere igjen:

```
sudo apt-get update
```

Installerer så Ansible:

```
sudo apt-get install ansible -y
```

Må også installere python dersom dette ikke er installert:

```
sudo apt-get install python -y
```

Man må nå koble Ansible til nodene/maskinene den skal ta seg av. Dette gjøres via SSH. Det må altså være en ssh-forbindelse mellom serveren Ansible er installert på og alle nodene den skal ha kontakt med. For å gjøre dette måtte man opprette et nøkkelpar med kommandoen SSH-keygen. Når de spør om passphrase trykker man enter. Man må så overføre den offentlige nøkkelen til alle noder i domenet. Her brukte jeg et enkelt script for å gjøre det til en automatisert prosess:

```
\textit{ for host in master.example.com \
    node1.example.com \
    node2.example.com; \
    do ssh-copy-id -i ~/.ssh/id_rsa.pub \$host; \
    done }
```

[4]
Når alle maskiner har fått SSH kobling til Ansible-serveren, må man installere python på alle nodene. Dette gjøres enten ved å gå på hver node og installere python på dem. Eller så kan vi la Ansible gjøre det på alle maskinene samtidig i en playbook, pythoninstall.yml:

```
- name: install python2 on all instances
  hosts: all
  gather_facts: false
  tasks:
    - name: run apt-get update and install python
      raw: "{{ item }}"
      loop:
        - sudo apt-get update
        - sudo apt-get -y install python
      become: true
      ignore_errors: true
```

[6] Kjører deretter denne med 'ansible-playbook pythoninstall.yml.

30.3.2 Oversikt

Nå som vi har installert alt Ansible trenger for å kjøre automatiserte prosesser til nodene i domenet, må vi konfigurere Ansible til å bli kjent med nodene og gruppere dem slik at utkjøringen blir mer presis. Vi oppretter en fil 'hosts' under etc/ansible/hosts og konfigurer denne til å ta med alle nodene i domenet og grupperer dem:

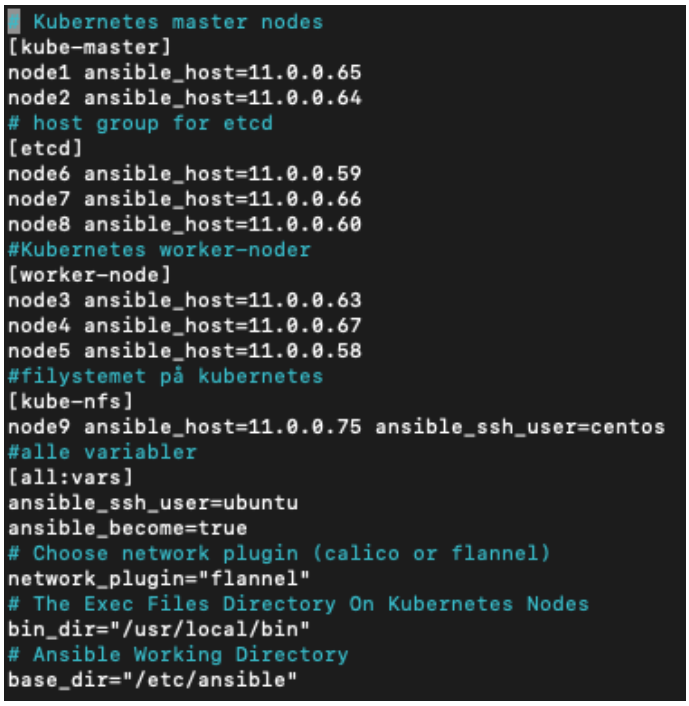


Figure 15: nodene i Ansible

For å SSH-et inn til MAAS-maskiner må man bruke navnet på operativsystemet som brukernavn istedenfor root. Vi er nødt til å si til Ansible at SSH-brukeren skal være ubuntu på alle utenom 'kube-nfs'-maskinen som bruker centos som operativsystem og må derfor ha centos som SSH-bruker. Når man har satt opp konfigurasjonsfilen 'hosts' som har oversikt over alle hoster i systemet må vi konfigurere selve konfigurasjonsfilen (/etc/ansible/config) til Ansible slik at vi kan fortelle Ansible hva den skal bruke av filer. For eksempel setter vi inventory = /etc/ansible/hosts for at Ansible skal bruke hosts-filen, og setter roles = /etc/ansible/roles for at Ansible skal vite hvor rollene den skal bruke er.

30.3.3 Roles

Roller gir et rammeverk for uavhengige eller gjensidig avhengige samlinger av oppgaver, variabler, filer, moduler og maler. I Ansible er rollen den primære mekanismen for å bryte en playbook i flere filer. Dette forenkler jobben å skrive komplekse playbook-er, samt gjør dem lettere å gjenbruke til andre playbook-er, da man bryter playbook-en inn i logiske deler. Selve rollene er i utgangspunktet begrenset til en spesifikk funksjonalitet eller ønsket output. Alle roller har med de nødvendige tiltakene som trengs for å gi ønsket resultat, enten innenfor rollen selv, eller i andre roller som er oppført som avhengigheter. Det er viktig å

notere seg at roller ikker er playbooks. Det er ingen måte å direkte kjøre en rolle. Roller er en liten funksjonalitet som kan brukes uavhengig av hverandre, men må brukes i playbooks. Den har heller ingen eksplisitt innstilling for hvilken host rollen skal gjelde for.

30.3.4 Ansible eksempel

For eksempel kan du ha en rolle som oppretter en mappe, en annen rolle som installerer nfs-common og en tredje rolle som mounter nodene til den opprettete mappen. Dette eksempelet skal jeg gå igjennom for å vise hvordan Ansible kan brukes. Først ser vi den ene maskinen vi skal utføre oppgaven på (virt09) at den ikke er mountet til noe NFS-filsystem.

