BACHELOROPPGAVE:

Openstack-testing

FORFATTERE:
Terje Pedersen
Kevin Abadi Barhaugen
Harald Sletten

DATO:
15.05.2015

Sammendrag av Bacheloroppgaven

Tittel:	Norwegian title.	Nr: 35
		Dato: 15.05.2015
Deltakere:	Terje Pedersen	
	Kevin Abadi Barhaugen	
	Harald Sletten	
Veiledere:	Eigil Obrestad	
Oppdragsgiver:	Høgskolen i Gjøvik	
Kontaktperson:	Erik Helmås, erik.helmas@hig.no, 61135000	
Stikkord	Norway, Norsk	

Antall sider: 150	Antall vedlegg: 8	Tilgjengelighet: Åpen

Kort beskrivelse av bacheloroppgaven:
Bedrifter har i dag et behov for å drive sikkerhetstrening på egne isolerte nettverk. De trenger et skalerbart system som kan settes opp hurtig og automatisk. Oppgaven tar for seg en oppgradering av fjorårets bacheloroppgave AutoStack, som omhandlet automatisk utrulling av en OpenStack-skyløsning. Rapporten beskriver hvordan AutoStack oppgraderes til nyere versjoner av OpenStack (IceHouse og Juno), samt hvordan administrasjonsverktøyet Foreman er blitt en integrert del av løsningen. Resultatet er en OpenStack Juno skyløsning som rulles ut fra Foreman, der man etterpå kan verifisere om utrullingen var suksessfull med et test-rammeverk.

Summary of Graduate Project

Title:	Openstack-testing	Nr: 35
		Date: 15.05.2015
Participants:	Terje Pedersen	
	Kevin Abadi Barhaugen	
	Harald Sletten	
Supervisor:	Eigil Obrestad	
Employer:	Høgskolen i Gjøvik	
Contact person:	Erik Helmås, erik.helmas@hig.no, 61135000	
Keywords	Thesis, Latex, Template, IMT	
Pages: 150	Appendixes: 8	Availability: Open

Short description of the main project:
Enterprises currently has a need to conduct safety training on its own isolated network. They need a scalable system that can be set up automatically and rapidly. This thesis has an upgrade of last year's bachelor thesis AutoStack, which automatically deployed an OpenStack-cloud solution. This report describes how AutoStack is upgraded to newer versions of OpenStack (IceHouse and Juno), and how the management tool Foreman has been an implemented part of the solution. The result is an OpenStack Juno cloud solution, deployed from Foreman, where afterwards one can verify whether the deployment was successful with a test-framework.

Forord

Erik Hjelmås [1] er førsteamanuensis ved Høgskolen i Gjøvik. Han har vært ansatt i HiG siden 1996, og underviser i emnene Operativsystemer og Systemadministrasjon. Erik Hjelmås er også ansvarlig for studiet Bachelor i drift av nettverk- og datasystemer. Erik Hjelmås og Høgskolen i Gjøvik ønsker å videreutvikle sin eksisterende skyløsning Sky-HiGh med en forbedret versjon, og kjøre denne løsningen med færre manuelle prosesser enn før.

Gruppen ønsker å gi en stor takk til oppdragsgiver Erik Hjelmås for god kommunikasjon og veiledning, samt en stor takk til veileder Eigil Obrestad for god veiledning og konstruktiv kritikk for denne bacheloroppgaven.

Gjøvik, 15.5.2015

Harald Sletten

Terje Pedersen

Kevin Abadi Barhaugen

Contents

List of Figures

1 Introduksjon

1.1 Bakgrunn

Enkelte bedrifter har i dag et behov for å drive sikkerhetstrening på et eget lukket nett med ulike scenarier, der man ikke risikerer å forårsake nedetid på systemer som allerede er i produksjon. Et eksempel på dette er Forsvarets Ingeniørhøgskole/Cyberforsvaret. De trenger realistiske omgivelser der de kan øve og trene på cyber defence-operasjoner. Til dette behøves et skalerbart system der man, hvis ønskelig, kan bruke egen maskinvare for å sette opp løsningen raskt og automatisk.

Det er her OpenStack kommer inn i bildet. OpenStack er en åpen skyløsning som brukes for å lage en privat eller offentlig skyløsning. Løsningen samler all CPU, minne, lagring og nettverksressurser fra ulike maskiner i et nettverk, og gjør alt dette tilgjengelig for administrasjon. OpenStack er et av de raskest voksende open-source prosjektene i verden og det er også blitt tatt i bruk ved HiG. Der har det fått navnet SkyHiGh, hvor det blir brukt til undervisning og forskning.

Det ble våren 2014 levert en suksessfull bacheloroppgave ved navn AutoStack som tok for seg automatisk installering av OpenStack, samt automatisk utrulling av et OpenStack-miljø. Det er ønskelig at man bygger videre på denne oppgaven. Dette medfører at AutoStack blir tilpasset nyeste versjon av OpenStack, Juno, samt innføring av et test-rammeverk for å verifisere at den automatiske installasjonen/utrullingen ble utført korrekt.

1.2 Problemområde

Fram til nå har formålet med SkyHiGh vært hovedsaklig testing, men den har også blitt brukt som en god læringsplattform for mange IT-studenter. Nå ønsker HiG å utvide sin målgruppe til bedrifter. Bedrifter ønsker å ha en skyplattform hvor de kan teste ulike scenarier som de lager selv. Disse scenariene blir vanligvis brukt for å teste forskjellige sikkerhetsløsninger, og simulere hacker-angrep og lignende. HiG vil derfor tilby AutoStack som en løsning bedrifter kan bruke for denne typen testing.

1.3 Avgrensning

Prosjektet er definert som en oppgradering av AutoStack, og dermed er det kun det gruppen skal gå ut ifra. Da det er sagt, så kommer løsningen til å bli redesignet, og det ferdige produktet kan framstå helt annerledes enn det gruppen begynte med, men funksjonaliteten forblir den samme.

Når det gjelder utstyr som skal brukes, holder gruppen seg kun til de maskinene gruppen får låne av skolen. Disse maskinene befinner seg på rom K202, og dette rommet blir derfor av praktiske årsaker gruppens arbeidsplass. Av eksterne IP-addresser kommer gruppen til å få utdelt noen få for bruk i prosjektet, frem til da så holder gruppen seg til DHCP.

1.4 Oppgavedefinisjon

Oppgaven omhandler å bygge videre på bacheloroppgaven 'AutoStack' fra våren 2014, og den kan deles inn i tre deler.

I den første delen skal gruppen oppdatere AutoStack fra dens nåværende OpenStack-versjon Havana frem til IceHouse som er neste versjon. I tillegg har gruppen blitt bedt om å bytte ut PXE bootserveren Cobbler med administrerings-systemet Foreman, der Puppet er en integrert del av systemet. Dette vil gjøre AutoStack både lettere å bruke, og lettere å teste for gruppen senere i oppgaven.

Etterpå følger videre oppdatering av AutoStack. Denne gangen oppdaterer gruppen den videre til den aller nyeste versjonen av OpenStack, Juno. Her er det dog et lite problem; siden det blir brukt ferdiglagde Puppet-moduler fra Puppetlabs [1] og Stackforge [2] som hovedkonfigurasjonsmetode, så er gruppen avhengig av at disse er oppdatert. Disse er det de nevnte utgivere som er ansvarlige for å holde oppdatert. I skrivende stund (16.1.2015) så er ikke modulene oppdaterte, men gruppen håper de kommer ut tidlig nok, slik at disse kan brukes. Hvis ikke må oppgaven fullføres med IceHouse-versjonen av OpenStack.

Etter at gruppen har oppdatert AutoStack så skal gruppen teste ut AutoStack på to forskjellige maskinvaresammensetninger. Den ene består av kun HP Proliant 380 G5-servere, den andre er ikke spesifisert av oppdragsgiver enda. Dette er for å sikre stabilitet på en type maskinvare, og forsikre seg om at det også fungerer på en annen type maskinvaresammensetning.

I del to så skal gruppen lage et test-rammeverk for skyløsningen AutoStack. Her skal det forsikres at alle noder og tjenester er oppe og kjører etter utrulling, og det skal rapporteres tilbake til den som har satt opp systemet gjennom Foreman. Her vil gruppen se om det finnes eksisterende test-verktøy som kan tas i bruk, og utnytte disse så mye som mulig før det skrives egen kode.

Del tre består av å rulle ut to scenarier og verifisere at de fungerer som de skal. Dette skal gjennomføres i samarbeid med "Digitalt øvingsfelt"-gruppen. Et scenario består av en rekke maskiner, rutere og programvare som skal rulles ut virtuelt i OpenStack-løsningen. "Digitalt øvingsfelt"-gruppen er ansvarlige for å lage disse scenariene.

1.5 Målgruppe

Målgruppen for oppgaven er oppdragsiver og HiG som vil trenge denne rapporten som en manual eller guide for å implementere/oppgradere sin eksisterende SkyHiGh-løsning. IT-studenter som bruker SkyHiGh som en læringsplattform, er også en målgruppe for denne oppgaven. I tillegg vil også bedrifter kunne ha et behov for å teste sine scenarier i en slik skyløsning.

1.6 Formål

Formålet med denne oppgaven er å gi oppdragsgiver en mer brukervennlig OpenStack-løsning ved å bruke Foreman og dets webgrensesnitt. Noen bedrifter har et ønske om å kunne teste sitt system mot for eksempel kjente hacker-angrep, og oppdragsgiver vil derfor gi et tilbud til disse bedriftene for nettopp dette. Oppdragsgiver vil også ha en metode for å teste om OpenStack-løsningen ble rullet ut uten problemer, og vil derfor ha et test-rammeverk som sjekker at skyløsningen ble rullet ut korrekt.

1.7 Bakgrunn og kompetanse

Gruppen som skal jobbe med dette prosjektet går alle sammen studiet "bachelor i drift av nettverk- og datasystemer". Dette gjør at den faglige bakgrunnen er forholdsvis lik hos alle tre, og gruppen kjenner derfor hverandres faglige styrker og svakheter. Utenom dette så har Terje Pedersen fagbrev i IKT-servicefag, Kevin A. Barhaugen har tatt IT-fag på videregående skole, og Harald Sletten har gått på folkehøgskolen med fokus på data, noe som også vil komme til nytte.

Oppgaven passer veldig godt til gruppens studium, fordi den omhandler fysisk interaksjon med servere, bruk av skripting og koding på flere forskjellige nivåer, nettverk, og tar utgangspunkt i flere av de fagene gruppen har hatt. Dette er fag som Operativsystemer, Systemutvikling, Systemadministrasjon, Nettverksadministrasjon, og Database- og applikasjonsdrift.

Det som var nytt for gruppen var at gruppen skulle bruke Foreman til å kontrollere utrullingen av AutoStack-løsningen. Gruppen måtte først og fremst sette seg inn i hvordan man installerte dette riktig på utrullingsserveren, og deretter lære seg funksjonaliteten som Foreman tilbyr i sitt grensesnitt. For at servere skal bli konfigurert automatisk ble Puppet benyttet. Gruppen kunne litt om dette fra Systemadministrasjon, men det måtte fremdeles leses mer om Puppet på detaljnivå. OpenStack generelt kunne gruppen heller ikke så mye om. Gruppen har brukt OpenStack i fag som Database- og applikasjonsdrift og Systemadministrasjon, men visste ikke hvordan det ble satt opp eller fungerte. Gruppen brukte derfor en god del tid på OpenStack sine installasjonsguider for å lære mer om installasjonsprosessen, og om OpenStack sine komponenter.

1.8 Rammer

Gruppen har fått disse kravene fra oppdragsgiver:

- Oppgaven skal bygges videre fra fjorårets AutoStack-oppgave.
- Operativsystemet som løsningen installeres på skal være Ubuntu Server 14.04
- Utrullingsverktøyet som brukes skal være Foreman.
- Gruppen skal forholde seg til de offisielle puppet modulene fra OpenStack

Ellers så gjelder rammene gruppen har utarbeidet med veileder og HiG. Dette innebærer møter, frister for innlevering og statusrapporter:

- Frist for rapportens innlevering: 15. Mai.
- Det skal leveres tre statusrapporter.
- Gruppen har møte med veileder én gang i uken dersom det er behov.

1.9 Øvrige roller

Oppdragsgiver for denne oppgaven er førsteamanuensis Erik Hjelmås ved Høgskolen i Gjøvik. Han er ansvarlig for gruppens studie "Bachelor i drift av nettverk og datasystemer", og er emneansvarlig i fagene Operativsystemer og Systemadministrasjon. Samtidig er han også ansvarlig for Høgskolen i Gjøviks

egen skytjeneste SkyHiGh. Han kan derfor mye om vårt tema og kan være til stor hjelp under arbeidet med denne oppgaven. Veileder for denne oppgaven er høgskolelærer Eigil Obrestad. Han er foreleser i Operativsystemer ved Forsvarets ingeniørhøgskole i Lillehammer, og går nå et masterstudium i informasjonssikkerhet.

1.10 Organisering av rapporten

Rapporten tar først for seg teorien som behøves for å kunne forstå rapportens innhold, samt hva OpenStack-løsningen består av. Etter dette går rapporten dypere inn i gruppens løsning, og beskriver videre hvordan løsningen skal settes opp i form av design på løsningen. Implementeringen av løsningen omhandler i stor grad, hvordan gruppens utrullingsserver blir installert, samt også hva som blir gjort for å sette opp skyløsningen. Etter implementasjonen dekker rapporten utfordringene som har blitt løst underveis i prosjektet, etterfulgt av testing og kvalitetssikring av gruppens OpenStack-løsning.

Rapportens vedlegg inneholder blant annet en brukerveiledning for Foreman, og en installasjonsguide for å installere utrullingsserveren, i tillegg til å sette opp OpenStack-løsningen med test-rammeverk.

2 Kravspesifikasjon

I dette kapittelet beskrives hvilke krav som må oppfylles av gruppens skyløsning. Her forklares litt av funksjonaliteten OpenStack-løsningen skal ha, og de forskjellige kravene til brukervennlighet, pålitelighet og ytelse.

2.1 Use Case

Et use case viser, i grove trekk, hvilken funksjonalitet et produkt, eller i gruppens tilfelle skyløsningen, skal ha. "Use case"-modellen viser også hvem eller hva som har tilgang til de forskjellige funksjonalitetene.

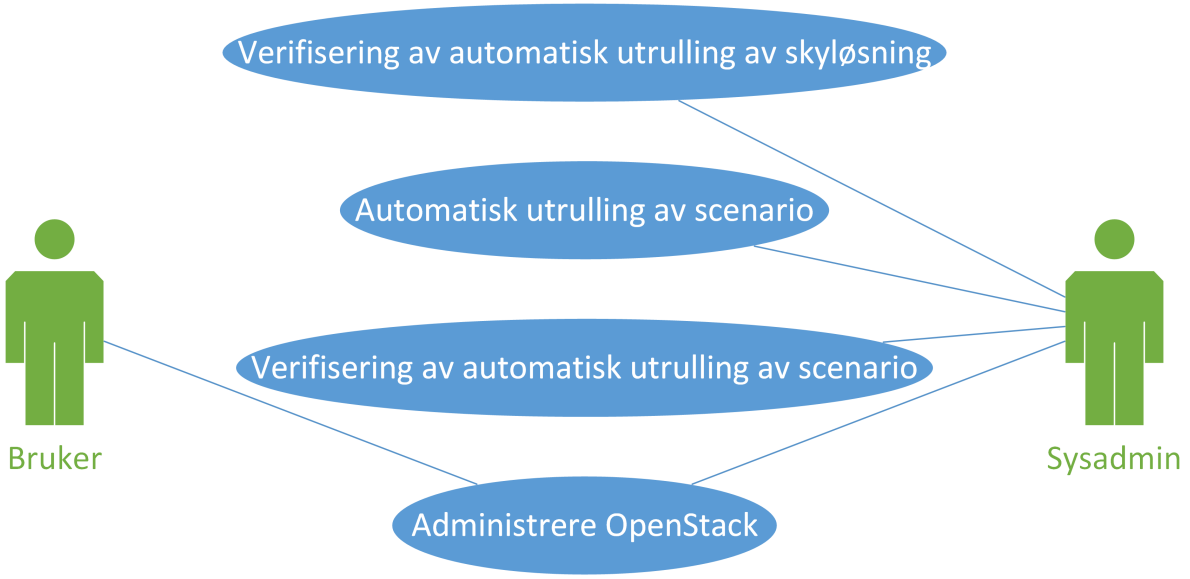


Figure 1: Use Case.