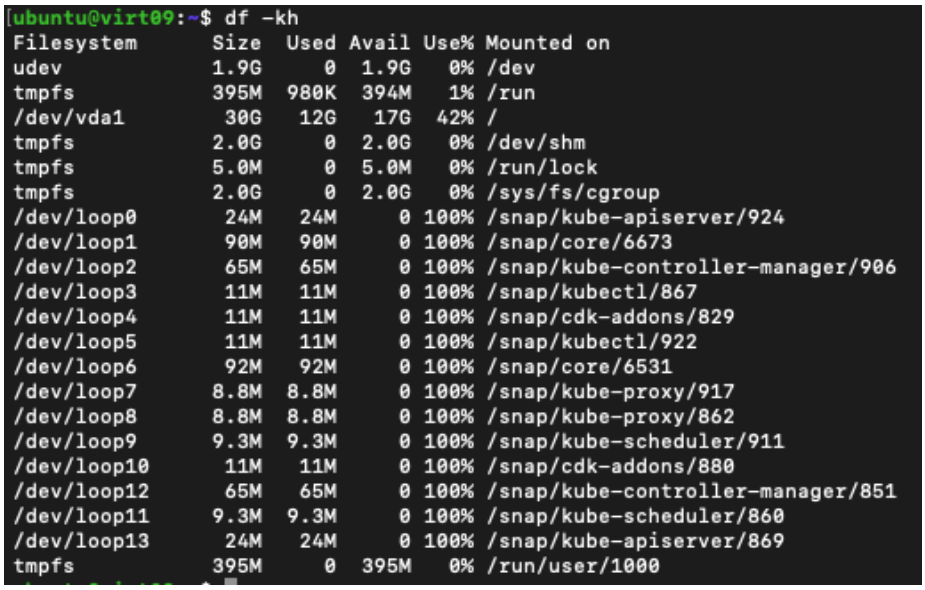


Figure 16: node virt09 unmountet

Vi skal lage tre roller som alle hver for seg skal gjøre en spesifikk oppgave og dereter kjøre disse rollene i en playbook. Oppretter rollen install-nfs under /etc/ansible/roles/ubuntu/install-nfs/tasks/main.yml som skal installere nfs-common på nodene:

```
---
#Install nfs-common

-
  apt:
    name: nfs-common
    state: present
    update_cache: true
  name: "Install nfs-common"
  tags:
    - nfs
```

Figure 17: Rolle som installerer nfs-common

Opretter så rollen createdir som ligger i /etc/ansible/roles/ubuntu/createdir/tasks/main.yml. Denne rollen oppretter hele mappen nfsshare og hele stien /mnt/nfs/var/nfsshare. Om den ikke finnes fra før da vi har med recurse: yes.

```
---
#Opprette mappen nfsshare
 - name: create the directory
   file:
     path: /mnt/nfs/var/nfsshare
     state: directory
     recurse: yes
   tags:
     - nfs
```

Figure 18: Rolle som installerer mappen det skal mountes på

Med rollen "mount" mounter man nodene til den opprettet mappen, samt legge til en linje i filen /etc/fstab for å permanent mounte noden.

```
---
#Mounte mappen nfs-mappen og konfigurere /etc/fstab på klientene

- name: mount the directory
  action: mount name=/mnt/nfs/var/nfsshare/ src=11.0.0.75:/var/nfsshare fstype=nfs state=mounted
  tags:
    - nfs
-
  shell: echo '11.0.0.75:/var/nfsshare    /mnt/nfs/var/nfsshare   nfs defaults 0 0' >> /etc/fstab
  name: "configure /etc/fstab on clients"
  tags:
    - nfs
```

Figure 19: Rolle som mounter og legger til en linje i en fil

For å kjøre disse rollene oppretter vi en playbook "nfs.yml". Denne skal fortelle hvilke noder dette skal kjøre på samt hvilke roller som skal være med:

```
---
#Installere nfs-common og mounting
- hosts: kube-master
  vars:
    http_port: 80
    max_clients: 200
  roles:
   - install-nfs
   - createdir
   - mount
```

Figure 20: Playbook som kjører rollene

Kjører så playbooken med ansible-playbook nfs.yml:

```
root@lego:/etc/ansible# ansible-playbook nfs.yml

PLAY [kube-master] *************************************************************

TASK [Gathering Facts] *********************************************************
Wednesday 08 May 2019  14:06:01 +0200 (0:00:00.043)       0:00:00.043 *********
ok: [node2]
ok: [node1]

TASK [install-nfs : Install nfs-common] ****************************************
Wednesday 08 May 2019  14:06:03 +0200 (0:00:02.345)       0:00:02.389 *********
ok: [node1]
ok: [node2]

TASK [createdir : create the directory] ****************************************
Wednesday 08 May 2019  14:06:08 +0200 (0:00:04.775)       0:00:07.164 *********
ok: [node1]
ok: [node2]

TASK [mount : mount the directory] *********************************************
Wednesday 08 May 2019  14:06:09 +0200 (0:00:00.429)       0:00:07.593 *********
ok: [node2]
ok: [node1]

TASK [mount : configure /etc/fstab on clients] *********************************
Wednesday 08 May 2019  14:06:09 +0200 (0:00:00.417)       0:00:08.011 *********
changed: [node1]
changed: [node2]

PLAY RECAP *********************************************************************
node1                      : ok=5    changed=1    unreachable=0    failed=0
node2                      : ok=5    changed=1    unreachable=0    failed=0

Wednesday 08 May 2019  14:06:09 +0200 (0:00:00.374)       0:00:08.385 *********
===============================================================================
install-nfs : Install nfs-common ---------------------------------------- 2.35s
Gathering Facts --------------------------------------------------------- 2.39s
createdir : create the directory ---------------------------------------- 4.78s
mount : mount the directory --------------------------------------------- 0.43s
mount : configure /etc/fstab on clients --------------------------------- 0.42s
root@lego:/etc/ansible#
```

Figure 21: Kjører playbooken

Ansible kjører og gir en tilbakemelding for hvert steg, her får nodene status 'ok' på de fire første taskene og 'changed' på siste task. Forskjellen på disse er at 'ok' forteller at oppgaven er utført på node, mens 'changed' forteller at noe er forandret på maskinen, i dette tilfellet forandret vi på filen /etc/fstab.
Ser på virt09 at den har mountet NFS-filsystemet:

```
ubuntu@virt09:~$ df -kh
Filesystem               Size  Used Avail Use% Mounted on
udev                     1.9G     0  1.9G   0% /dev
tmpfs                    395M  980K  394M   1% /run
/dev/vda1                 30G   12G   17G  42% /
tmpfs                    2.0G     0  2.0G   0% /dev/shm
tmpfs                    5.0M     0  5.0M   0% /run/lock
tmpfs                    2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/loop0                24M   24M     0 100% /snap/kube-apiserver/924
/dev/loop1                90M   90M     0 100% /snap/core/6673
/dev/loop2                65M   65M     0 100% /snap/kube-controller-manager/906
/dev/loop3                11M   11M     0 100% /snap/kubectl/867
/dev/loop4                11M   11M     0 100% /snap/cdk-addons/829
/dev/loop5                11M   11M     0 100% /snap/kubectl/922
/dev/loop6                92M   92M     0 100% /snap/core/6531
/dev/loop7               8.8M  8.8M     0 100% /snap/kube-proxy/917
/dev/loop8               8.8M  8.8M     0 100% /snap/kube-proxy/862
/dev/loop9               9.3M  9.3M     0 100% /snap/kube-scheduler/911
/dev/loop10               11M   11M     0 100% /snap/cdk-addons/880
/dev/loop12               65M   65M     0 100% /snap/kube-controller-manager/851
/dev/loop11              9.3M  9.3M     0 100% /snap/kube-scheduler/860
/dev/loop13               24M   24M     0 100% /snap/kube-apiserver/869
11.0.0.75:/var/nfsshare  2.8T  9.5G  2.7T   1% /mnt/nfs/var/nfsshare
tmpfs                    395M     0  395M   0% /run/user/1000
ubuntu@virt09:~$
```