Use case navn: Verifisering av automatisk utrulling av skyløsning

Beskrivelse: Etter at AutoStack har kjørt og alle noder er operative, kan systemadministratoren velge å kjøre et skript som verifiserer at alle tjenester er tilgjengelige. Hvis noe ikke er som det skal vil systemadministratoren få beskjed om dette, og om mulig, hva som må gjøres for å rette opp feilen.

Use case navn: Automatisk utrulling av scenario

Beskrivelse: Etter suksessfull utrulling av AutoStack kan systemadministratoren, om ønskelig, bruke et skript for å rulle ut scenarier automatisk. Administratoren trenger bare å gjøre små endringer i skriptet hver gang et nytt scenario skal rulles ut.

Use case navn: Verifisering av automatisk utrulling av scenario

Beskrivelse: Når scenariet er ferdig utrullet kan systemadministrator velge å kjøre en test på scenariet for å se om alt er som det skal. Det vil skje på samme måte som med testen av selve skyløsningen, men her vil de virtuelle maskinene testes i stedet.

Use case navn: Administrere OpenStack (Sysadmin)

Beskrivelse: Gjennom webgrensesnittet Horizon, en tjeneste levert av OpenStack, kan systemadministrator administrere og overvåke alle de virtuelle maskinene og nettverkene som er satt opp. Administratoren styrer tilgang og rettigheter til brukere.

Use case navn: Administrere OpenStack (Bruker)

Beskrivelse: Gjennom Horizon kan brukeren gjøre mye av det samme som en systemadministrator, men brukeren har mer begrenset tilgang. Brukerne kan ikke se eller styre andre virtuelle maskiner eller nettverk enn det de har fått tilgang til, og har en viss begrensning på hvor mye ressurser de får bruke.

2.2 Kvalitetsmessige og operasjonelle krav

Brukervennlighet

Det gruppen her mener med brukervennlighet er hovedsaklig brukervennlighet mot administratoren i systemet. Dette er fordi selve brukeren kun har webgrensesnittet Horizon å forholde seg til, som er laget av selve OpenStack teamet, dette er noe gruppen ikke får gjort noe med. For administratoren er det viktig at koden er lett å lese dersom det er ønskelig å tilpasse koden til eget miljø og oppsett. Gruppen må derfor sørge for at egenprodusert kode følger standarder [link] for kodestil og kommentering. Dokumentasjonen skal være detaljert og tydlig nok til at de som leser den forstår teorien og kan sette opp skyløsningen selv. Det er også et krav at løsningen skal være så automatisk som mulig. Dette innebærer å unngå manuelle konfigurasjoner.

Test-rammeverket som skal settes opp vil gi administratoren en enklere problemløsningsprosess for systemet, og det vil i tillegg være tidsbesparende. Test-rammeverket skal verifisere at all funksjonalitet i skyløsningen fungerer. Administrator trenger bare å kjøre test-rammeverket, og se på rapporten som blir produsert.

Ytelse

Administrator skal ikke bruke lang tid for å sette opp en OpenStack-løsning. Gruppen har satt som et krav at oppsettet av OpenStack-løsningen skal settes opp på en dag. Denne løsningen skal ikke være laget for å fungere over lengre tid. Løsningen skal raskt settes opp, og leve en bestemt tid som er avtalt med administrator. Når denne tiden er brukt skal sky-løsningen fjernes fra alle servere som er involvert.

Pålitelighet

Det er satt som et krav at løsningen garantert skal fungere på HP DL380G5-servere, men med litt konfigurering av skript og kode, skal det også fungere på det meste av annen maskinvare. Test-rammeverket som skal settes opp vil hjelpe administratoren med å avdekke feil og lettere finne løsningen på problemet. Dette vil sørge for at AutoStack blir mer stabilt og pålitelig.

2.3 Deployment View

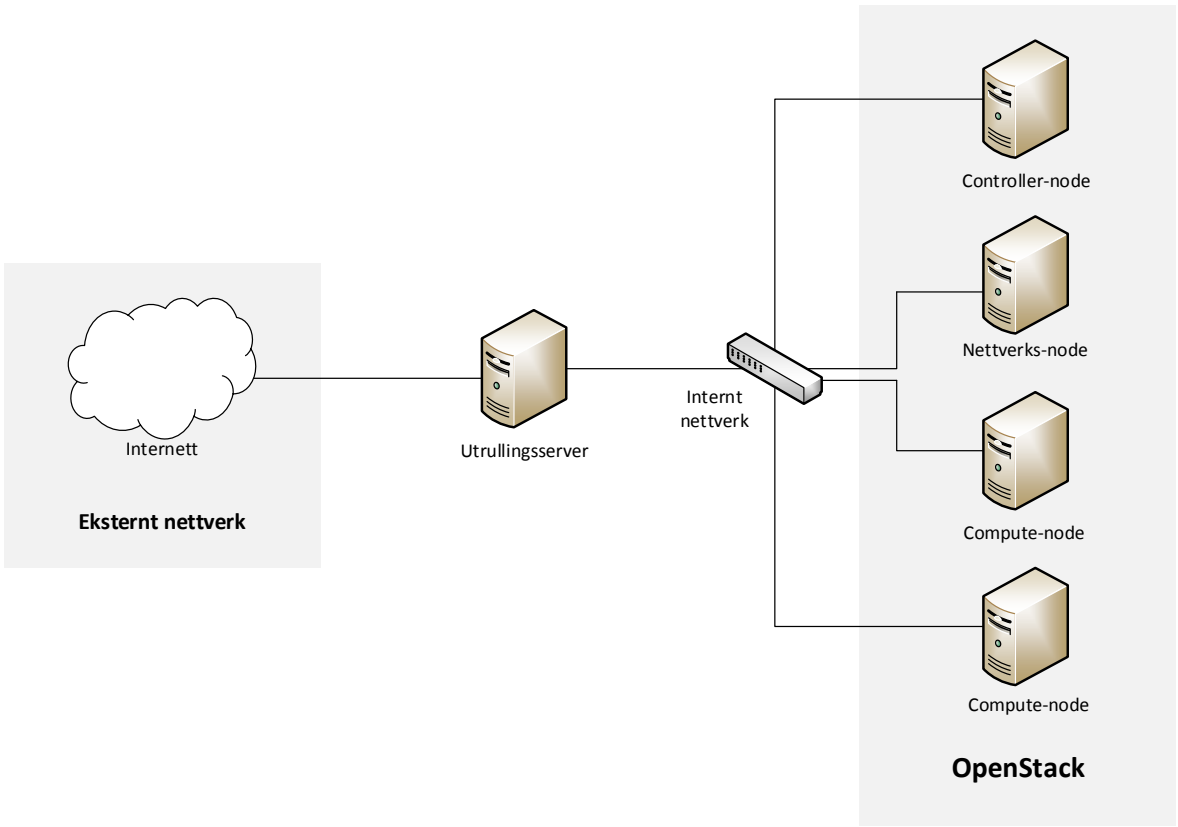


Figure 2: Deployment View.

I denne oppgaven kommer gruppen til å benytte seg av fem servere og èn switch. Utrullings-serveren vil bli satt opp mellom det eksterne nettverket og OpenStack-miljøet. Dette er serveren som er ansvarlig for å rulle ut hele AutoStack løsningen og alle endringer og konfigurasjoner som blir gjort på nodene, vil komme fra denne. Utrullingsserveren vil også fungere som en ruter mellom det eksterne nettverket og det interne OpenStack-miljøet, slik at man har tilgang til internett eller andre nettverk/tjenester. De resterende serverne vil da bli tildelt ulike roller i AutoStack-løsningen, derav èn kontroller-node, èn nettverks-node og to compute-noder. Kontroller-noden er ansvarlig for å administrere skyen. Nettverks-noden skal fungere som en ruter for de virtuelle maskinene der den sørger for kontakt med omverdenen i form av NAT, og virtuelle lokalnettverk med DHCP. Compute-nodene skal tilby sin maskinvare i form av CPU, RAM og HDD for å tjene de virtuelle maskinene i skyen.

3 Teoretisk grunnlag

I dette kapittelet vil det bli presentert teknologier som er relevante for oppgaven. Det er en hel del terminologi, begreper og konsepter som er nødvendig for å få med seg essensen av oppgaven.

3.1 OpenStack

OpenStack [3] er en sky-løsning som kontrollerer store forekomster av databehandlings-, lagrings-, og nettverksressurser. Alt styres gjennom et webgrensesnitt "Horizon" som gir administratorer kontroll mens de kan gi tilgang til sine brukere. Der kan brukerne bruke de ressursene som er tilgjengelige i skyen gjennom et webgrensesnitt eller en kommandolinje. OpenStack er gratis, og er en åpen kildekode. OpenStack lanserer en ny versjon av sin programvare to ganger i året, en versjon på våren (rundt april/mai), og en versjon på høsten (rundt oktober). OpenStack består av mange forskjellige del-komponenter hvorav hver av dem har sin egen jobb å utføre. Nedenfor er det nærmere beskrevet hva deres oppgave er.

3.1.1 Nova

Nova [4] er hovedkomponenten i OpenStack, og håndterer blant annet opprettelsen av virtuelle maskiner (VM). Nova er utformet for å administrere og automatisere store mengder dataressurser og kan også arbeide med offentlig tilgjengelige virtualiseringsteknologier, slik som VMware[5] og Hyper-V[6]. Noen av Novas oppgaver i OpenStack:

- Håndtere Local Area Networks (LAN) for rask klargjøring av nettverksmuligheter og sikkerhetsfunksjoner.
- Nova har ansvaret for å tildele IP-adresser til VM-er (floating-IP).
- Nova kan tildele og kontrollere tilgang til VM-er ved å lage regler under betegnelsen "security groups".
- Nova har ansvaret som hypervisor for VM-er [7]

Dette er bare noen av alle oppgavene Nova har ansvar for i OpenStack.

3.1.2 Neutron

Neutron[8] er komponenten som sørger for at administratorer og brukere kan administrere nettverk og IP-adresser. Neutron sikrer at nettverket ikke vil være en flaskehals eller generelt en begrensende faktor i skyen, og gjør at brukerne selv får kontroll over sine nettverkskonfigurasjoner. Nettverksmuligheter:

- Neutron er fleksibel i forhold til nettverksmodeller, og kan dekke behovene til ulike applikasjoner eller brukergrupper. Standard modeller inkluderer "flat networks" og "VLAN".
- Neutron håndterer IP adresser, både statiske IP-adresser og DHCP.
- Brukere kan lage sine egne nettverk, kontrollere trafikk, og koble servere eller enheter til ett eller flere nettverk.

3.1.3 Keystone

Keystone[9][10][11] er identitetstjenesten som brukes av OpenStack for autentisering og høy-nivå autorisasjon. Den fungerer som et felles godkjenningssystem over skyen. Keystone støtter flere former for autentisering som for eksempel standard brukernavn og passord. Keystone presenterer ett grensesnitt som andre tjenester bruker til autentisering og autorisering.

3.1.4 Glance

Glance[12] er OpenStack sin image-tjeneste. Glance finner, registrerer, og leverer tjenester for disk- og server images. Images kan brukes som en oppstartsreferanse for en VM. Image som Glance håndterer er et operativsystem som blir installert på VM-en. Man kan velge mellom flere images hvis man ønsker.

3.1.5 Heat

Heat[13] eller "orchestration" som den også kalles, er en tjeneste for å organisere flere sammensatte skyapplikasjoner ved hjelp av templates. Heat kan brukes til å sette opp mange forskjellige scenarier ved å bruke templates, for å teste for eksempel sikkerheten i et system. En template vil gjerne inneholde nettverk som skal settes opp, et eller flere subnet, rutere, og virtuelle maskiner. Ved å bruke templates kan man dermed sette opp et helt virtuelt system for testing i skyen.

3.1.6 Cinder

OpenStack Block Storage (Cinder)[14] gir blokknivå lagringsenheter som brukes sammen med OpenStack sine VM-er i skyen. Cinder håndterer opprettelsen av volum, samt knytte disse til servere, eller løsne de fra servere. Cinder volum er fullt integrert i OpenStack slik at brukere kan administrere sine egne lagringsbehov. Ved å ha en lagringsserver i et OpenStack system kan man knytte volum til VM-er i skyen, slik at VM-ene kan benytte seg av en ekstra lagringsplass.

3.1.7 RabbitMQ

RabbitMQ[15] er en meldingstjeneste, en slags "megler", en mellommann for meldinger. Det gir komponenter en felles plattform for å sende og motta meldinger, i tillegg til at meldingene har et sikkert sted å være før de mottas. I OpenStack brukes RabbitMQ nettopp som en meldingstjeneste mellom OpenStack sine komponenter slik at de kan snakke sammen.

3.1.8 MySQL

MySQL[16] er et databaseadministrasjonssystem. MySQL er mye brukt på grunn av dens brukervennlighet, og holder høy ytelse i forhold til både pris og krav til maskinvare. I OpenStack brukes MySQL for å administrere alle OpenStack-komponentenes databaser og tabeller.

3.1.9 Horizon

OpenStack Horizon[17] gir brukere og administratorer et webgrensesnitt for å få tilgang, administrere, og automatisere skybaserte ressurser. Designet skal tjene tredjeparts produkter og tjenester, som for ek-

sempel fakturering, overvåkning, og ytterligere styringsverktøy. Horizon er en av to måter brukere kan samhandle med OpenStack sine ressurser på, som for eksempel kommandolinjen.

3.2 Foreman

Foreman er et administrasjonsverktøy som skal hjelpe systemadministratorer å håndtere servere gjennom hele deres livssyklus. Man kan si at Foreman holder styr på tre ting:

- Provisjonering - sørger for installering av et operativsystem via PXE-boot
- Konfigurering - etter at serverne er satt opp med et OS, sørger Foreman for at serverne blir konfigurert automatisk.
- Monitorering - når maskinene er ferdig satt opp vil Foreman kunne holde oversikt over tilstanden til serverne.

3.3 DNS

DNS står for Dynamic Name Resolution. Det er en tjeneste som oversetter domenenavn (eller vanlige, lokale navn) til IP-addresser og motsatt.

3.4 PXE

PXE står for Preboot Execution Environment. Det er en tjeneste som gjør det mulig for en maskin å boote fra et nettverk i stedet for en harddisk. Den bruker protokollene DHCP og TFTP for å gjøre dette.

3.5 TFTP

En enkel filoverføringsprotokoll, designet for å være lett å implementere.

3.6 DHCP

Dynamic Host Configuration Protocol er en tjeneste som deler ut IP-adresse, subnettmaske og annen relatert IP-informasjon til enheter på ett nettverk.

3.7 NAT

NAT står for Network Address Translation. Dette er en funksjon hvor man tar én IP-addresse og setter et helt nettverk bak den. Fordi enheter ikke bare bruker en IP-addresse, men også gjerne porter i kommunikasjon mellom seg, oversetter NAT IP-addresser til enkeltporter og kan dermed ekspandere én IP-addresse til alt fra 65.000 IP-addresser , til én IP-addresse med 65.000 porter.

3.8 Puppet

Puppet er et konfigurasjonsstyringssystem som lar gruppen definere en tilstand på sin infrastruktur, for deretter å sørge for at dette automatisk blir konfigurert og vedlikeholdt utover alle servere og maskiner. Puppet bruker et deklarativt språk for å beskrive konfigurasjonen til en maskin, dvs hvordan tilstanden til en maskin skal være, i stedet for hvordan det skal gjøres.

3.9 Rally

Rally[18] er hovedsaklig et ytelsestestings-verktøy som kan brukes i OpenStack for å se hvor bra OpenStack skalerer, og om selve skyen fungerer. For å gjøre dette har Rally mange ferdiglagde templates som kommer med installasjonen. Disse templatene ruller ut scenarier som målt og testet. Administrator kan selv endre på templatene for å passe sammen med sitt OpenStack oppsett.

3.10 Nostrand

Nostrand er navnet som er blitt gitt til utrullingsserveren i sky-løsningen. Dette navnet er blitt brukt gjennom hele prosjektet, og det er derfor naturlig at dette navnet også er blitt brukt i rapporten. Derfor vil utrullingsserveren fra nå av bli kalt for Nostrand.

3.11 Instanser

Navnet på de virtulle maskinene som blir opprettet i OpenStack. Videre i rapporten blir virtuelle maskiner omtalt som instanser.

4 Design

Nostrand

I dette oppsettet er det Nostrand som har hovedrollen i systemet. Ved å bruke NAT skal Nostrand fungere som en ruter mellom et eksternt nettverk og det interne nettverket som opprettes. På det interne nettet skal Nostrand også fungere som en DHCP og DNS server. Det er Nostrand som skal rulle ut hele AutoStack løsningen, og til det brukes administrasjonsverktøyet Foreman. I Foreman lager man "verter" for hver node slik at de er klare til å få et operativsystem installert via PXE på det interne nettverket.

Når installasjonen av operativsystemet er fullført vil Puppet, som er en integrert del av Foreman, ta over og sørge for at OpenStack blir installert og konfigurert på de forskjellige nodene. Foreman sørger for at Puppet er installert på både Nostrand og alle nodene. Puppet agentene som kjører på nodene vil kontakte Puppet mesteren på Nostrand, i et satt tidsintervall, for å hente sine konfigurasjoner, og deretter utføre disse på noden. For å realisere dette brukes ferdiglagde Puppet moduler fra Puppetlabs og Stackforge som sørger for at OpenStack komponentene blir installert og konfigurert på nodene. Ved å bruke og tilpasse disse modulene vil et oppsett av OpenStack, som ellers er en manuell prosess, bli gjort automatisk. Mer om Puppet i kapittel 5.

I tillegg vil det bli laget et test-rammeverk på Nostrand som skal sørge for at utrullingen av skyen er gjennomført uten feil og om den er klar til bruk.

Roller i OpenStack

I gruppens scenario vil OpenStack bli rullet ut til fire servere. Oppdragsgiver ønsket at rollefordelingen på serverne skulle bli en kontroller-node, en nettverks-node og to compute-noder. Disse vil sammen utgjøre en operativ OpenStack skyløsning.

Controller

Kontroller-noden er ansvarlig for å administrere skyen. Her vil følgende OpenStack komponenter installeres: Nova, Cinder, Glance, Keystone, Heat, Neutron og Horizon. I tillegg kommer RabbitMQ og MySQL som skal støtte oppunder OpenStack komponentene.

Network

Nettverks-noden skal fungere som en ruter for instansene dersom de ønsker å kommunisere ut til et eksternt nettverk. Den skal også være en DHCP-server for de virtuelle nettverkene som blir opprettet i skyen. For å sørge for at disse nettverkstjenestene kommer på plass må OpenStack-komponenten Neutron installeres.

Compute

Compute-nodene er "arbeidshestene" i løsningen, det er på disse serverne at brukernes VM-er blir laget

av tilgjengelig maskinvare som Compute-nodene stiller med. Nova og Neutron er komponentene som blir installert her.

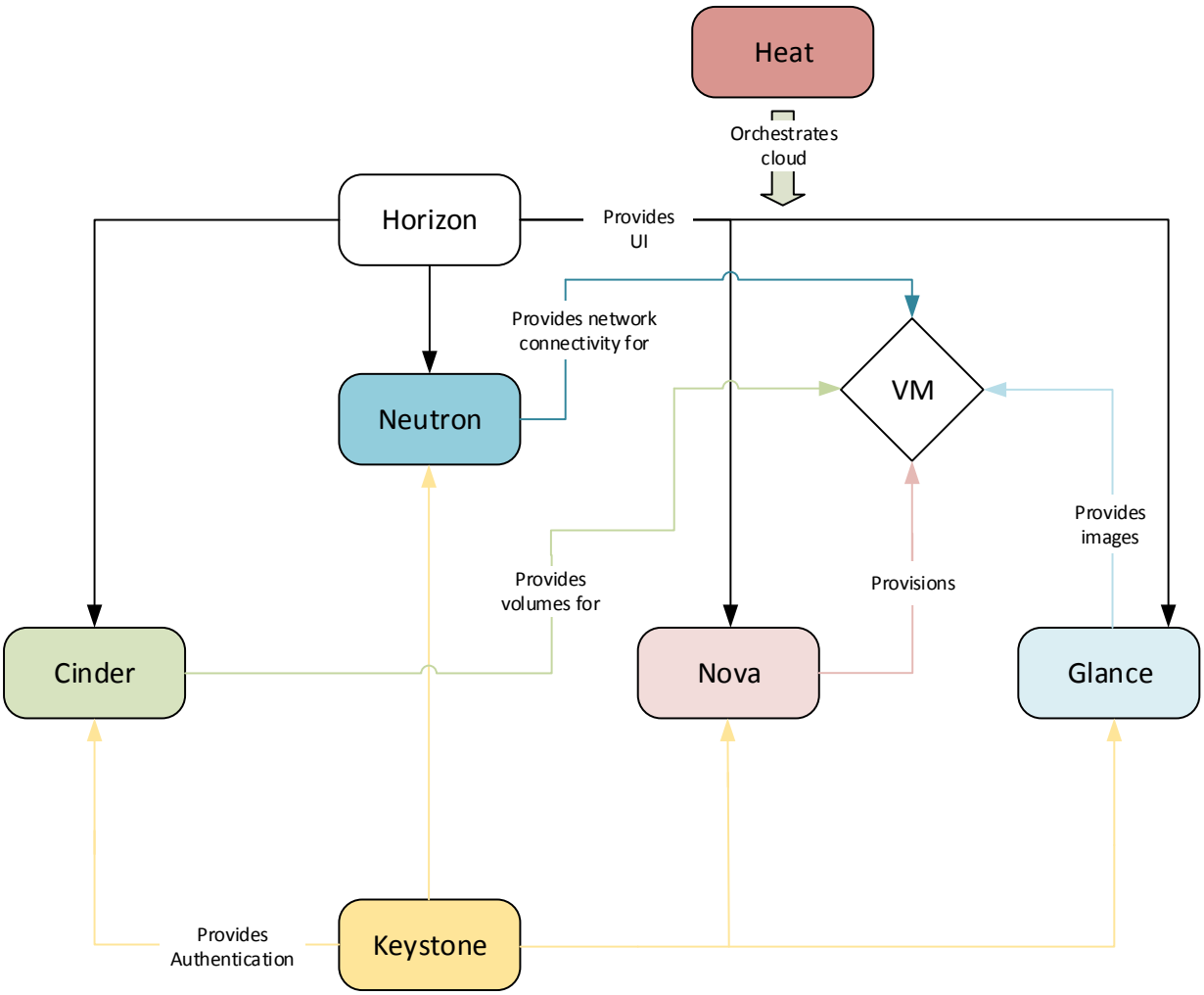


Figure 3: Design av de innvendige komponentene til OpenStack.

Fig. 3 viser forholdet mellom de ulike OpenStack komponentene. Se “Teoretisk grunnlag” kapittelet for beskrivelse av hver komponent.

5 Implementering

Implementering er det kapittelet som omhandler installasjonen av Nostrand i denne oppgaven. Her blir det forklart hva som er gjort for å sette opp skyløsningen. Siden kapittelet først og fremst viser hva man må gjøre for å komme frem til den løsningen som er gjeldende, er den rettet mot de som skal sette opp en slik løsning senere. Derfor vil det være en del kode og kommandoer i rapporten, men dette gjenspeiler også hvor mye gruppen har gjort kodemessig.

Nostrand blir installert med Ubuntu Server 14.04 LTS. Ubuntu Server er blitt valgt fordi utrullingsserveren i fjorårets AutoStack-oppgave ble også satt opp med Ubuntu Server. Det er derfor en fordel at man bruker samme Linux distro slik at dokumentasjonen fra AutoStack-oppgaven kan gjenbrukes i mest mulig grad. Ubuntu Server 14.04 er også en LTS som står for "Long Term Support". Dette er en garanti for at Ubuntu Server 14.04 vil motta stabilitetsoppdateringer og sikkerhetsoppdateringer i fem år fra utgivelsesdato.

5.1 Nostrand

5.1.1 Oppsett av DHCP på Nostrand

For å få tilgang til det interne nettet må nettverkskortene konfigureres riktig. Nettverksinnstillingene skal settes opp slik at det nettverkskortet som vender mot det eksterne nettverket mottar en IP-adresse automatisk, og det nettverket som vender mot skyløsningen har en statisk konfigurert IP-adresse. Dette gjøres ved å endre filen /etc/network/interfaces som vist nedenfor:

```
auto eth1
iface eth1 inet dhcp

auto eth2
iface eth2 inet static
address 10.0.0.1
netmask 255.255.255.0
network 10.0.0.0
broadcast 10.0.0.255
```

For å installere et operativsystem via PXE, kreves det at DHCP er aktivert. Derfor er det behov for et lukket nettverk hvor DHCP skal settes opp. Nostrand har i dette tilfellet to nettverksporter, ett for det eksterne nettverket og ett for det interne nettverket, hvor nodene skal rulles ut. På Nostrand installeres det Internet Systems Consortium sin DHCP server, siden denne er brukt i installasjonsguide på Foreman sine hjemmesider:

```
apt-get update
apt-get install isc-dhcp-server
```

Her er nettverkskortene satt opp slik at eth1 mottar en IP-adresse fra det eksterne nettet, mens eth2 er konfigurert med statiske IP-innstillinger og vil være koblingen mot det interne nettverket der den forsyner servere og maskiner med IP-adresser. For å starte det interfacet som skal tjene det interne nettverket, kjøres denne kommandoen:

```
ifup eth2
```

Det må gjøres endringer i /etc/dhcp/dhcpd.conf og /etc/default/isc-dhcp-server for at DHCP serveren skal bli satt opp riktig. Her settes det opp hvilket nettverk og subnet som skal brukes, i tillegg til ekstra informasjon slik som bootp-protokollen.

Dette oppsettet er hentet fra Foreman sin dokumentasjon , og tilpasset til oppsettet som brukes i oppgaven. Nederst i filen /etc/default/isc-dhcp-server settes det hvilket interface DHCP-serveren skal håndtere DHCP trafikk på. Dette er viktig å sette til det interne nettverksinterfacet, for hvis dette settes til feil nett, så vil alle enheter som trenger tilgang til nettet få svar fra feil DHCP-server. I dette tilfellet er det interne nettet eth2. Dette kan sjekkes med kommandoen ip a.

Dermed gjenstår det bare å starte DHCP tjenesten med denne kommandoen:

```
service isc-dhcp-server start
```

5.1.2 Oppsett av NAT

Nostrand må fungere som en ruter mellom det eksterne og det interne nettverket. Dette er viktig, siden utrulling skal skje på samme metode uansett plassering. Derfor må det aktiveres NAT slik at serverne på det interne nettverket får tilgang til omverdenen. Ubuntu Server 14.04 kommer ikke med dette aktivert som standard, så dette må gjøres manuelt. Iptables er brannmuren i Ubuntu et er lagt til følgende regler i iptables for å aktivere NAT:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth1 -o eth2 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -j ACCEPT
```

Den første regelen aktiverer NAT, mens de to andre linjene sørger for at pakker blir videresendt fra det interne til det eksterne nettverket, og motsatt. IPv4 videresending må også aktiveres for å bruke Nostrand som en ruter. I filen /etc/sysctl.conf, settes net.ipv4.ip_forward til 1:

```
net.ipv4.ip_forward=1
```

For at dette skal aktiveres må kommandoen sysctl -p /etc/sysctl.conf kjøres.

IP-reglene blir tilbakestilt hver gang Nostrand startes på nytt. Derfor er det laget et skript som legger tilbake disse innstillingene hver gang dette skjer. Først kjøres kommandoen:

```
iptables-save > /etc/iptables.rules
```

som lagrer innstillingene. Deretter lages et skript “startopp.sh” som legges i /etc/init.d/. Ved å legge skriptet i /etc/init.d/ så vil skriptet kjøre ved oppstart. Skriptet inneholder én linje:

```
iptables-restore < /etc/iptables.rules
```

Dette skriptet gjenoppretter de lagrede innstillingene hver gang det kjøres. DHCP-serveren og generell ruting skal da være operativt.

Kommandoen “route -n” gir oss en utskrift på hvordan rutingen til Nostrand ser ut:

```
Kernel IP routing table
Destination         Gateway       Flags     Iface
128.39.142.0/24     128.39.142.1    U       eth1
10.0.0.0/24         10.0.0.1        U       eth2
0.0.0.0/0           128.39.142.1    UG      eth1
```

5.1.3 Oppsett av DNS

DNS må installeres på Nostrand fordi Puppet bruker domenenavn for å finne konfigurasjonsserveren sin. Det blir derfor installert BIND9 som DNS-server, fordi den bl.a. er støttet og dokumentert i Foreman sine nettsider. Denne programvarepakken er det Internet Systems Consortium som vedlikeholder. Denne programvarepakken installeres med følgende kommando:

```
apt-get install bind9
```

For oppsett av DNS på Ubuntu er en guide fra askubuntu.com benyttet.

De fleste innstillinger ved denne tjenesten blir gjort i mappen /etc/bind/.

/etc/dhcp/dhclient.conf: Denne filen inneholder konfigurasjonen til den lokale DHCP-klienten. Denne legger informasjonen sin inn i den lokale dnsmasq-serveren, som Ubuntu bruker bl. a. som mellomlagring når den skal se etter navneservere. Disse valgene overstyrer hvor maskinen skal se etter navn, og gjør det dermed mulig å koble direkte til serverene uten å skrive inn en IP-adresse:

```
supersede domain-name "auto.stack";
prepend domain-name-servers 127.0.0.1;
```

/etc/apparmor.d/usr.sbin.named: AppArmor er et sikkerhetssystem i Linux som håndterer tilgang til filer på et applikasjonsnivå. Denne tjenesten passer på at ingen programmer skal få tilgang til noe de ikke

har rettigheter til. AppArmor hindrer Bind i å gjøre endringer på konfigurasjonsfilene sine, det må derfor endres en innstilling i konfigurasjonsfilen til AppArmor:

I filen /etc/apparmor.d/usr.sbin.named skal det endres én linje:

```
/etc/bind/** r,
til
/etc/bind/** rw,
```

Testing av DNS gjøres til slutt ved å kjøre kommandoen nslookup:

```
root@nostrand:/home/nostrand# nslookup nostrand
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   nostrand.auto.stack
Address: 10.0.0.1

root@nostrand:/home/nostrand# nslookup 10.0.0.1
Server:         127.0.0.1
Address:        127.0.0.1#53