Figure 22: Noden mountet mappen med nfs-filsystemet

30.4 Kubernetes

Kubernetes er et containerorkestrasjonsverktøy som håndterer deployments, versjonsoppdateringer, og high availability på diverse tjenester man velger å kjøre.

30.4.1 Installasjon

For vår installasjon av Kubernetes brukte vi Juju for å enkelt sette opp et fungerende cluster. Juju har en pakke som er klar til å installeres.[5] Denne pakken oppretter hele clusteret og alle avhengighetene dette clusteret har for å fungere. Dette inkluderer worker node, master node, load-balancer, flannel, easyrsa, og etcd.

For å installere den pakken trenger man 10 maskiner. Disse må ha en minimum av 2 cpukjerner, 4GB Ram, og 16GB lagringsplass. Vi hadde ikke 10 maskiner tilgjengelig, så vi opprettet virtuelle maskiner for å oppnå dette kravet. Dette fungerer bra for vårt oppsett, men worker nodene bør settes opp på egne fysiske maskiner, da de kan utnytte maskinkraften optimalt via opprettelse og flytting av pods.

For å starte installasjonen vil man åpne Jujus GUI. dette kan enkelt gjøres ved å bruke kommandoen:

```
Juju gui
```

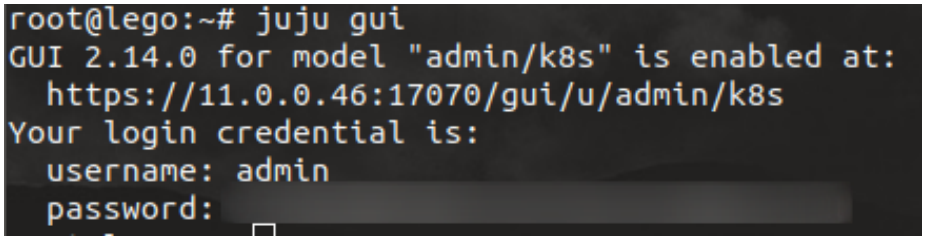


Figure 23: Juju gui kommando

Dette er resultatet av kommandoen.
Dette bringer oss til:

Herfra kan man legge til "charms" som man vil. Ved å trykke på "+" vil man få opp en meny hvor man kan finne de pakkene som passer.

Dette er charmen vi valgte, da den har alt som trengs for å kjøre et Kubernetes-cluster.

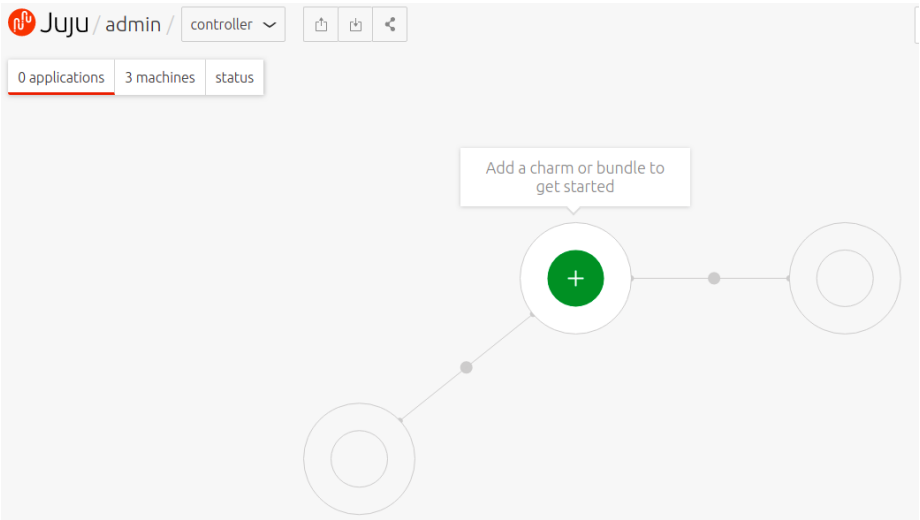


Figure 24: Juju gui

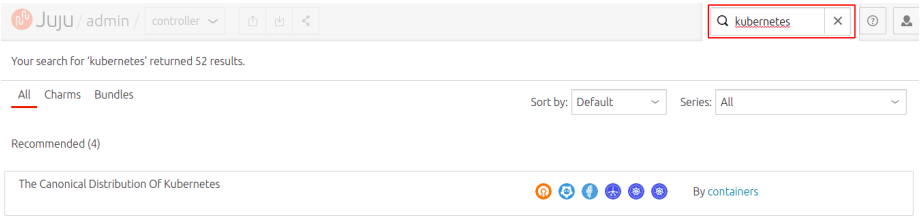


Figure 25: Juju kubernetes charm

Slik ser det ut på Gui når charmen er installert.
Obs! Dette kan ta en god stund.
Så må man koble sin bruker opp mot Kubernetes clusteret.[7]

```
sudo cp /etc/kubernetes/admin.conf $HOME/
sudo chown $(id -u):$(id -g) $HOME/admin.conf
export KUBECONFIG=$HOME/admin.conf
```