1.0.0.10.in-addr.arpa   name = gateway.auto.stack.
1.0.0.10.in-addr.arpa   name = nostrand.auto.stack.

root@nostrand:/home/nostrand#
```

5.2 Foreman

5.2.1 Installasjon av Foreman

Installasjon av Foreman gjøres gjennom et eget installasjonsprogram. Det lastes ned fra den offisielle nettsiden og kjøres i kommandolinjen. Versjon 1.7 installeres siden dette er den seneste stabile versjonen av Foreman. For å laste ned Foreman-installasjonen kjøres følgende kommando:

```
apt-get update && apt-get -y install foreman-installer
```

Etter at dette er gjort så skal selve installasjonen kjøres. Her er det viktig at kommandoen

```
ping $(hostname -f)
```

viser en ekstern ip-adresse, ikke 127.0.0.1 og at interfacet til det interne nettet er oppe og fungerer. Puppet bruker domenenavn for å finne konfigurasjonsserveren sin, og uten dette vil ikke oppsettet fungere. Dersom kommandoen viser 127.0.0.1 er det bare å endre innholdet i /etc/hosts, slik at det vises riktig. Dette er dekket i installasjonsguiden.

Da alt stemmer, skal det bare være å kjøre kommandoen

```
foreman-installer
```

så installeres Foreman på serveren. Denne prosessen er automatisk, og sier i fra når den er ferdig med å kjøre. Den vil også skrive ut den initielle administrator-brukerkontoen med passord til slutt, så terminalutskriften er viktig å få med seg. Dette passordet kan endres senere i web-grensesnittet til Foreman.

5.2.2 Konfigurasjon av Foreman-proxier

Foreman kommer med det meste ferdigkonfigurert, men noen ting mangler for å få distribusjonssystemet helt automatisert. Dette er smart-proxyer. Disse håndterer nettverkstjenester, slik at det er minst mulig manuell jobb med konfigurering av disse tjenestene ved utrulling.

Her støtter Foreman både TFTP, DNS, DHCP, Puppet og Puppet CA (sertifikatshåndtering). Ved installasjon er det bare TFTP, Puppet og Puppet CA som er aktivert. DHCP og DNS må aktiveres manuelt. Dette gjøres gjennom en dedikert tjeneste; foreman-proxy. I mappen /etc/foreman-proxy/settings.d/ ligger alle innstillingene for denne tjenesten. Det er her de konfigureres og settes opp til de ulike funksjonene. Der aktiveres DNS og DHCP. Når dette er gjort, er det bare å gå på fanen “Infrastructure -> Smart proxies” inne i Foreman sin nettside og trykke på “Refresh features” ved siden av Nostrand for å oppdatere listen, så skal de nye tjenestene dukke opp.

DHCP er nødvendig å håndtere fra Foreman. Siden det er DHCP-serveren som sier ifra hvor noden skal lete etter filer for PXE-booting, er det viktig at Foreman får styre dette til sine filer. TFTP har Foreman allerede installert gjennom sin egen installasjonsprosess.

DNS er viktig for Puppet sin del, siden det er den eneste metoden som brukes for å finne serveren den skal hente konfigurasjonen sin fra og sende rapporter tilbake til. Dette kan settes statisk i hver eneste node sin nettverkskonfigurasjon, men dette vil ikke være skalerbart, og ikke minst modulært. Å styre DNS gjennom utrullingen til Foreman viser seg derfor å være nødvendig for enklest mulig utrulling av systemet, og derfor er det lagt til støtte for det i denne oppgaven.

Name	URL	Features	
nostrand.auto.stack	https://nostrand.auto.stack:8443	TFTP, DNS, DHCP, Puppet, and Puppet CA	Certificates

Figure 4: Smart Proxies.

5.2.3 Puppet management

Etter at installasjonen av Foreman er ferdig skal det være satt opp en Puppet-mester på Nostrand, som er fullt integrert med Foreman. For å sende den første statusrapporten til Foreman må man først kjøre Puppet kommandoen:

```
puppet agent --test
```

Når kommandoen kjører vil det komme opp en advarsel første gangen den kjøres. Den sier ifra at den ikke finner noden. Denne meldingen forsvinner andre gangen man kjører kommandoen, og varselmeldingen kan derfor bare ignoreres i følge Foreman sine hjemmesider.

Når en vert har kjørt Puppet uten problemer, vil den dukke opp under fanen 'Hosts –> All Hosts' i Foreman sitt grensesnitt med statusen "O". Dette viser at verten er operativ, og at ingen endringer er blitt gjort siden siste statusrapport.

5.2.4 Klargjøre utrulling i Foreman

Nå som Foreman sin struktur er satt opp, skal alle innstillinger settes inne i web-grensesnittet. Dette er dekket i installasjonsguiden, og finnes også på Foreman sine hjemmesider. Det meste er pek-klikk, og gjør det initielle oppsettet enklere for nye oppsett av noder.

PXE-miljø

PXE blir brukt på det interne nettverket for installering av nye noder uten noe interaksjon fra brukeren. Foreman styrer dette, og bruker TFTP for overføring av de viktigste oppstartsfilene. Tilgangen blir styrt gjennom verifisering av MAC-adresse, slik at ingen maskiner starter opp gjennom PXE ved uhell.

PXE-boot

De serverene som er tildelt har en annerledes RAID-kontroller enn vanlige PCer. Siden den navngir harddisker og partisjoner på en helt spesiell måte, må det konfigureres flere funksjoner slik at de fungerer med serverene.

Under installeringen av en ny node bruker Foreman forskjellige templates for å konfigurere serveren. De fleste standard-templatene blir brukt, men en av de må endres: "Preseed default" under "Partition tables". Her må linjen "d-i partman-auto/disk string /dev/sda /dev/vda" endres til "d-i partman-auto/disk string /dev/sda /dev/vda /dev/cciss/c0d0". Mer om dette i [link/referanse til utfordringer /dev/cciss, testing].

Oppstart etter installasjon av OS

Når OSet er installert, vil ikke serverene starte opp fra harddisk. Dette er fordi at filen /var/lib/tftpboot/pxelinux.cfg/<MAC-addresse> ikke fungerer med disse (en fil for hver vert/node). Dette er filen som gis til serveren etter en installering, da den skal starte fra harddisk i stedet.

Tidligere innhold:

```
        LOCALBOOT 0
```

Slik løsningen er for våre servere:

```
    COM32 chain.c32
    APPEND hd0 0
```

Siden dette er noe Foreman styrer, kan dette lett endres inne i web-grensesnittet, uten å dykke dypere inn i konfigurasjonsfiler. I templaten 'PXELiunx localboot' inne i Foreman legges denne koden til:

```
      COM32 chain.c32
<% if @host.shortname == 'DL 320' || @host.shortname == 'Dl 320 2' || @host.shortname ==
    'DL 320 3' -%>
      APPEND hd0
<% else -%>
      APPEND hd0 0
<% end -%>
```

Denne koden skal sørge for at de to ulike HP serverene DL380 og DL320 skal klare å boote automatisk etter installering av OS. Den fungerer også på andre maskiner, da det er blitt testet flere forskjellige maskiner i produksjon. Den eneste forskjellen mellom de to er: "APPEND hd0 0" og "APPEND hd0". Forklaringen på dette kan være så enkel som at de to serverene bruker forskjellige RAID-konfigurasjoner. Gruppen har ikke klart å finne ut av hvorfor det er slik.

5.3 Puppet

For å få til automatisk konfigurasjon av serverne bruker Foreman verktøyet Puppet. Puppet er et konfigurasjonsstyringssystem som lar gruppen definere en tilstand på sin infrastruktur, for deretter å sørge for at dette automatisk blir konfigurert og vedlikeholdt utover alle servere og maskiner. Puppet bruker et deklarativt språk for å beskrive konfigurasjonen til en maskin, dvs hvordan tilstanden til en maskin skal være, i stedet for hvordan det skal gjøres.

5.3.1 Mester og agent

Med Puppet har man et mester-agent forhold. I gruppens løsning blir Puppet-mester installert på Nostrand, mens Puppet-agentene blir installert på nodene. Det er Puppet-mesteren som besitter konfigurasjonen som skal tas i bruk på de forskjellige nodene. I et fast tidssintervall tar hver node kontakt med Puppet-mesteren for å se om de er konfigurert i forhold til det som er spesifisert på mesteren. Dersom det er spesifisert nye konfigurasjoner på mesteren, tar Puppet-agenten og konfigurerer noden automatisk til nye innstillinger angitt av mesteren.

5.3.2 Puppet moduler og manifester

Puppet koden blir lagret i tekstfiler kalt manifester med filendelser .pp. I Puppet koden deklarerer man ressurser.

```
class { 'ssh::server':
  options => {
    'PermitRootLogin' => 'yes',
  }
}
```

Denne koden sørger for at gruppen kan koble til nodene med SSH fra Nostrand. Her deklarerer vi ressurser 'PermitRootLogin' med verdien 'yes'. Det denne koden gjør er å gå inn i filen /etc/ssh/sshd_config, og endre verdien 'PermitRootLogin' fra et standard 'without-password' til 'yes'. Alt gjøres automatisk av Puppet. Mange slike deklareringer kan settes inn i en klasse slik det er vist ovenfor.

Puppet moduler er en samling av manifest filer, sortert slik at de blir automatisk lokalisert og lastet inn av Puppet-mesteren. Moduler kan inneholde mange klasser. For eksempel, en Apache modul kan ha en klasse som installerer og får Apache til å starte opp, mens en annen klasse aktiverer PHP i Apache.

Man kan lage egne moduler, men det finnes mange ferdiglagde Puppet-moduler som man kan laste ned og bruke gjennom Puppet Forge. Det er ferdiglagde Puppet-moduler fra PuppetLabs og StackForge som gruppen benytter seg av i dette prosjektet. Ved å bruke disse modulene installeres og konfigureres OpenStack automatisk på serverne.

5.4 Oppdatering til OpenStack IceHouse

AutoStack systemet ble i fjor satt opp med OpenStack-versjonen Havana. Det var derfor ønskelig at gruppen skulle oppgradere AutoStack til neste versjon ved navn IceHouse. Fra AutoStack sin bacheloroppgave fulgte det med Puppet-kode og en mengde skript som skulle rulle ut OpenStack automatisk. Fjorårets gruppe benyttet seg av ferdiglagde Puppet-moduler fra Puppetlabs [link] for å sikre en automatisk utrulling av OpenStack. Koden og skriptene ble hentet ut fra oppgaven deres, og tilpasset til oppsettet med IceHouse.

5.4.1 Installasjon av moduler

Det første som ble gjort var å installere IceHouse-versjonen av Puppetlabs-OpenStack modulene på Nostrand: *(Puppetlabs sine OpenStack-moduler stoppet opp med versjonen IceHouse, og vil sannsynligvis alltid være gjeldende.)*

```
puppet module install puppetlabs-ntp
puppet module install puppetlabs-nova
puppet module install puppetlabs-neutron
puppet module install puppetlabs-cinder
puppet module uninstall --force puppetlabs-rabbitmq
puppet module install puppetlabs-rabbitmq --version 2.1.0
puppet module uninstall --force puppetlabs-mysql
puppet module install puppetlabs-mysql --version 0.9.0
puppet module install puppetlabs-heat
puppet module install saz-ssh --version 2.4.0
```

5.4.2 Puppet-kode og deploy-skript

Puppet har en standard lokasjon som kalles “main manifest”, der puppet-kode vil bli kjørt dersom den blir lagt til. Denne er lokasjonen er /etc/puppet/environments/production/manifests/site.pp. All Puppet-kode fra fjorårets oppgave kom fra deres site.pp. I tillegg hadde de tre bash-skript som ordnet konfigureringer Puppet-modulene ikke hadde funksjonalitet for. Disse skriptene blir kjørt fra Puppet-koden.

Dermed ble det neste steget å kopiere AutoStack sin fil site.pp som legges i /etc/puppet/environments/production/manifests/. De respektive utrullingsskriptene for nettverk, kontroller og compute-nodene skal lagres i /etc/puppet/modules/nova/files.

5.4.3 Tilpasning til ny OpenStack versjon

Ved innføring av nye OpenStack versjoner kreves det som regel en del endringer i konfigurasjonsfilene til de ulike OpenStack-komponentene for at en oppgradering skal kunne skje. Det forekommer både sletting, innføring og endringer av variabler, så for å få rullet ut dette automatisk til nodene må dette innføres som Puppet-kode i site.pp eller som bash-kode i skriptene. OpenStack legger ut gode brukerguider for hvordan brukere skal kunne oppgradere til en ny versjon[19][20]:

Framgangsmåten som er blitt brukt under prosjektet er at det settes i gang en manuell oppdatering først for å bekrefte at oppgraderingen fungerer, før vi begynner å implementere den automatisk. Her er et eksempel på en manuell konfigurering som må gjøres ved oppdatering til IceHouse fra Havana, i filen

glance-api.conf:

```
rpc_backend = rabbit
rabbit_host = 10.0.0.2
rabbit_password = SRVPW
```

Dette blir gjort automatisk ved bruk av Puppet-modulen og ser slik ut:

```
class { 'glance::config':
  api_config => { 'DEFAULT/rpc_backend' => { value => 'rabbit'},
        'DEFAULT/rabbit_host' => { value => '10.0.0.2'},
        'DEFAULT/rabbit_password' => { value => 'SRVPW'},
  }
}
```

Dersom endringen ikke støttes av Puppet-modulene, må dette legges til i deploy-skriptene.

En fullstendig oversikt over hvilke endringer som måtte gjøres finnes i installasjonsguiden

Noen tilleggsmodifikasjoner

- Alt av IP-adresser er endret så det skal passe til hvert enkelt miljø.
- I løsningen til AutoStack hadde de med et overvåkningsverktøy som heter Munin og et test scenario som skulle rulles ut etter at løsningen var operativ. Etter samtale med oppdragsgiver ble det bestemt at disse ikke skulle være med i oppgaven, og de ble derfor fjernet.

5.5 Oppdatering til OpenStack Juno

Etter en suksessfull utrulling av OpenStack IceHouse, var det på tide å begynne med oppdateringen til Juno. Denne skjedde på samme måte som med oppdateringen til IceHouse. Først ble det oppdatert manuelt med hjelp av OpenStack sin offisielle guide. Denne inneholder alle de innstillingene som skal endres på, og ble fulgt til punkt og prikke. Når dette var gjort og alle småfeil var rettet på, var det neste steget å få dette inn i Puppet-kode.

5.5.1 Installasjon av moduler

Fra IceHouse til Juno har de som skriver[1] og vedlikeholder pluginene byttet navn[2]. Dette medfører at det må lastes inn nye moduler for bruk sammen med site.pp Det nye navnet disse går under er Stackforge. De har byttet navn fordi de egentlig ikke er en offisiell del av PuppetLabs.

Disse modulene må installeres for Juno:

```
puppet module install puppetlabs-ntp
puppet module install saz-ssh
puppet module install stackforge-nova
puppet module install stackforge-neutron
puppet module install stackforge-heat
puppet module install puppetlabs-rabbitmq --version 2.1.0 --force
puppet module install puppetlabs-mysql --version 0.9.0 --force
```

Selv om disse modulene er offisielle for Juno, ble det oppdaget at de hadde en del mangler. Puppet-modulene måtte modifiseres for at de skulle riktige i forhold til OpenStack sine oppgraderingsguider. Etter en prat med oppdragsgiver om Puppet-modulene skulle endres, ble svaret at modulene skulle lastes ned direkte fra Github. Det må fremdeles installeres fra Puppet Forge først, siden det er viktig å få med seg avhengigheter til andre moduler.

Puppet moduler som installeres legges i /etc/puppet/modules.

```
cd /etc/puppet/modules
rm -rf nova neutron glance heat cinder
```

Så installeres Git som brukes for å hente siste oppdatering fra Github:

```
apt-get install git

cd /etc/puppet/modules/
git clone https://github.com/stackforge/puppet-nova.git nova
git clone https://github.com/stackforge/puppet-neutron.git neutron
git clone https://github.com/stackforge/puppet-glance.git glance
git clone https://github.com/stackforge/puppet-heat.git heat
git clone https://github.com/stackforge/puppet-cinder.git cinder
```

5.5.2 Tilpasning til ny OpenStack versjon

Her ble samme prosedyre fulgt som under IceHouse installasjonen. En manuell installering ble gjort, før det ble ordnet en helautomatisk løsning.

I tillegg har det blitt lagt til funksjonalitet på eget initiativ. I begynnelsen av site.pp er det blitt lagt til variabler som brukes ofte gjennom hele filen. Dermed kan brukere av systemet bare endre variablene øverst i site.pp for at endringene skal bli gjennomført gjennom hele filen. Denne funksjonaliten er tidsbesparende siden brukeren bare trenger å endre variabelen ett sted, i stedet for å endre den gjennom hele filen. En annen ting som er implementert er at alle klasser med noe til felles, er blitt dratt sammen i en klasse nest øverst i site.pp, og deretter kalt individuelt. Dette er blitt gjort for å minske duplisering av kode.

Etter en suksessfull utrulling av OpenStack Juno, får man denne oversikten i Horizon:

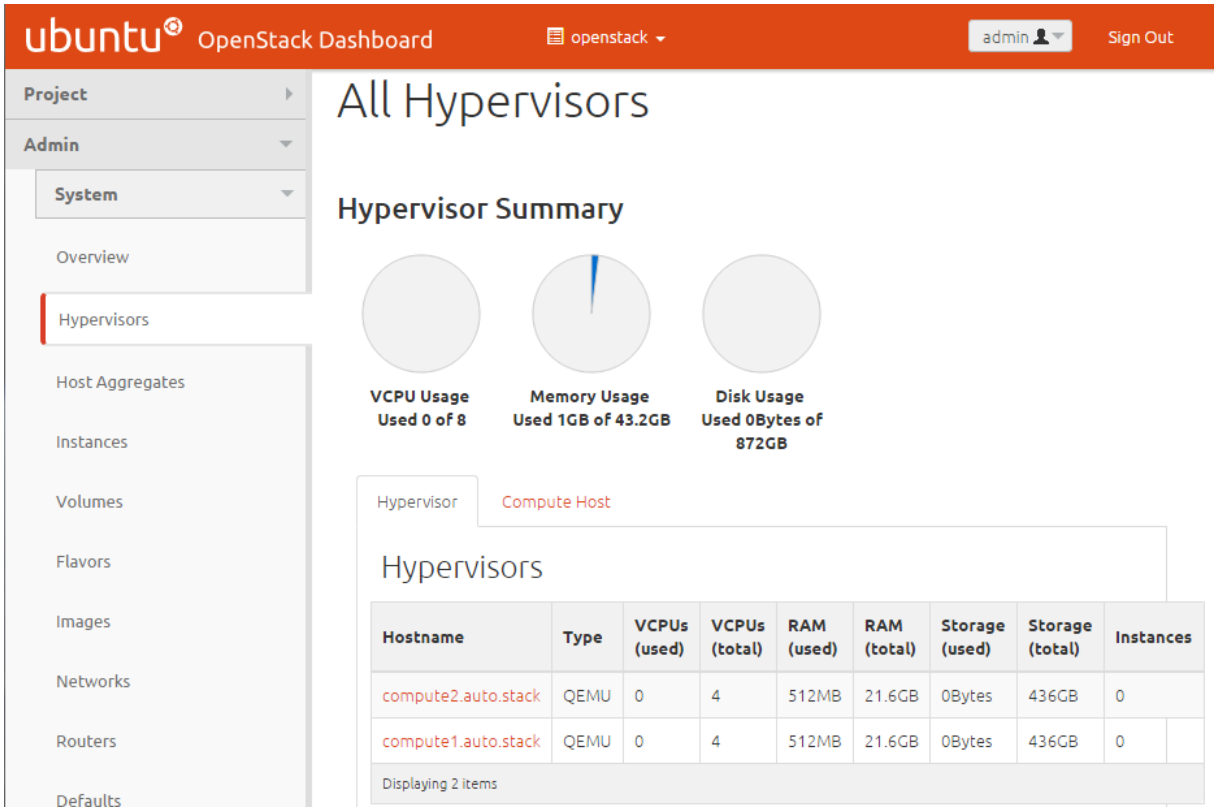


Figure 5: Et skjermbilde fra web-interfacet Horizon.

5.5.3 Konvertere skript-filene til Puppet kode

Oppdragsgiver ville at gruppen skulle gå vekk fra bruk av skript og prøve å implementere disse i site.pp, slik at det bare blir én fil å forholde seg til. Dette vil gi en bedre oversikt over all kode og ikke minst bedre kontroll av når kommandoer skal kjøres. For å erstatte kommandoene inne i skriptet, kan man i Puppet kjøre terminal-kommandoer ved å bruke exec-"ressurser" og "command =>" atributtet. Ved å bruke denne metoden ble alle deploy-skript flyttet over i site.pp slik at det ikke lenger er noen skript-filer å forholde seg til ved installeringen av OpenStack.

Eksempel på kode fra deployCompute.sh:

```
sed -i 's/#net.ipv4.conf.all.rp\_filter\=1/net.ipv4.conf.all.rp\_filter\=0/g' /etc/sysctl
    .conf
sed -i 's/#net.ipv4.conf.default.rp\_filter\=1/net.ipv4.conf.default.rp\_filter=0/g' /etc
    /sysctl.conf
sysctl -p
```

Ovenfor vises et eksempel på noen linjer fra scriptet deployCompute.sh. Slik blir det når det flyttes over i Puppet kode:

```
exec { 'filtering':
    command => '/bin/sed -i "s/#net.ipv4.conf.all.rp\_filter\=1/net.ipv4.conf.all.rp\
        _filter\=0/g" /etc/sysctl.conf;\
      /bin/sed -i "s/#net.ipv4.conf.default.rp\_filter\=1/net.ipv4.conf.default.rp\
          _filter=0/g" /etc/sysctl.conf; \
      /sbin/sysctl -p',
    unless  => '/bin/grep "net.ipv4.conf.default.rp_filter=0" /etc/sysctl.conf',
  }
```

For å sikre seg mot at kommandoen blir kjørt flere enn en gang, brukes atributtet unless => for å sjekke om denne kommandoen har kjørt allerede. Dette gjøres, i eksempelet ovenfor, ved å sjekke om net.ipv4.conf.default.rp_filter=0 er satt i filen /etc/sysctl.conf. Dette gjør kommandoen idempotent; hvilket betyr at hvis endringen er funnet sted så vil ikke kommandoen kjøre på nytt. Dette er en nødvendighet siden konstante endringer uten grunn kan skape en ustabil OpenStack installasjon, og det vil bli unødvendig mye jobb å gjøre hver gang Puppet-agentene kjører på nodene.

5.6 Nettverk i OpenStack

I oppgaven er det fokusert på to nettverksteknologier når det kommer til å lage virtuelle nettverk, VLAN og GRE. Dette kapittelet dekker hva disse teknologiene er, og hvordan de er implementert i oppgaven.

OVS

Open vSwitch er en open-source plugin i OpenStack, og den fungerer som en virtuell switch. Open vSwitch sin konfigurasjon består av broer og porter. Porter representerer tilkoblinger til andre enheter, slik som fysiske nettverkskort og patche-kabler. Broer dukker opp som nettverks-interfaces på Linux, slik at de kan tilegnes IP-adresser.

GRE

Generic Routing Encapsulation (GRE) er en teknologi som er brukt i mange VPN-løsninger. Den pakker inn IP-pakker for å lage helt nye pakker med forskjellig ruting informasjon. Når den nye pakken kommer fram til sin destinasjon, blir den pakket ut, og IP-pakken blir så rutet videre. For at GRE skal fungere med Open vSwitch må Neutron lage "GRE tuneller". Disse tunellene er porter på en bro og gjør at broer på andre systemer blir koblet sammen. Instansene vil da kommunisere som om de var på samme lag 2 switch. Med GRE slipper man da å bruke en switch som må holde styr på en rekke VLAN.

VLAN

Virtuelle LAN gjør en modifikasjon i Ethernet headeren. De legger til en "VLAN tag" på 4 byte som går fra 1 til 4094. Noen switcher og rutere vet hvordan de skal håndtere "VLAN tags", og det gjør Open vSwitch også. Pakker som er merket for et VLAN blir bare delt med andre enheter som finnes på det VLAN-et, selv om alle enheter er på det samme fysiske nettverket.

5.6.1 Implementasjon av nettverk i oppgaven

I de switchene som ble brukt for å koble serverene i ett nettverk var det allerede konfigurert støtte for å bruke VLAN. Oppdragsgiver forklarte at det var dette nettverksoppsettet han brukte i sin SkyHiGh-løsning, og ville at AutoStack skulle bruke det samme. De portene som er brukt til virtuelle VLAN nettverk går fra VLAN id 2000 opp til VLAN 2099, og dermed brukes VLAN 2000 og VLAN 2001 i test-oppsettet til test-rammeverket når to test-nettverk blir satt opp.

I OpenStack.org sine installasjonsguider [link] brukes det GRE om nettverkstype. Dette var ikke noe problem i seg selv, men etter flere møter med oppdragsgiver og veileder, hvor det blant annet ble diskutert gre-nettverk i OpenStack, ble det enighet om at det skulle automatisk implementere begge deler i Puppet-koden. I IceHouse-installasjonen er det bare brukt VLAN, mens i Juno er det lagt til muligheten for at brukeren skal kunne velge mellom VLAN eller GRE som sin nettverkstype.

Det å ha muligheten til å bytte nettverkstype kan være praktisk dersom utrullingen skal skje et sted hvor man ikke har tilgang til å endre switchen sin konfigurasjon. I site.pp er det laget en egen variabel som inneholder enten "GRE" eller "VLAN". Hvis variabelen er "GRE" installeres det et GRE-nettverk, og hvis variabelen er "VLAN" installeres VLAN. Vær klar over at dette må bestemmes/settes før installasjonen av OpenStack begynner. Hvis man ombestemmer seg og allerede har et system oppe, må alle maskiner re-installeres på nytt for at endringen skal tre i kraft og fungere. Man kan ikke bare endre variabelen,

siden dette vil føre til korrupte innstillinger, og kan føre til datatap.

Eksempel på kode fra site.pp (spesifikt ml2 seksjonen):

```
# Network type (VLAN or GRE).
$networkType = "GRE"