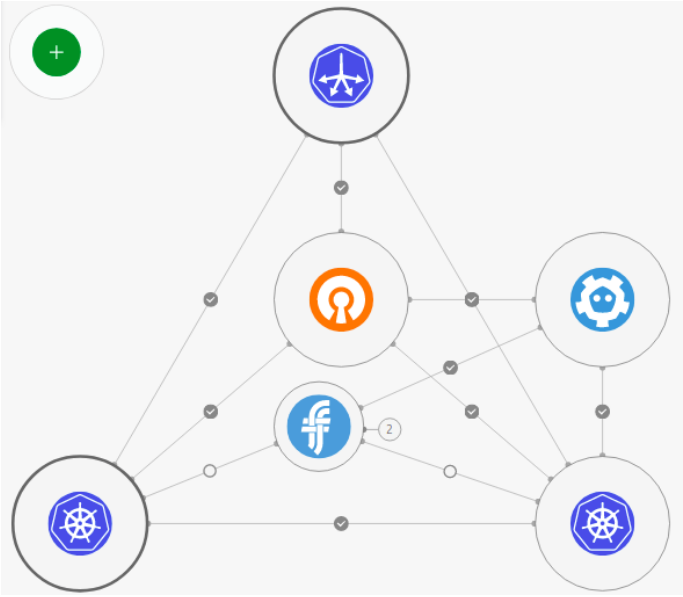


Figure 26: Juju kubernetes charm gui

30.4.2 Kubernetes Gui

For å få tilgang til Kubernetes sitt dashboard fra arbeidsmaskin må man ha tilgang til en nettleser og installere kubectl.

Mac:

```
brew install kubectl
```

Ubuntu:

```
snap install kubectl --classic
```

Deretter bør man opprette en mappe, og kopiere relevant data fra server:

```
mkdir -p ~/.kube
scp <brukernavn@<server-ip>:~/.kube/config ~/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Da kan man kjøre

```
kubectl proxy
```

Så kan man kopiere følgende inn i nettleser:

```
http://localhost:8001/api/v1/namespaces/kube-system/
services/https:kubernetes-dashboard:/proxy/#!/node/virt02?namespace=default
```

Logg så inn ved å finne den filen man kopierte.

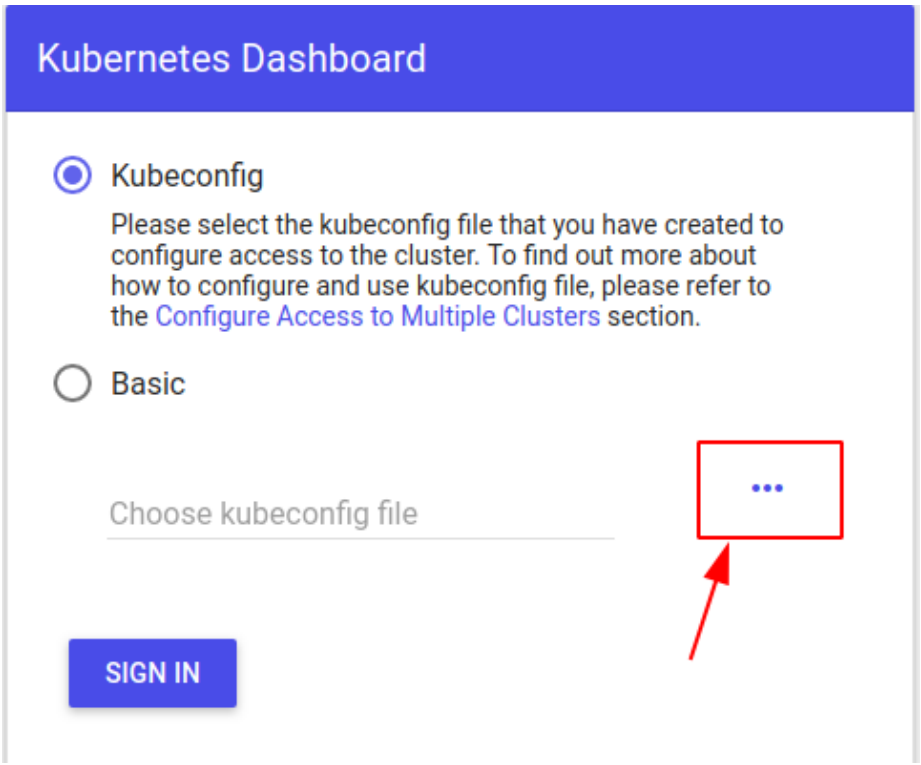


Figure 27: kubernetes dashboard login

30.4.3 Oversikt

I vårt Kubernetes-system har vi to relevante namespaces: Default, og kube-system. Default er der vi har ikke-kritiske deployments kjørende, og i kube-system kjøres det som brukes i Kubernetes sin infrastruktur. Vi bruker en PVC laget via filen 4-pvc-nfs.yaml. Her har vi gitt navn pvcX til de forskjellige pvc-ene vi har opprettet. Hvis man vil lage en ny pvc som skal kobles til NFS, er det viktigste at den bruker storageClass: managed-nfs-storage. Da peker pvc-en til korrekt mappe. Alle deploys som trenger persistent storage bør bruke en pvc som peker mot NFS, da dette er det vi har satt opp av langtids-lagring for Kubernetes.

30.5 Network File System

Network File System er en filsystemprotokoll for nettverk som gir tilgang til filer fra disker på et nettverk på samme måte som om de var installert lokalt på klientmaskinen. Dette var optimalt for vår løsning, da dette kan brukes for Kubernetes' lagringsbehov over flere noder. Vi har opprettet en NFS-server som kjører på Centos7 og som alle Kubernetes-nodene har fått tilgang til.

30.5.1 Installasjon

Til å starte med opprettet vi en virtuell maskin med CentOS 7 som operativsystem.[8] Første steg er å installere serveren.

Serverkonfigurasjon

Start med å installere nfs-utils.

```
yum install nfs-utils
```

Lag så en mappe som skal deles og sett rettigheter

```
mkdir /var/nfsshare
chmod -R 755 /var/nfsshare
chown nfsnobody:nfsnobody /var/nfsshare
```

Her bruker vi /var/nfsshare som en delt mappe. Vi oppretter denne fordi bruk av mange andre mapper kan gi store tilgangsproblemer, og dermed ødelelegge hele hierarkiet.

Så må man starte opp tjenestene som trengs.

```
systemctl enable rpcbind
systemctl enable nfs-server
systemctl enable nfs-lock
systemctl enable nfs-idmap
systemctl start rpcbind
systemctl start nfs-server
systemctl start nfs-lock
systemctl start nfs-idmap
```

Deretter må vi eksportere rett mappe

```
nano /etc/exports
```

Her legger vi til[9]

```
/var/nfsshare *(rw,sync,no_root_squash,no_all_squash,insecure)
```

Dette betyr at alle IPadresser kan få tilgang til mounting hos NFS serveren. Siden dette er bak vår regioncontroller er vi ikke bekymret for sikkerheten ved dette tiltaket. "insecure" parameteren viser seg å være viktig for å bruke NFS som volume for Kubernetes.

Så for å gjøre oss ferdig på serversiden:

```
systemctl restart nfs-server
firewall-cmd --permanent --zone=public --add-service=nfs
firewall-cmd --permanent --zone=public --add-service=mountd
firewall-cmd --permanent --zone=public --add-service=rpc-bind
firewall-cmd --reload
```

Klientkonfigurasjon

På nodene som skal ha tilgang må man installere nfs-common og mounte mappen.

```
sudo apt-get install nfs-common
mkdir -p /mnt/nfs/var/nfsshare
mount -t nfs <NFSserver-ip>:/var/nfsshare /mnt/nfs/var/nfsshare/
```

<NFSserver-ip> er i vårt tilfelle 11.0.0.77 Man kan også teste om dette var suksessfullt.

```
df -kh
```

For å permanent mounte må man også legge inn detaljene i /etc/fstab

```
echo "11.0.0.77:/var/nfsshare /mnt/nfs/var/nfsshare nfs defaults 0 0" >>
    /etc/fstab
```

Volumes på Kubernetes

For at kubernetes nå skal kunne bruke NFS, må man opprette en storage class, og en persistent volume claim.

Storage class[10]

En Storage Class gir administratorer muligheten til å beskrive forskjellige typer lagring. De forskjellige typene kan kartlegge forskjellige kvalitetsgrader, eller til backup policies, eller til egendefinerte policies satt opp av administrator. Kubernetes er agnostisk til hva disse klassene representerer.

Persistent Volume Claim

Pods kan bruke Persistent Volume Claims som Persistent Volumes. Dette fører til at man kan få kubernetes til å opprette persistent volumes etter behov.

For å sette opp en NFS provider på clusteret må man opprette en service account. Dette gjør vi via filen rbac.yaml.[11] Denne kan vi deploye til kubernetes, og vil sette opp autorisasjon til de komponentene som trenger det.

Deretter må vi opprette en storage class, her bruker vi class.yaml.[12] Videre må vi sette opp en provisioner for volumene. Her brukes deployment.yaml.[13] I denne må vi bytte ut $<< NFSServerIP >>$ med vår NFS server IP(11.0.0.77), og "/srv/nfs/kubedata" med vår mount path hos NFS serveren (/var/nfsshare).

Til slutt må vi opprette et persistent volume claim. Dette kan brukes til å automatisk opprette volumes etter som kubernetes pods trenger det.[14] I denne filen vil vi forandre "storageClassName" fra "manual" til "managed-nfs-storage". Dette er navnet på storage classen vi opprettet tidligere. Vi kan også forandre navn til hva vi vil.

30.5.2 Oversikt

Dette har gjort at vi kan bruke en NFS-server for lagring av data. Alle deployments må bruke de persistent volume-claims vi har opprettet for bruk ved NFS for å kunne kjøre normalt.

31 Sluttrapport

32 Forord

Vi har gått tre år på studiet Informatikk, drift av datasystemer. Som navnet så fint beskriver går studiet i stor grad på å lære seg ulike datasystemer, hvordan de fungerer, kode for å automatisere prosesser og hvordan man drifter og vedlikeholder disse. Vi har også hatt ulike programmeringsfag, men mer en introduksjon til disse enn spesialisering. Selve studieretningen Informatikk er meget aktuell i dagens samfunn da Internett har blitt en avhengighet for menneskeheten. For det er nemlig det informatikk, eller informasjonsteknologi handler om - Internett, IT og alt rundt det.
Hensikten med prosjektet har vært å prøve oss i jobblivet på et reelt system som skal brukes i produksjon av SINTEF fremover. Det å jobbe i enn bedrift fire dager i uken har vært veldig forskjellig fra det vanlige studieliv. Det er en helt annen jobbkultur i bedriften enn hva vi er vant med fra studiet på Kalvskinnet. Det gjør at vi jobber mer målrettet og fokuserer mer på oppgaven. Faglig er dette et steg opp fra systemer vi har jobbet med tidligere på studiet, noe en bacheloroppgave skal være. Prosjektet er veldig relevant da systemet tar for seg hvordan en sky kan bli opprettet med bare-metal, hvordan man kan automatisere oppgaver over flere maskiner og hvordan automatisering av utrulling, skalering og styring av containeriserte applikasjoner fungerer og hvordan dette blir satt opp.

33 Oppgavebeskrivelse

I denne oppgaven skulle vi opprette en lokal skyløsning for SINTEF, ved hjelp av verktøy som Ubuntu MAAS, Juju, Ansible, og Kubernetes. Dette skal brukes til å kjøre diverse workloads som trenger skalerbar maskinkraft. Henskiten med prosjektet var å redusere arbeidet SINTEF delegerer til drift av diverse servere de har stående til dags dato.

Vi skulle i denne oppgaven sette opp løsningen fra bunn av, med fire HP workstations beskrevet i driftsdokumentet. Vi måtte jobbe med alt fra firmware, og pakkeforwarding, til nettverksdesign og containerorkestrering.

Oppdragsgiveren i denne oppgaven var SINTEF, hvor vår kontaktperson var Peter Haro. Hos NTNU var Tor Ivar Melling vår veileder.

34 Hvordan ble oppgaven løst

I forhold til hvert av punktene nedenfor kan du ta med noen ord om eventuelle problemer som har oppstått, og hvordan disse i tilfelle ble løst, dersom dette ikke er dekket av annen dokumentasjon.

- Metoder og standarder som ble brukt