if $networkType == 'VLAN' {
 vlan konfigurasjon...
}

if $networkType == 'GRE' {
 gre konfigurasjon...
}
```

5.6.2 VLAN oppsett med Open vSwitch i IceHouse og Juno

Denne figuren viser hvordan nettverket fungerer når det er satt opp til VLAN.

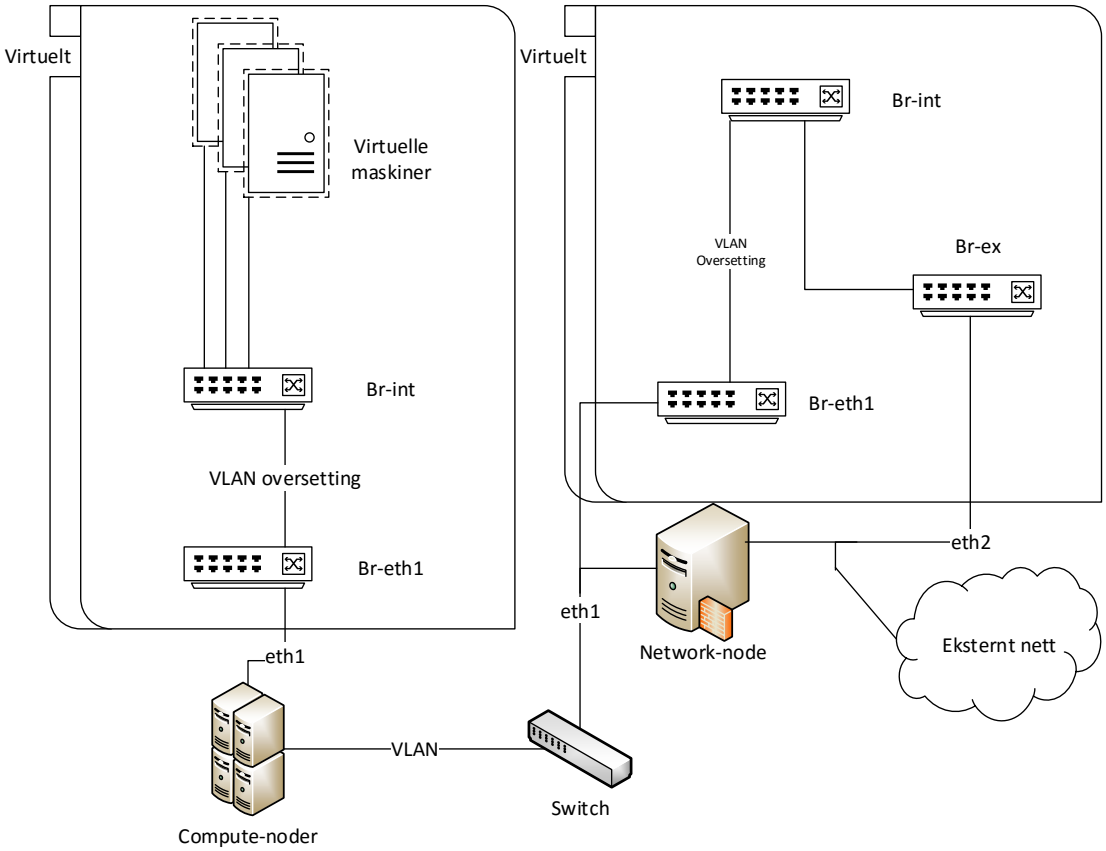


Figure 6: Oppsett med VLAN.

Compute-node konfigurasjon br-int

br-int er kort for “Integration bridge” og er broen som alle instanser på Compute-noden kobler seg til. Isolasjon mellom instansene sikres ved at porter på br-int konfigureres.

br-eth1

br-eth1 er broen som etablerer kontakt med det fysiske nettverkskortet eth1. br-eth1 er koblet til br-int som et såkalt “veth-par” (int-br-eth1, phy-br-eth1). Et “Veth-par” kan ses på som en patche-kabel mellom to virtuelle broer.

Nettverks-node konfigurasjon

br-int og br-eth1

Lagt til akkurat som på Compute-noden

VLAN oversetting

Mellom br-int og br-eth1 skjer en VLAN oversetting mellom den reelle VLAN ID og en intern VLAN ID. Dette holder Open vSwitch styr på ved hjelp av en intern database kalt OVSDB. Dette er for å sikre at IP-pakkene kommer seg dit de skal, og for å opprettholde sikkerhet.

br-ex

Denne broen kobler til det fysiske nettverkskortet som er koblet mot det eksterne nettverket, derav navnet ‘ex’(ekstern). For gruppen er dette eth2. br-ex er koblet til br-int som et “veth-par” (int-br-ex, phy-br-ex).

5.6.3 GRE oppsett med Open vSwitch i Juno

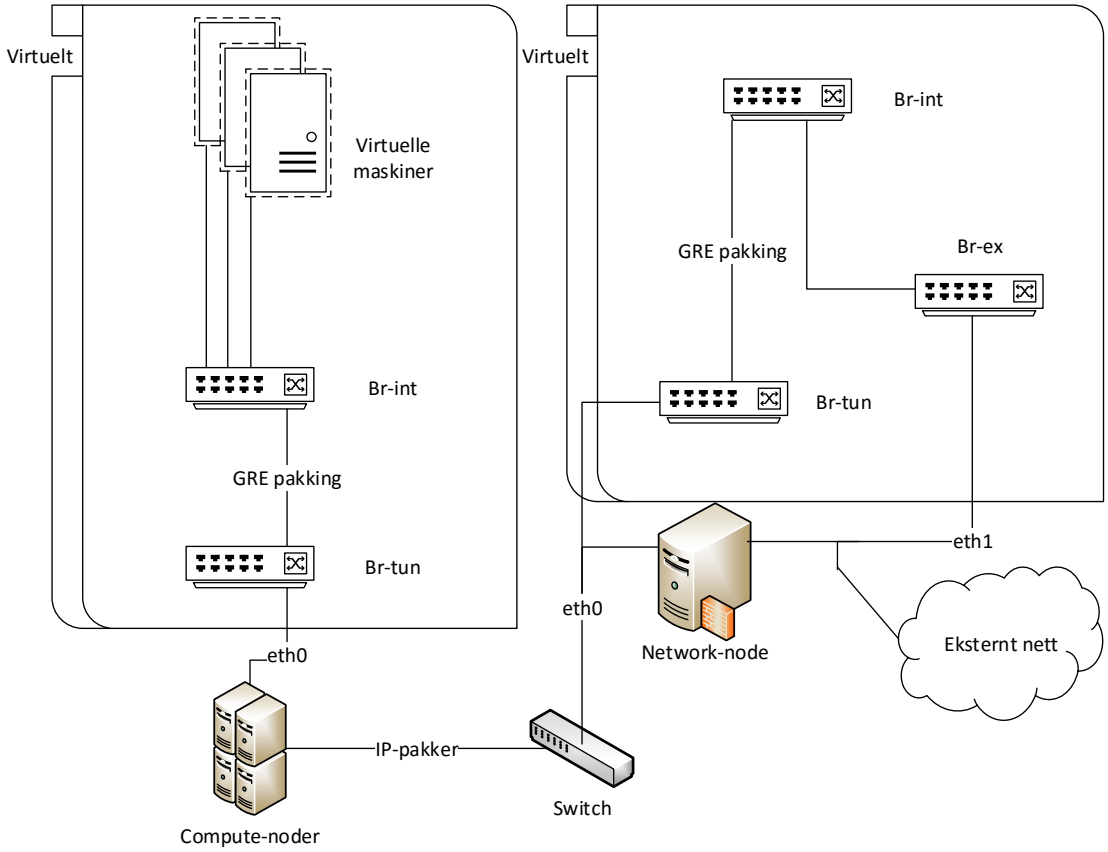


Figure 7: Oppsett med GRE.

Compute-node konfigurasjon

br-int

br-int er kort for “Integration bridge” og er broen som alle instanser på Compute-noden kobler seg til. Isolasjon mellom instansene sikres ved at porter på br-int konfigureres.

br-tun

br-tun er broen som etablerer kontakt med det fysiske nettverkskortet eth0. br-tun er koblet til br-int som et såkalt “veth-par” (patch-tun, patch-int). Et “Veth-par” kan ses på som en patche-kabel mellom to virtuelle broer. br-tun blir automatisk laget av Open vSwitch.

Netverks-node konfigurasjon br-int og br-tun

Akkurat som på Compute-noden er følgende bruer lagt inn:

br-ex

Denne broen kobler til det fysiske nettverkskortet som er koblet mot det eksterne nettverket. For gruppen er dette eth1. br-ex er koblet til br-int som et "veth-par" (int-br-ex, phy-br-ex).

DHCP agent

DHCP agenten bruker som standard en prosess kalt dnsmasq for å gi DHCP tjenesten til instanser. Det lages en intern port for hvert nettverk som krever DHCP tjenesten, og fester en dnsmasq prosess til den porten.

L3 agent

Gir instansene tilgang til eksterne nettverk. L3 agenten bruker interne porter for å implementere ruting, og er avhengig av at nettverks-noden ruter pakkene mellom nettverkskortene. IP forwarding må være aktivert på noden for at dette skal fungere. L3 agenten bruker iptables for å implementere "floating-IPs", som skal sørge for at NAT er fungerende.

Fra og med OpenStack Juno trenger man ikke å legge til br-int selv, dette blir gjort automatisk[21].

5.7 Fra site.pp til en egen AutoStack modul

Etter en demo av systemet for oppdragsgiver den 27. Mars, ble gruppen klar over at oppdragsgiver ville ha en annen løsning på systemet enn det gruppen hadde kommet frem til. I stedet for å bruke site.pp, ville oppdragsgiver ha en Puppet-modul. Man skal da fra Foreman kunne tildele nodene Puppet-klasser slik at man enkelt kan tilegne de forskjellige roller. Dette begynte gruppen på umiddelbart, og det var ikke så mange justeringer som skulle til for å få dette til å fungere:

Puppet moduler er bygget opp slik at hver modul har sin egen mappe inne i /etc/puppet/modules. Der igjen har de gjerne en mappe som heter "manifests" der det ligger .pp filer med Puppet-koden, stien blir da følgende for gruppens egen modul: /etc/puppet/modules/autostack/manifests. Gruppen tok all koden fra site.pp og delte den opp i forskjellige .pp filer. Dermed ble det opprettet en .pp fil for hver node: controller.pp, network.pp, compute.pp og storage.pp. I hver .pp fil lages en klasse som inneholder all Puppet-koden til den aktuelle noden.

Eksempel fra compute.pp:

```
class autostack::compute inherits autostack {
  include autostack::common
    // Puppet kode her
}
```

Koden over definerer en klasse autostack::compute som arver variabler fra autostack.pp, som igjen arver variabler fra params.pp. Den inkluderer autostack::common slik at den installerer koden som er felles for alle nodene.

Filen init.pp blir laget og alle nodene arver sine verdier herfra. init.pp arver igjen sine verdier fra params.pp som inneholder alle variablene som kan endres på. Dette gjør at man kan tilpasse installasjonen til det spesifikke oppsettet fra Foreman. Det er i tillegg laget en common.pp fil som inneholder kode som skal installeres på flere noder. Denne brukes da for å unngå duplisering av kode. Denne blir inkludert i hver "node-klasse" som blir definert i hver sin node .pp fil.

Filene blir da som følger:

init.pp

Lager klassen autostack og arver variablene fra params.pp

params.pp

Lager klassen autostack::params og definerer variablene som brukes for å tilpasse installasjonen.

common.pp

Lager klassen autostack::common som inneholder all felles Puppet-kode for nodene.

compute.pp, controller.pp, network.pp og storage.pp

Lager klassen autostack::*navn på node* og inkluderer koden i common.pp

Fordelene med å lage en modul er for det første at koden blir mer modulær og enklere å forholde seg til. OpenStack-nodene trenger ikke lenger å ha navn som stemmer overens med de som sto definert i site.pp. Nå kan brukeren legge til den Puppet-klassen som ønskes i Foreman, og rollen til noden blir definert derfra. For det andre kan variablene i params.pp endres via grensesnittet til Foreman med noe som kalles "Overriding". Dette betyr at en administrator kan gå inn i Foreman å endre variabelen som setter hvilken IP-adresse kontroller-noden har. Dette vil da bli gjeldene for hele OpenStack konfigurasjonen, slik at en slipper å endre dette manuelt i AutoStack-modulen.

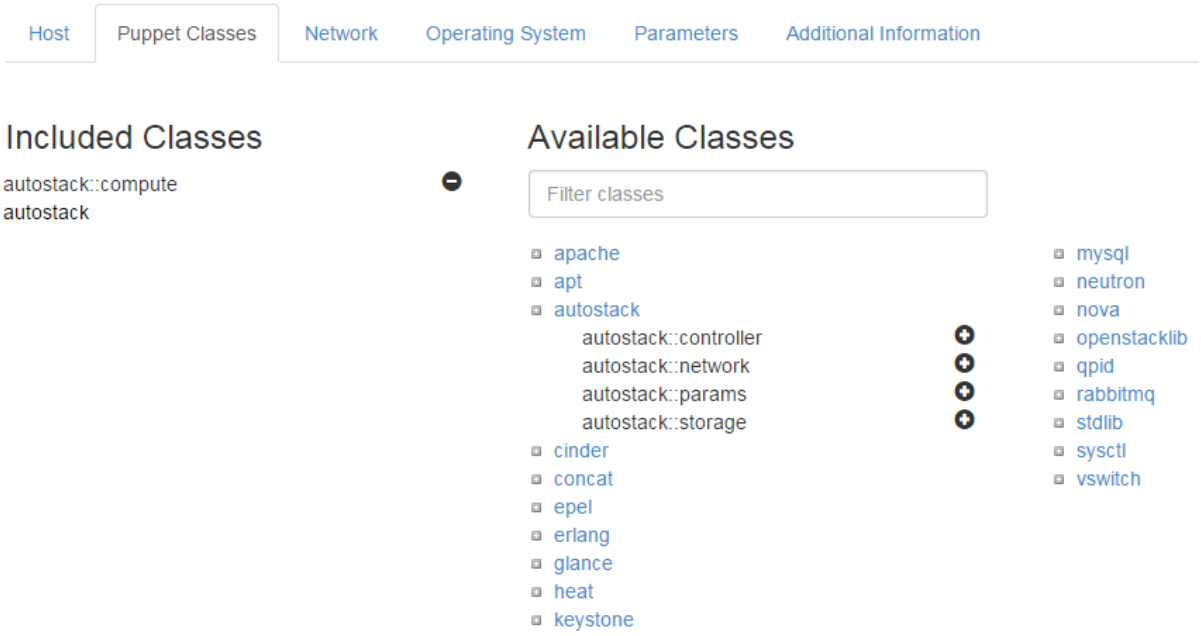


Figure 8: Her vises det en oversikt inne i Foreman over klassene.

5.8 Storage-node

Det var ikke noe krav fra oppdragsgiver at Cinder skulle implementeres, men den ble implementert på grunnlag av to ting. Den første grunnen er at test-verktøyet Rally krevde at Cinder var installert for å fungere. Den andre grunnen var i tilfelle "Digitalt øvingsfelt"-gruppen hadde behov for dette i sine scenarier. Gruppen valgte derfor å implementere Cinder inn i løsningen slik at man kan ha en eller flere storage-noder. Gruppen prøvde å installere en storage-node på en av gruppens servere med bare èn harddisk, men fant fort ut at dette ikke gikk. Standardpartisjoneringen fra Foreman tok hele disken, og det var ikke mulig å lage en ny partisjon som ble stor nok for lagring. Det gikk heller ikke å endre den eksisterende partisjonen. Gruppen fant dermed ut at de måtte ha en harddisk til, og lage partisjonen som skulle brukes på den harddisken for storage-noden.

5.8.1 Automatisk oppsett av storage-node

Gruppen laget en ny Puppet fil kalt "storage.pp" som plasseres i etc/puppet/modules/autostack/manifests/. Storage-noden trenger ikke noe mer enn SSH, NTP og Cinder konfigurasjon. Gruppen måtte gjøre noen få endringer i alle .pp filene i modulen, siden storage-noden ikke skulle kjøre noe annen Puppet-kode i common.pp enn SSH og NTP. I common.pp lagde gruppen egne klasser for SSH og NTP slik at man bare kunne bruke "include" kommandoen helt øverst i de andre .pp filene. Dermed ville ikke storage-noden kjøre hele common.pp koden, samtidig som de andre filene fortsatt kunne kjøre SSH og NTP koden.

Eksempel fra common.pp:

```
class { 'ssh-conf':
  # SSH konfigurasjon her
}

class { 'ntp-servers':
  # NTP konfigurasjon her
}
```

Eksemplet ovenfor viser den ekstra puppet koden som ble lagt til i common.pp. Klassen "ssh-conf" og "ntp-servers" var de klassene som ble lagt til for at storage-noden kunne få tilgang til både SSH og NTP.

Når LVM er installert på storage-noden, må LVM konfigureres til å ikke filtrere bort diskene som blir brukt for å lage fysiske volum. I filen /etc/lvm/lvm.conf i "devices" seksjonen er det en "filter" variabel som må endres på. "filter" variabelen står som default på at den ikke skal filtrere ut noe. "filter" variabelen måtte endres til å filtrere begge diskene (i gruppens tilfelle sda og sdb).

5.9 Test-rammeverk for skyløsningen

I følge oppgavebeskrivelsen skal gruppen implementere et test-rammeverk for selve sky-løsningen. Det første gruppen måtte ta stilling til, var om et slikt test-rammeverk fantes fra før og om det var noe gruppen kunne ta i bruk. I en epost fra oppdragsgiver i høst (se vedlegg X) ble det hintet til å se nærmere på Rally. I følge videoen er det egentlig bare to løsninger å velge mellom når det kommer til testing i OpenStack: Det er Tempest som er et offisielt OpenStack verktøy for testing, og et annet verktøy som heter Rally. Det man fort innser når man begynner gjøre research om disse verktøyene, er at dette er noe som er ganske nytt, og ikke så veldig utbredt. Det er vanskelig å finne dokumentasjon om begge verktøy, særlig Tempest. Gruppen finner fort ut med det vesle de har av informasjon om Tempest at det kommer til å bli veldig vanskelig å få brukt, til det gruppen ønsker. I følge denne linken[22] så er Tempest beskrevet som komplisert, vanskelig og tidskrevende å bruke. Det er derfor startet et OpenStack prosjekt som heter Rally og skal gjøre bruken av Tempest enklere samt sette fokus på ytelsestesting. Gruppen begynte å se på Rally siden dette var mye bedre beskrevet enn Tempest[23].

Oppsettet av Rally er beskrevet under:

Gruppen prøvde å installere Rally på kontroller-noden, men dette gikk ikke på grunn av at installasjonen skrev over python-filer som OpenStack var avhengig av. Rally ble derfor installert på Nostrand:

```
git clone https://git.openstack.org/stackforge/rally
./rally/install_rally.sh


nano deplyoment.json
```

deployment.json[24] inneholder:

```
{
    "type": "ExistingCloud",
    "auth_url": "http://10.0.0.2:5000/v2.0/",
    "region_name": "regionOne",
    "endpoint_type": "public",
    "admin": {
        "username": "admin",
        "password": "SRVPW",
        "tenant_name": "openstack"
    }
}
```

```
rally deployment create --filename=deployment.json --name=openstack
```

```
$ rally deployment check

keystone endpoints are valid and following services are available:

+----------+---------------+-----------+
| services | type          | status    |
+----------+---------------+-----------+
| glance   | image         | Available |
| heat     | orchestration | Available |
| keystone | identity      | Available |
| neutron  | network       | Available |
| nova     | compute       | Available |
| nova_ec2 | ec2           | Available |
| novav3   | computev3     | Available |
+----------+---------------+-----------+
```

Lager en enkel testfil: boot-and-delete.json med følgende innhold:

```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "CirrOS 0.3.3"
                },
                "force_delete": false
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 2
            },
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                }
            }
        }
    ]
}
```

```
rally task start boot-and-delete.json
```

Resultatet av denne testen viser hvor lang tid som blir brukt på å lage og slette noen VM-er.

Test som oppretter nettverk og instanser, kjører et script, og sletter alt etterpå.