  De eneste standardene som har blitt tatt i bruk, er standardene knyttet til dokumentasjonen vi har skrevet. Her er det kriterier for hva de ulike rapportene skal ha med, standarden forteller overordnet hva rapportene skal ha med og når disse skal skrives i prosjektperioden. Dette er en standard vi ikke har vært helt konsekvent med. Vi var allerede på utvikling av systemet fra dag en, noe som gjorde at standarden for dokumentasjonskriving ble forskjøvet og skrevet side om side med utviklingen av oppgaven. Systemkravrapporten ble spesielt forskjøvet under et problem vi hadde i MAAS i startfasen som gjorde at dokumentasjonen ble satt på vent til vi hadde kommet oss videre. Det gjorde at fremdriftsdiagrammet som man ser ble forksjøvet. Metodene vi har brukt i prosjektet har vært knyttet til Alan R. Hevners 'A three Cycle View of Design Science Research'[19]. Denne metoden har fungert veldig bra for oss, da den iterative prosessen som han beskriver har gitt gode resultater. Metoden for dokumentasjonskriving har blitt utviklet fra malene vi har fått fra NTNU, som vi har gått utifra, men også delvis gjort vår egen. Metode for problemløsing og informasjonshenting gikk på bruk av litteratur og Internett.

- Bruk av litteratur og Internett
  I denne oppgaven har alt av informasjon blitt funnet via nettsøk, spørsmål på forum og lesing av offisiell dokumentasjon som er lagt ut for de verktøyene vi har tatt i bruk. Mange av problemene har vi fått løst via å finne andre som har hatt liknende problemer, og tilpasset deres løsninger til vårt system.

- Oversikt over maskinvare som er brukt
  Maskiner: 4x HP Z4 G4 Workstation
  - Hovedkort: HP 81C5
  - CPU: Intel Xeon W-2102
  - RAM: 2x Hynix HMA84GR7AFR4N-VK 32GB ECC
  - Nettverk: Intel Ethernet Connection (2) I219-LM
  - Skjermkort: NVIDIA GP107GL
  - Lagring: 2x Seagate ST4000NM0165 ATA
  - Lagring: SAMSUNG MZVKW512HMJP-000H1 521GB

  Én D-Link DGS-1024D switch
  Én D-link DIR-842 ruter

- **Kort beskrivelse av standarprogramvare som er brukt**
  Ubuntu Server 18.04 LTS. Dette er Operativsystemet Ubuntu Server versjon 18.04 Long Time Support. Det er et Linux system og er open-source. Det er laget av Canonical som er de samme som lager både Juju og MAAS.
  CentOS 7. Dette operativsystemet er også Linuxbasert. Det er et veldig stabilt operativsystem som passer veldig godt som NFS-server.
  Ubuntu MAAS er et verktøy som er designet for orkestrering av 'bare-metal' maskiner. Det utfører veldig kompliserte oppgaver, ved å skjule de bak enkle abstraksjoner.
  Juju er et verktøy som baserer seg på å gjøre veldig kompliserte systemer enkle å sette opp. Her kan man bruke "charms" som er ferdig definerte installasjoner av diverse programmer man trenger.
  Ansible er et verktøy som skal hjelpe administratorer å konfigurere større mengder med servere og maskiner, ved å automatisere SSH-tilganger via 'playbooks'.
  Kubernetes er et verktøy laget av Google for å orkestrere store mengder containere.

- **Hvordan arbeidet ble fordelt mellom personene i gruppen**
  I dette prosjektet har Even Ofstad og Eivind Dybvik tatt på seg en arbeidsmetode som går ut på å jobbe parallelt om oppgavene vi har påtatt oss. Det vil si å skrive dokumentasjon mens man utfører oppgaven. Dette er for å utnytte ressurser på en mest mulig effektiv måte, da man alltid vil ha arbeid pågående i arbeidstid. Arbeid utenom vanlig arbeidstid skal meldes inn og skrives ned i vår timeliste, slik at alt arbeid blir regnet med i vår sluttrapport.

- **Oversikt over dokumentasjon som er utarbeidet**
  Forstudierapport
  I forstudierapporten ble interessenter, bakgrunn, og mål for prosjektet presentert. Verdien av prosjektet ble også analysert.
  Systemkravrapport
  I denne rapporten setter vi kravene til både hardware og software.
  Driftdokument
  I driftdokumentet blir det forklart i detalj hvordan vi har satt sammen systemet, og hvordan systemet fungerer.
  Sluttrapport
  I sluttrapporten blir det gitt en oppsummering over hva som har blitt gjort gjennom prosjektet, og anbefalinger til videre arbeid.
  Kost/Nytte-analyse
  I kost/nytte-analysen viser vi antatt kost av prosjektet, og potensiell nyteverdi.
  Risikoanalyse
  I risikoanalysen ser man hvilke risiko som potensielt kan oppstå, hvor sannsynlig de er, og hvor truende de er for prosjektet. De blir også gitt instrukser om hvordan man kan mitigere problemene, og hva man skal gjøre hvis de oppstår.
  Timeskjema
  Timeskjemaet viser hvor mye prosjektdeltakerne har jobbet med oppgaven.
  Arbeidsplanlegging i ganttdiagram
  I ganttdiagrammet viser man til hvordan man har lagt opp arbeidsoppgavene, og hvor mye tid man skal bruke på hver del.

35 Gjennomføring av prosjektet

Hele prosessen gjennom prosjektperioden har bestått av kontinuerlig målrettet arbeid med systemet. Vi har alltid hatt en klar forståelse av hva som er oppgaven for hver av oss for dagen. Disse oppgavene har store deler av prosjektperioden bestått av feilsøking. Enten ved å søke opp problemet, spørre på forum, eller diskusjon med hverandre og/eller med input fra oppdragsgiveren vår på SINTEF. Dette er ikke noe vi planla ved prosjektstart, men heller noe som var et resultat av dårlig grunnkunnskap om systemene og uflaks. Med uflaks tenker jeg på problemer med feil grunnet hardware-til-software komplikasjoner. Det positive ved å møte slike feil, er at man bli veldig godt kjent med systemet/programmet, noe vi i prosessmålet spesifiserte at vi skulle. Dette gir oss også en bra forståelse på hvordan en praktisk prosjektoppgave hos en bedrift foregår.
Resultatmålene fra forstudierapporten reflekterer bra vår utføring av prosjektet. Det vi ikke fikk gjort som vi hadde i resultatmålene er resultater som fysisk ikke var mulig for oss. For eksempel kan vi ikke ha 99.99 prosent oppetid når hele oppgaven baseres på en server (single point of failure). Dette gjør også at skaleringen fra skyen blir vanskelig.
Dersom vi hadde startet på nytt er det noen ting vi ville gjort annerledes. Selve feilene vi gjorde ville vi selfølgelig unngått eller fikset veldig fort. Utenom det ville en bedre forståelse av de forskjellige programmene vi skulle bruke fra staren hjulpet oss. Dette hadde nok spart oss for mye tid, da et par feil kunne vært unngått, samt tiden det tar å forstå feilene hadde blitt redusert. Vi burde også vært bedre på å reflektere og skrive hva vi lærte av feilene underveis. Når vi hadde feil, feilsøkte vi –> skrev ned feilen –> feilsøkte mer –> fikset feilen –> gikk videre. Dersom vi hadde gjort dette på nytt ville vi også reflektert over feilene og skrevet hva som var årsaken, samt hva som fikset problemet.

Ved start av prosjektet visste vi lite om systemet og hva alt av de ulike programmene vi skulle lære om og implementere var. Det var derfor vanskelig for oss å kvantifisere hvor lang tid vi kom til å bruke på de ulike delene av prosjektet, og ikke minst hvor feilene vi kom til møte på var. Det er derfor den planlagte prosessen i framdriftsplanen ikke helt reflekterer faktisk brukt tid på de fleste prosesser. Dette vises også i selve framdriftsplanen.
Timeregnskapet reflekterer nettopp dette, her ser man hvor mange timer vi brukte på ulike feil og problemer gjennom prosjektperioden (mest med MAAS). Man ser også her hvor lite timer som gikk andre steder i prosjektet hvor vi gjorde mye av systemet. Vi har virkelig merket hvordan det er å jobbe i et IT-prosjekt, der noen uker er kraftig fremgang, mens andre gir bortimot null. Det kan være ganske frustrerende å jobbe med samme type problem over flere uker der man prøver å søke seg frem til andre med lignende problem, eller å spør på forum om hjelp, eller prøve selv via diskusjon (finne mest mulig logisk forklaring i forhold til feilmelding og logger). En blanding mellom disse tre gjorde at vi omsider kom oss gjennom feilene vi hadde, noe som har vært noen av de beste opplevelsene gjennom bacheloren. Feil er viktig for å lære seg program og system best mulig. Det er også viktig for å bli en best mulig feilsøker, som er det vi har lært mest av utenom selve systemet.

Dokumentasjonen har heller ikke blitt gjennomført helt etter planen fra gantt-diagrammet (eller prosessmålene). Det var en periode vi var veldig dedikert til å fikse en feil i MAAS, noe som gjorde at det ble vanskelig å skrive dokumentasjon samtidig da man ikke helt visste hvordan det ferdige systemet skulle se ut. Vi utsatte derfor dokumentasjonen etter systemkravrapporten til vi var ferdige med systemet og kunne bedre skrive dokumentasjonen.

I dette prosjektet jobbet vi jevnt, og i henhold til våre planer fra start til slutt. Vi har oppnådd det vi har satt ut for å oppnå gjennom prosjektet

36 Videre arbeid

Vurder hva som kan gjøres videre, eksempelvis for at produktet skal kunne settes i produksjon eller drift. I denne oppgaven har vi tatt på oss oppstarten av et system som i praksis alltid kan forbedres. Som systemet er nå, fins det mange potensielle forbedringer. Oppsett av automatisk backup, fjerne 'single points of failure', sette opp et blockchain-rammeverk, dirigere nett-trafikk bort fra Region-controller, bedre bruk av hver node, sette opp ekstern Power Supply Unit, monitorering av tjenestene, bedre brannmur.

- Automatisk Backup
  Et system som dette, som kanskje akkumulerer store mengder data, vil være avhengig av at dataen er trygt lagret. Hvis det skjer noe med systemet, og kunder har viktig data lagret der, vil det være imperativt å ha en klok backupløsning. Dette vil si en backupløsning som gjerne har en off-site og en offline lagring.

- Single Point of Failure
  I dette systemet har vi noen Single Points of Failures. Her inkluderer vi Region/rack-controller, som ikke er i High Availability, og som all trafikk går gjennom. All trafikk går også gjennom en enkelt switch. Våre Etcd, EasyRSA, og Kubernetes load-balancer-noder er også uten high availability, og kjører bare på Server4. NFS-serveren bør også opprettes i High availability.