```
{
    "VMTasks.boot_runcommand_delete": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "CirrOS 0.3.3"
                },
                "floating_network": "HIG",
                "force_delete": false,
                "script": "/root/rally/samples/tasks/support/instance_dd_test.sh",
                "interpreter": "/bin/sh",
                "username": "cirros"
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 2
            },
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                },
                "network": {
                }
            }
        }
    ]
}
```

Rally fungerte etter å ha blitt installert på Nostrand, men gruppen følte seg litt låst med tanke på hva som kunne testes, og fleksibiliteten på rapporteringen som det ble ønsket at test-rammeverket skulle gi brukeren. Dessuten viste det seg at Rally er et verktøy som først og fremst egner seg for ytelsestesting. Gruppen fant derfor ut at det var best å skrive et test-rammeverk fra bunnen av. Dette vil gi fullstendig kontroll over hva, hvordan, og når ting skal testes. I tillegg styrer gruppen helt selv hvordan rapportering av resultatene skal se ut.

5.10 Oppsett av eget test-rammeverk

Etter samtale med oppdragsgiver om hva test-rammeverket skulle omfavne, ble det klart at det var opp til gruppen selv å komme opp med kriteriene for dette. Et godt utgangspunkt da blir å se for seg scenariet: *Løsningen er satt opp hos en kunde, hva skal sjekkes før man er sikker på at skyløsningen er operativ?* Gruppen diskuterte seg i mellom og kom fram til følgende måte for å bekrefte en suksessfull utrulling av skyløsningen:

Hva som testes

Test-rammeverket setter opp et lite scenario i OpenStack der hver komponent og funksjon blir tatt i bruk og testet for å verifisere om de virker som de skal.

- Test-rammeverket lager først to virtuelle nettverk i OpenStack, et internt nettverk (testInternt) og et eksternt nettverk (testEksternt) med hvert sitt subnett. Det interne nettverket er et lukket nettverk mens det eksterne er tilkoblet det interne nettverket som Nostrand og OpenStack-nodene er tilkoblet. Deretter lages en ruter som kobles mellom de to virtuelle nettverkene som akkurat ble laget.
- En "Security rule" blir lagt til i brannmuren til OpenStack som tillater brukere å koble seg til instansene med SSH.
- Det opprettes SSH-nøkler der den offentlige nøkkelen blir lagt til i OpenStack slik at man skal kunne koble seg til instansene via SSH.
- En instans blir laget og denne kobler seg til 'testInternt' nettverket, og legger til SSH-nøkkelen som ble laget.
- En floating-IP blir allokert til instansen. Dette er en ledig IP-adresse fra 'testEksternt' nettverket. Det er denne IP-en nettverks-noden bruker i ruting mellom 'testInternt' og 'testEksternt'. Nettverks-noden sørger også for DHCP på 'testInternt' nettverket.
- Dersom det er satt opp en storage-node prøver skriptet å lage et volum som instanser kan koble seg til. Hvis en storage-node ikke er satt opp, vil skriptet hoppe over dette punktet.
- Deretter vil skriptet sende et eget test-skript over til instansen med SSH, der det skal kjøre å gi tilbake resultatene. Dette skriptet tester om instansen har tilgang til internett og dersom en storage-node er satt opp, vil den prøve å montere volumet som er laget på storage-noden.
- Det sjekkes så om Heat-komponenten fungerer ved å lage en stack med en heat-template fil der et scenario er gitt.
- Dersom alle stegene ble utført uten feil, begynner skriptet å slette alt den har opprettet, da i motsatt rekkefølge av det de ble satt opp i.

Hvis noe går galt underveis vil testen stoppe og skrive ut en feilmelding. Brukeren blir da bedt om å fikse problemet før man kan kjøre testen på nytt for å se om feilen har blitt rettet. Skriptet vil da hoppe over de stegene som den tidligere har utført og begynne der feilen inntraff sist. Dette vil sørge for at testen blir raskere fullført.

Hvordan det testes

For å teste at skyen har blitt rullet ut korrekt benyttes OpenStack kommandoer i et bash skript. Nedenfor er et eksempel fra test-rammeverket hvor det opprettes en ruter i skyen. Skriptet skriver først ut en tekst om hva som testes. Deretter sjekker skriptet om denne ruteren allerede har blitt laget, og skriver ut en melding dersom den finnes. Så kjøres kommandoen: "neutron router-create testRouter –tenant-id $id", dette er kommandoen som lager ruteren. Hvis kommandoen er vellykket skrives standard utskriften til /dev/null, noe som sletter utskriften, og blir erstattet med en melding om at ruteren ble laget. Dersom kommandoen ikke er vellykket vil kommandoens egen feilmelding bli skrevet ut i tillegg til en egen feilmelding som sier "Could not create router!", og skriptet stoppes slik at brukeren kan prøve å fikse problemet ut i fra feilmeldingen.
Eksempel fra testrammeverk.sh.

```
echo "### Trying to create router"
if neutron router-list | grep testRouter > /dev/null; then
  echo -e "<a style='color:blue'>INFO: </a>This router already exists!\n"
else
  id=$(keystone tenant-list | grep openstack | awk '{print $2;}')
  if neutron router-create testRouter --tenant-id $id > /dev/null; then
    echo -e "- Router created!                    <a style='color:green'>- OK!</a>\n"
 else
    echo -e "<a style='color:red'>ERROR:</a> Could not create router!\n"
    exit 1
 fi
```

Det brukes fem filer for at test-rammeverket skal fungere.

script.sh: Når brukeren starter testen fra Foreman er det dette scriptet som kjøres. Dette skriptet kobler seg til kontroller-noden via SSH, og sender over de fire filene som er listet opp nedenfor. Grunnen til at disse filene blir sendt til kontroller-noden er fordi Nostrand ikke kan kjøre OpenStack-kommandoene, siden den ikke har installert noen OpenStack-komponenter. Derfor må testen kjøres på kontroller-noden, siden denne har installert alle OpenStack-komponenter som er i bruk. Når alle filene har blitt sendt over, vil script.sh starte testrammeverk.sh på kontroller-noden. Derfra vil all utskrift fortsatt komme opp hos Nostrand siden dette blir kjørt over SSH, og alt blir lagret i filen /var/lib/foreman/public/testResultater/testrapport ved hjelp av denne kommandoen øverst i script.sh:

```
exec 3>&1 4>&2  1> /var/lib/foreman/public/testResultater/testrapport 2> /var/lib/foreman
    /public/testResultater/testrapport
```

Denne kommandoen flytter standard output til en fil i stedet for terminalen. Dermed kan rapporten vises gjennom nettleseren.

Nederst i script.sh ser man koden som starter testrammeverk.sh på kontroller-noden:

```
if ssh -i nokkel root@$controller "cd /tmp; ./testrammeverk.sh"; then
  echo -e "<a style='color:green'>\n\nThe test have run without errors and the cloud is
                    up and running!</a>\n"
else
  echo -e "\n\n<a style='color:red'>ERROR: </a>Something went wrong! Fix the problem  and
       run the test again!\n"
fi
```

Dersom testrammeverk.sh kjører uten feil på kontroller-noden så blir det skrevet ut nederst i rapporten at skyen er operativ og klar til bruk, men hvis det oppsto feil under testen, skrives det ut at det har oppstått et problem.

Følgende filer blir sendt over til kontroller-noden fra Nostrand:

testrammeverk.sh: Dette er skriptet som kjører alle OpenStack-kommandoene og finner ut om alt fungerer i skyløsningen. Eksempelet ovenfor [figur x] kommer fra testrammeverk.sh.

scriptToInstance.sh: Dette skriptet blir fra testrammeverk.sh, sendt over til en instans som har blitt opprettet og som også via testrammeverk.sh, blir kommandert til å kjøre på instansen. Dette enkle skriptet sjekker bare om instansen har tilgang til internett, og om den klarer å montere et storage-volum dersom dette eksisterer.

template.yml En Heat-template som inneholder et enkelt scenario for å teste om Heat fungerer i skyløsningen.

demo.sh: Inneholder identifikasjon som kreves for å kjøre OpenStack-kommandoene. Kommandoen source demo.sh kjøres på kontroller-noden før testrammeverk.sh begynner. Dermed slipper man å inkludere disse kriteriene i alle OpenStack-kommandoene i testrammeverk.sh.

Rapportering

Utskriften fra script.sh, som kjører på Nostrand, blir lagret i en egen rapport fil. Denne filen befinner seg i /var/lib/foreman/public/testResultater/testrapport. Når det gjelder hvilke meldinger som rapporteres tilbake til brukeren og hvordan, har gruppen hatt noen klare mål. Først og fremst måtte test-rammeverket gi gode tilbakemeldinger dersom noe galt inntreffer. Her bruker gruppen OpenStack-kommandoene sine egne feilmeldinger dersom noe feiler, siden disse gir gode tilbakemeldinger på hva som har gått galt, og av og til hva som må fikses for å rette problemet. Deretter er det viktig at det underveis skrives hva som blir testet, og hvordan det gikk i hvert steg av prosessen slik at det blir oversiktlig for brukeren hva som fungerer, og hvor feilen inntraff. Dersom en feil har inntruffet under en test, blir det neste gang skrevet ut meldinger om komponenter som allerede er på plass fra forrige gang testen kjørte.

Oppsett av test-rammeverk-nettside i Foreman

For å gjøre testingen av skyløsningen mest mulig brukervennlig har gruppen implementert test-rammeverket inn i Foreman. Ved å trykke på “Monitor” -> “Test-rammeverk” har test-rammeverket fått en egen side i Foreman der man kan starte testen ved å trykke på en knapp. Resultatet av testen vil da bli skrevet ut fortløpende på samme side. Siden er dynamisk og oppdaterer rapporten 4 ganger i sekundet, slik at man ser resultatet av testen i sanntid. Rapportfilen som genereres blir derfor laget som en html-fil.

Foreman sitt web-grensesnitt er basert på “ruby on rails”, som er en løsning hvor man bruker programmeringsspråket Ruby for å generere dynamiske nettsider. Dette gjør det litt annerledes å legge til en nettside, siden det må skrives en egen plugin for å vise den i web-grensesnittet. Gruppen har benyttet en veiledning[25] på hjemmesiden til Foreman der det er beskrevet hvordan en plugin skal lages.

Figure 9: Utskriften fra test-rammeverket

6 Utfordringer ved implementering

I løpet av prosjektet har det dukket opp en rekke problemer og utfordringer som gruppen har måttet løse. Dette kapittelet vil ta for seg de mest krevende utfordringene og hvordan gruppen løste de.

Feil ved partisjonering under installasjon av operativsystem

De serverene gruppen bruker har et hardware RAID som gjør at man kan koble sammen flere disker for redundans, hastighet eller en kombinasjon av dette. Denne trenger spesielle drivere fra operativsystemet, og Linux fikser dette ved å kalle diskene i denne navnekategorien: /dev/cciss/c0d0p0, hvor c står for kontroller, d står for disk og p står for partisjon.

Problemet med dette er at Foreman ikke støtter denne konfigurasjonen uten noen endringer. Ved installering av OS kom det bare opp en generell feilmelding. Inne i en såkalt "preseed" (Innstillingene som sendes med en server i det den installerer OSet fra bunnen av) ligger også partisjoneringsinnstillingene, og Foreman sender bare med standarddisker, som /dev/sda og /dev/vda som mulige harddisker.

Løsningen ble å endre på en standard partisjoneringstemplate som Foreman hadde innebygd. I partisjoneringstemplaten "preseed default" var det bare å legge til vårt system, og etter det fungerte installasjon av servere akkurat som det skulle. Dette er gjort i kapittelet implementering.

Virtualisering blir deaktivert ved strømbrudd til servere

All virtualisering i OpenStack går igjennom hardware, dvs. at serveren må ha hardware-støtte for å kunne kjøre nova-compute-tjenestene. Alle serverene våre hadde det, men av en eller annen grunn så fikk gruppen opp at hardware ikke var støttet av nova. Etter litt frustrasjon fant gruppen ut at VT-X (virtualiseringsstøtten for Intel-prosessorer) var skrudd av i BIOS. Da gruppen skrudde på den støtten, fungerte alt som det skulle.

Selv om gruppen skrudde på VT-X, fikk gruppen samme feilen neste dag, og gruppen gikk inn i BIOSen for å sjekke. Der viste det seg at batteriet som skulle holde innstillingene på plass var defekt, og derfor ble BIOS innstillingene tilbakesatt hver dag.

Servere vil ikke starte fra harddisk etter installasjon av operativsystem

Foreman har et godt fungerende system gjennom PXE hvor alle servere laster inn oppstartsekvensen sin over nettverket. Denne sjekker status fra Foreman om serveren er ferdig konfigurert, og hvis den er det så fortsetter den oppstarten fra harddisk i stedet. Dette fungerte ikke helt som det skulle. I stedet for å starte opp fra harddisk etter endt installering av operativsystemet, så stopper den opp på nettverksoppstartsskjermen. Den finner rett og slett ikke harddisken den skal starte fra. Gruppen mistenker at dette har en sammenheng med hardware-RAIDet, og etter litt leting på SYSLINUX (PXE-oppstartsløsningen som Foreman bruker) sine hjemmesider og litt prøving og feiling fant gruppen ut at kombinasjon hd 0 0 i stedet for localboot 0 fungerte på HP sine DL 380-servere.

Gruppen fikk det samme problemet når gruppen skulle installere OS på HP DL 320-serverne. De ville ikke fungere på noen av disse konfigurasjonsfilene, men ville fungere på hd 0. Dette blir det imidlertid en bedre løsning på i neste versjon av Foreman (1.8), for løsningen gruppen har nå baserer seg på navngiving av serverene, som både er upraktisk og ikke ville fungert like godt over flere utrullinger. Løsningen gruppen har på dette ligger i installasjonsguiden.

Konflikt ved bruk av to NTP moduler

Etter å ha fulgt installeringen av Foreman til punkt og prikke, fikk gruppen likevel feilmelding når gruppen skulle kjøre puppet-agenten på klientene. Det viste seg at den ntp-modulen gruppen installerte i forhold til Foreman-veiledningen skapte en konflikt med den ntp-modulen gruppen lastet ned etterpå.

Endring av navn på nettverkskort etter installasjon av operativsystem

Da endelig alle serverne var fungerende og kjørte puppet-agenten uten problemer, mistet to av dem tilkobling til internett. Dette var fordi ethernet port 1 ikke nødvendigvis ble til eth0 når man hadde ekstra nettverkskort. Det problemet sammen med at gruppen setter statisk IP på nettverksportene førte til at gruppen mistet kontakten og ikke fikk tilgang til nodene. Den midlertidige feilrettingen ble å endre på skript og puppet-konfigurasjoner slik at det ble satt opp riktige porter. Dette har gruppen i Juno fikset dynamisk gjennom Puppet, slik at uansett hvilken ethernet-port man starter opp i fra, så blir den satt til eth0 den første gangen maskinen starter opp og puppet kjører.

Utfordringer ved installering av OpenStack

Installering av IceHouse

Først prøvde gruppen å sette opp fjorårets AutoStack-løsning, der de hadde all kode i en site.pp og i noen separate bash-skript. Deres løsning var tilpasset en OpenStack versjon som het Havana. I årets bacheloroppgave skal denne oppdateres til neste versjon, IceHouse. Allerede i høst prøvde gruppen å sette opp denne løsningen i et prosjekt i faget Systemadminstrasjon. Gruppen klarte ikke å få alt til å fungere, men det ble kommet et godt stykke på vei. Denne erfaringen var god å ha med seg i det gruppen skulle forsøke på nytt, men gruppen kom til samme punkt som sist, og brukte en god del tid for å få ting til å fungere.

Etter all installering og konfigurering fra Puppet, var det på tide å teste systemet. En innlogging på Horizon ga oss feilmeldingen “500, internal server error”, og etter en del feilsøking viste det seg at gruppen hadde feil versjon av RabbitMQ-modulen. Feilen her var at det ble brukt en for ny versjon av RabbitMQ-modulen (3.x.x istedet for 2.1.0) som ikke var fungerende sammen med OpenStack-modulene. Det var nemlig ikke spesifisert hvilken versjon av RabbitMQ som ble brukt i dokumentasjonen fra forrige bachelorgruppe.

Videre viste kjøring av Puppet-agenten noen feilmeldinger, hvor den første var “class mysql::server not found”. En undersøkelse av nova-modulens konfigurasjon viste at det hadde blitt brukt feil MySQL-modul. Den modulen bacheloroppgaven “AutoStack” hadde skrevet til, var en mye tidligere versjon av modulen, og etter litt research fant gruppen ut at versjon 0.9.0 var den nyeste som passet.

Gruppen fikk en ny feilmelding under kjøring av Puppet-agenten: kunne ikke finne fil: deployController.sh. Litt søking på internett viste at “files”-funksjonen i Puppet krever at filene ligger under modulmappen/-files. I dette tilfelle blir det: /etc/puppet/modules/nova/files.

Etter at gruppen klarte å logge inn på Horizon var tiden inne for å prøve å lage en instanse. Dette endte med en feilmelding. I det instansen så ut til å skulle starte ble det generert en "ERROR 500". Etter hvert ble det klart at nettverksinnstillingene ikke var som de skulle. Fra Havana til IceHouse har det skjedd et bytte av neutron/plugins/ filene, fra ovs_neutron_plugin.ini til ml2.conf.ini. Der ble følgende seksjoner flyttet fra ovs-pluginen til ml2-pluginen: [ovs] og [securitygroup].

Feil ved kjøring av puppet agent -t på nostrand. Å legge til følgende i site.pp fikset dette:

```
        node default{
        }
```

Dobbel deklarering av nettverk plugins

Puppet gav en feilmelding om at begge klassene/plugins (ovs og ml2) ble deklarert i site.pp, siden de begge deklarerte/etc/puppet/modules/neutron/manifests/plugins/plugin.ini. Derfor ble denne delen kommentert bort fra /etc/puppet/modules/neutron/manifests/plugins/ovs.pp, siden denne ikke skal brukes.