- Blockchain-rammeverk
  Vi har i dette prosjektet lagt rammeverket for å kunne kjøre blockchain-rammeverk som Hyperledger Fabric eller Hyperledger Sawtooth. For å kunne kjøre blockchainteknologi på dette systemet bør en slik løsning installeres.

- Dirigere nett-trafikk
  Som systemet er nå, går all nettverkstrafikk gjennom Region/rack-controller. Dette er ikke anbefalt i en reell drift-setting. Region/rack-controller bør settes som et punkt i topologien, og ikke som gateway for 11.0.0.0/8 nettverket.

- Bedre bruk av hver node
  Systemet kan forbedres, ved å bruke bare-metal som worker-nodes til Kubernetes. Da kan Kubernetes selv utnytte ressursene etter behov. Dette systemet er i teoriene laget for å ha en rekke bare-metal servere å jobbe med, og ikke virtuelle maskiner.

- Sette opp ekstern PSU og/eller nødaggregat
  Dette systemet bør ha tilgang til en ekstern PSU hvis man vil garantere en god oppetid for tjenestene man kan tilby på det. Hvis et strømbrudd oppstår vil hele systemet være nede til noen slår det på etter strømmen

har kommet tilbake. Ut i fra hvor kritiske tjenestene som kjører på systemet blir, vil vi også anbefale å se på nødaggregat for å garantere strøm gjennom lengre strømbrudd. Alt dette kan fikses ved å flytte systemet til en serverpark.

- Monitorering av tjenestene
  For å finne feil, og ta seg av alarmer før de utvikler seg til å bli problemer, bør man sette opp monitorering av systemet. Dette kan være kritisk for å ha et pålitelig system.

- Brannmur
  Nå har Region/rack-controller en generisk brannmur. Det kan være bedre for sikkerheten om det blir satt opp en mer dedikert brannmur.

37 Referanser

[1] https://www.ubuntu.com/download/server/provisioning
Offisielle nedlastningssiden til Ubuntu MAAS. Besøkt: 15.2.2019

[2] https://docs.MAAS.io/2.5/en/installconfig-network-dhcp
Dokumentasjon på hvordan MAAS DHCP fungerer. Offisiell dokumentasjonsside. Besøkt: 7.2.2019

[3] https://discourse.maas.io/t/forcing-gpt-boot-disk-working-around-boot-order-issues/262
Bugrapport hos discourse.maas (forum for alt MAAS-relatert) startet av 'KingJ'. Besøkt: 20.2.2019

[4] https://docs.okd.io/latest/install/host_preparation.html
Openshift Kubernetes Distribution (OKD) dokumentasjon med undertittel "Prepare your hosts". Besøkt: 03.04.2019

[5] https://Jujucharms.com/canonical-kubernetes/
The charmed distribution of Kubernetes. gitt ut av bruker 'containers'. Besøkt: 11.3.2019

[6] https://stackoverflow.com/questions/32429259/ansible-fails-with-bin-sh-1-usr-bin-python-not-found
Forumtråd på stackoverflow.com startet av bruker 'jdavis' Sep 6, 2015. Besøkt:

[7] https://github.com/kubernetes/kubernetes/issues/44665
Bug rapport av bruker 'fedya' Apr 19, 2017. Besøkt: 20.3.2019

[8] https://www.howtoforge.com/nfs-server-and-client-on-centos-7
Howtoforge, artikkel om installasjon av NFS server for Kubernetes. Forfatter: Srijan Kishore. Besøkt: 1.4.2019

[9] https://youtu.be/AavnQzWDTEk
Youtube film. Instruksjonsvideo over å installere NFS for Kubernetes. Utgiver: Justmeandopensource' Besøkt: 3.4.2019

[10] https://kubernetes.io/docs/concepts/storage/storage-classes/
Dokumentasjonsside for Kubernetes. Besøkt: 1.4.2019

[11] https://github.com/justmeandopensource/kubernetes/blob/master/yamls/nfs-provisioner/rbac.yaml
Github for 'justmeandopensource'. Besøkt: 4.4.2019

[12] https://github.com/justmeandopensource/kubernetes/blob/master/yamls/nfs-provisioner/class.yaml
Github for 'justmeandopensource'. Besøkt:4.4.2019

[13] https://github.com/justmeandopensource/kubernetes/blob/master/yamls/nfs-provisioner/deployment.yaml
Github for 'justmeandopensource'. Besøkt: 4.4.2019

[14] https://github.com/justmeandopensource/kubernetes/blob/master/yamls/4-pvc-nfs.yaml
Github for 'justmeandopensource'. Besøkt: 4.4.2019

[15] https://www.ansible.com/overview/how-ansible-works
How ansible works. Besøkt: 22.4.2019

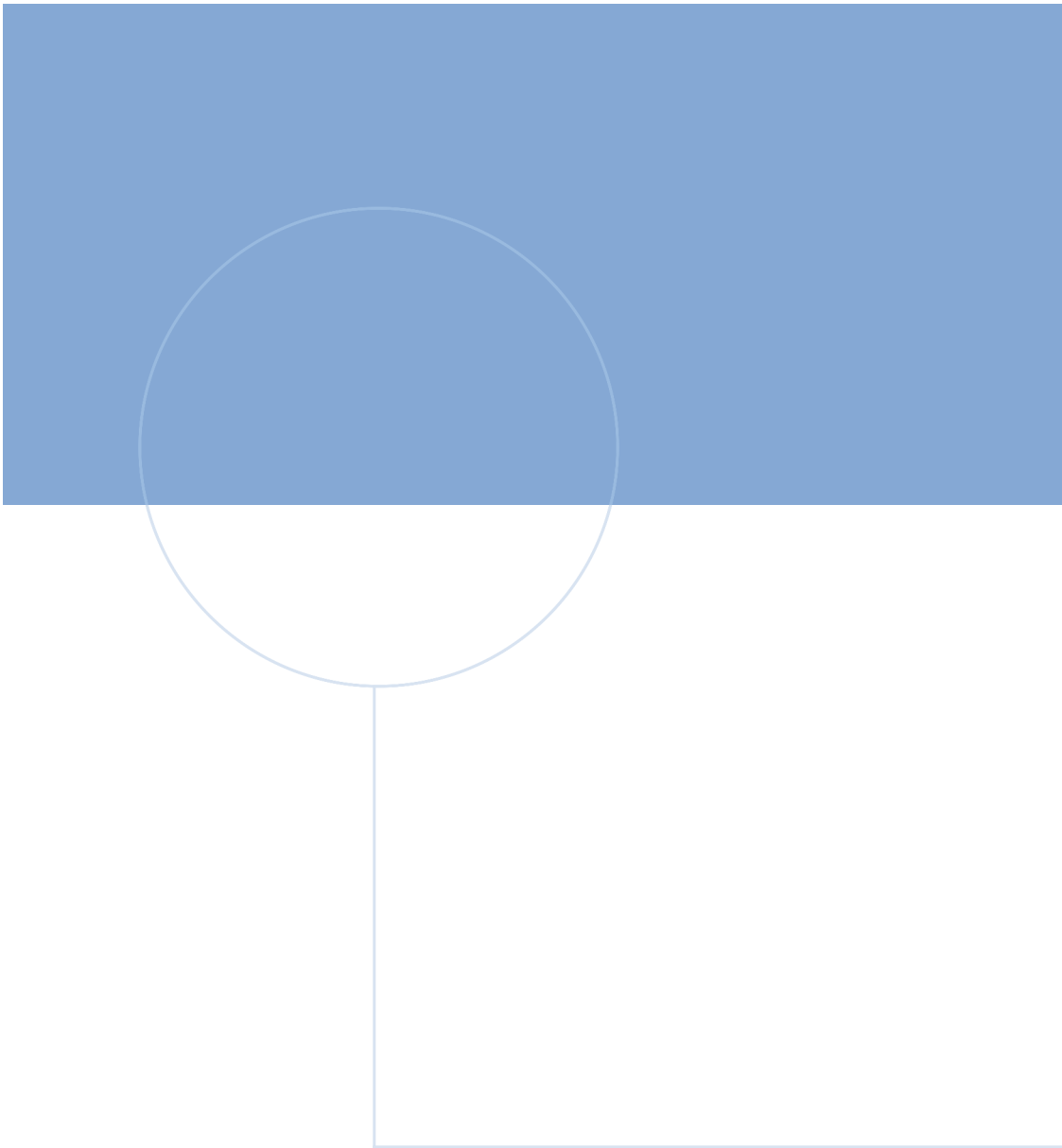
[16] https://Jujucharms.com/u/containers/easyrsa/
EasyRSA Juju charm
Author: Matthew Bruzek <Matthew.Bruzek@canonical.com> Contributor: Cory Johns <Cory.Johns@canonical.com> Besøkt: 15.3.2019

[17] https://coreos.com/flannel/docs/latest/flannel-config.html
Informasjon om programvaren Flannel. Besøket 4.4.2019

[18] https://kubernetes.io/docs/reference/glossary/?fundamental=true
Dokumentasjonsside for Kubernetes. Besøkt: 6.4.2019

[19] https://www.uio.no/studier/emner/jus/afin/FINF4002/v13/hefner-design.pdf
'A Three cycle View of Design Science Research. Forfatter: Alan R. Hevner. 01.01.2007. Besøkt: 16.1.2019

[20] https://bugs.launchpad.net/maas/+bug/1616231
Bug report startet av Andres Rodriguez. 23.08.2016 Besøkt: 4.3.2019



NTNU

Kunnskap for en bedre verden