```
#  file {'/etc/neutron/plugin.ini':
#    ensure  => link,
#    target  => '/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini',
#    require => Package['neutron-plugin-ovs']
#  }
```

Da skal det gå greit å ha med både class::plugins::ovs og class::plugins::ml2 i site.pp.

nova_admin_tenant_id blir satt hver gang puppet kjører

I /etc/puppet/modules/neutron/manifests/server/notifications.pp er det noe gruppen mener er en bug. Etter at alt er installert av Puppet, er det fortsatt noe som settes hver gang agenten kjører. Følgende melding dukker opp når puppet kjøres:

```
Notice: /Stage[main]/Neutron::Server::Notifications/Nova_admin_tenant_id_setter[
    nova_admin_tenant_id]/ensure: created
```

Det vil si at valget nova_admin_tenant_id blir satt på nytt hver gang puppet kjører, og tjenesten Neutron blir restartet. En måte å unngå dette på er å kommentere bort følgende:

```
#    nova_admin_tenant_id_setter {'nova_admin_tenant_id':
#      ensure            => present,
#      tenant_name       => $nova_admin_tenant_name,
#      auth_url          => $nova_admin_auth_url,
#      auth_username     => $nova_admin_username,
#      auth_password     => $nova_admin_password,
#      auth_tenant_name  => $nova_admin_tenant_name,
#    }
```

Dette hindrer nova_admin_tenant_id i å blir satt hver gang puppet agenten kjører. Dette er rettet på av utviklerne i Juno-versjonen.

Automatisk installering av Juno

Gruppen begynte å oppdatere site.pp i tråd med det som stod i OpenStack manualene, men gruppen oppdaget etter hvert at puppet-modulene fra Puppet Forge[1] virket utdaterte siden gruppen måtte gjøre endringer i modulene selv for at ting skulle være riktig. Dette var ting som utdaterte variable, ting måtte kommenteres bort og nye variable som rett og slett ikke fantes i modulene. Etter samtale med veileder og oppdragsgiver ble gruppen anbefalt å sjekke ut statusen på modulene på github. Etter en kjapp titt her fant gruppen ut at de "offisielle releasene" på Puppet Forge var uten patcher som er blitt tilført siden release i november 2014. Gruppen brukte derfor "git clone" fra disse prosjektetene. Dermed ble alle problemene som eksisterte med de utdaterte modulene, løst. Dette er stien til prosjektene:

https://github.com/stackforge/puppet-nova

https://github.com/stackforge/puppet-neutron

https://github.com/stackforge/puppet-glance

https://github.com/stackforge/puppet-heat

https://github.com/stackforge/puppet-cinder

I tillegg har gruppen oppdaget noe de mener er en feil i neutron-modulen. I følge den offisielle OpenStack dokumentasjonen skal variabelen "rpc_backend" settes til "rabbit", men i modulen må den settes til "neutron.openstack.common.rpc.impl_kombu" ellers vil ikke "rabbit_host" og "rabbit_password" bli satt. Det har gruppen rapportert en feil på til utviklerne: https://bugs.launchpad.net/puppet-neutron/+bug/1431192

Det neste steget ble etter samtale med oppdragsgiver å kvitte seg med skriptene og i stedet implementere disse inn som puppet-kode i site.pp. Puppet har en funksjon kalt command => slik at man kan kjøre kommandoer med puppet-koden, og gruppen begynte arbeidet med å flytte alle kommandoene fra skriptet inn i slike "exec-klasser". Her var det mye som måtte stemme. Det måtte sørges for at exec-ene bare kjørte en gang, slik at ting ikke skulle bli overskrevet hele tiden og at Puppet kjøringer dermed ikke skulle ta en evighet. En annen ting var også at gruppen her måtte passe på hvilken rekkefølge ting ble kjørt i.

Deretter skulle site.pp omgjøres til en puppet modul, slik at man i Foreman kan velge puppet-klasser som man vil installere på nodene som skal rulles ut. Gruppen har gjort det slik at man kan velge om noden skal bli en kontroller, nettverks eller compute-node. For at dette skulle bli riktig måtte gruppen dele opp koden i riktige klasser, og inkludere de riktige klassene for at alt skulle gå knirkefritt.

Endring av pakkestørrelse ved bruk av GRE

Etter å ha installert Juno med GRE-støtte fikk ikke gruppen kontakt med instansene. Etter litt granskning på dette området viste det seg at GRE bruker litt overhead på IP-pakkene og trenger noen ekstra byte for å få pakkene dit de skal (se illustrasjon). Standard pakkestørrelse er maks. 1500 bytes, og switchene mellom serverene var ikke satt opp til å transportere mer enn dette. For å få ned pakkestørrelsen brukte gruppen OpenStack sin DHCP-server som leverer ut IP-ene til instansene for å sette ned den innvendige pakkestørrelsen. Det å sette

```
dhcp-option-force=26,1454
```

i filen /etc/neutron/dnsmasq-neutron.conf ga oss en løsning på dette, for den sier ifra hvor stor payload som skal sendes i hver pakke. Dette rammer virtualiserte Linux-OSer. Rapporter rundt omkring OpenStack sine forumer[26] sier at denne løsningen ikke fungerer på Windows-operativsystemer.

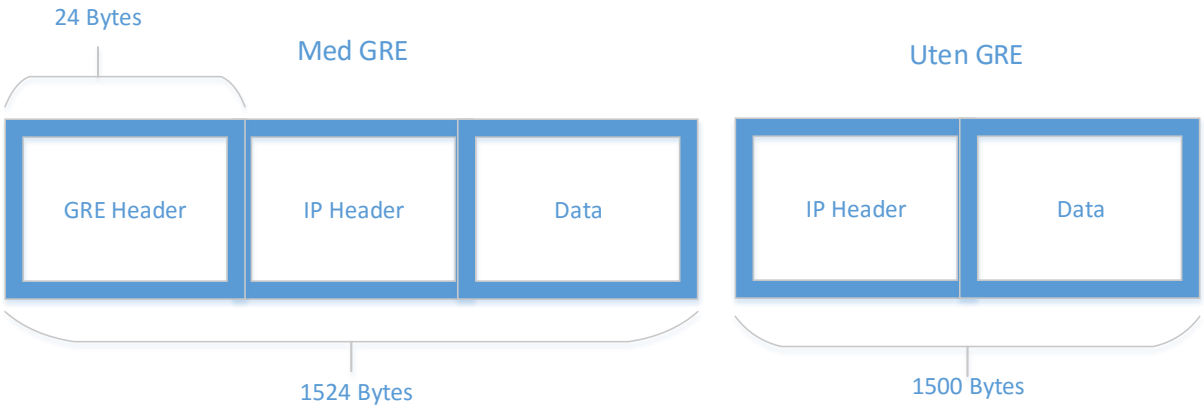


Figure 10: En illustrasjon over hva overheaden vil ha å si på nettverket.

Rally

I eksperimenteringen med å bruke Rally som et test-rammeverk, fikk gruppen ikke Heat til å fungere. Det viste seg at Rally installerer og bruker en annen versjon av nova-rootwrap(Admin-tjeneseten til flere komponenter i OpenStack) enn det Heat gjør. Gruppen prøvde å komme rundt dette med blant annet å installere en nyere versjon av denne modulen, men ingenting fungerte, noe som gjorde Rally ubrukelig.

Samarbeidet med 'Digitalt øvingsfelt' gruppen

I henhold til oppgaven så skulle gruppen:

"Automatisk rulle ut to "cyber range"-scenarier (i samarbeid med "Digitalt øvingsfelt"- gruppen)"

I slutten av april ble det opprettet kontakt med Digitalt øvingsfelt-gruppen for å høre om de hadde noen utkast til templates som gruppen kunne teste ut. Etter litt testing og feiling viste det seg at de hadde brukt noen funksjoner i sine templates som er helt nye og som blir implementert i neste versjon av

OpenStack, versjonen kalt Kilo. Denne versjonen ble offisielt gitt ut 30 april, og gruppen har, som i følge oppgaven, OpenStack Juno installert på infrastrukturen. Å endre på templatene var ikke mulig for "Digitalt øvingsfelt-gruppen", siden de var avhengig av funksjonaliteten til Kilo. Etter møte med oppdragsgiver ble det derfor bestemt at gruppen ikke fullfører denne delen av oppgaven.

Gruppen er dog sikre på at vår infrastruktur støtter utrulling av Heat-templates siden dette er noe gruppen har testet en del. Gruppen har testet å rulle ut Heat-templates både på IceHouse og Juno, og verifisert at det fungerer. Dessuten er utrulling av en Heat-template en del av test-rammeverket der den ruller ut et scenario, verifiserer at det går bra, rapporterer tilbake, for så å slette det igjen. Det er derfor ikke noe som skulle tilsi at dersom man skriver en Heat-template som er tilpasset for IceHouse eller Juno, at dette ikke skal går bra på skyløsningen.

7 Testing og kvalitetssikring

Testing og kvalitetssikring er helt avgjørende i et slikt prosjekt som gruppen har gjennomført. Helt fra starten er dette et viktig punkt som man gjennomgår i hvert steg av prosessen. Gruppen har vært veldig klare på at ting må fungere likt hver gang man utfører dem, og at man alltid oppnår ønsket resultat. Metodene som har blitt brukt ved testing er stort sett de samme over alt, og det blir gjort små endringer for å se om de ble utført riktig eller feiler. På denne måten blir det enklere å feilsøke dersom noe skulle gå galt, kontra å implementere en rekke endringer. Dette kapittelet vil ta for seg hva som er blitt testet og hvorfor.

7.1 Nostrand

Utrulling av operativsystem med Foreman er noe gruppen har fått testet veldig mye siden gruppen har installert systemet på nytt veldig mange ganger for å teste at puppet-koden fungerer som den skal, og at OpenStack er oppe å kjøre i denne automatiske prosessen.

En annen ting gruppen også har gjort en del av er å ta et image av Nostrand før en ny implementering er på plass. Dette er noe som har vært en sikring og en tidsbesparelse siden gruppen i ny og ne har gjort konfigureringer på Nostrand som har gjort serveren ustabil. Gruppen har da kunnet legge inn imaget på nytt og få Nostrand tilbake til stabil tilstand på 10 minutter, i stedet for å bruke en hel dag på å sette opp Nostrand på nytt. Til dette har gruppen brukt Clonezilla [27].

I vårt oppsett har gruppen brukt Foreman versjon 1.7 som har vært den nyeste versjonen tilgjengelig mens gruppen satte opp systemet. I slutten av april kom versjon 1.8[28] av Foreman som gruppen prøvde en rask test av, men her ble det problemer når det skulle slettes verter. Det kan virke som om dette er en feil i den nye versjonen av Foreman. Med det lille som var igjen av tid, ble det bestemt at Foreman 1.7 skulle brukes på systemet siden dette er den mest stabile versjonen.

7.2 Test-rammeverk

Test-rammeverket for skyløsningen skal automatisk sjekke om skyen er klar til bruk. Her var flere ting viktige, hva som skulle sjekkes, rekkefølgen ting ble sjekket, hvordan og hva som skulle rapporteres når både ting gikk som normalt, og når ting feilet.

For å sørge for at skyen er klar til bruk, tar test-rammeverket og ruller ut et lite scenario der det blant annet opprettes nettverk, ssh-nøkler, og instanser for å til slutt sjekke om instansen har tilgang til internett. For at dette skal gjennomføres på en vellykket måte, må testen skje i en viss rekkefølge, for eksempel at det må finnes et nettverk før en instanse kan opprettes, og at det da derfor må opprettes ett nettverk før en kan lage en instanse. Til slutt skal alt slettes, og det må da gjerne skje i motsatt rekkefølge, siden man da for eksempel, ikke kan slette et nettverk som en instanse benytter.

En annen viktig ting med et test-rammeverk er hvordan den rapporterer tilbake resultatene. Gruppen kunne skrevet alt til en tekstfil og bedt brukeren om å åpne denne etter at testen var kjørt og se på resultatet, men gruppen mente det fantes bedre metoder. Gruppen fikk etterhvert test-rammeverket inn i Foreman gjennom en plugin-funksjon slik at testen kan startes og observeres fra Foreman. En dynamisk html side viser resultatet fra testen ettervert som den utføres, slik at brukeren med en gang ser om noe går galt. Måten skriptet returnerer om ting går bra eller dårlig på er at hver kommando blir kjørt i en if-statement, hvis det går bra skrives det ut en melding om at kommandoen fungerte, hvis kommandoen feilet, skriver den ut kommandoens egen feilmelding pluss den egenproduserte. Skriptet skal i tillegg stoppe opp når det skjer en feil, slik at brukeren kan prøve å fikse feilen for deretter prøve å kjøre skriptet på nytt. Skriptet vil da hoppe over det den allerede har laget å begynne der det stoppet opp sist. For å få alt dette til å bli stabilt måtte gruppen kjøre testen x-antall ganger før de ble fornøyde. Det dukket opp ting underveis som for eksempel at gruppen må vente på at instansen skal boote opp før gruppen prøver å koble til med ssh.

Gruppen har også kontrollert at test-rammeverket fungerer med IceHouse installasjonen. Her måtte gruppen endre heat_template_version: 2014-10-16 til heat_template_version: 2013-05-23 i template.yml, for at IceHouse skulle rulle ut stacken.

7.3 Maskinvare

Gruppen har stort sett hatt en stabil maskinvare gjennom prosjektet, men gruppen oppdaget etter hvert at to av servene har defekte BIOS-batterier og blir satt til at virtualisering er slått av som default når de mister strømmen. Dette er noe gruppen oppdaget etterhvert når de installerte OpenStack komponenter på dem og ting ikke fungerte som de skulle, siden virtualisering ikke var aktivert på CPU-en. Etter at dette ble oppdaget, har det ikke vært noe problem siden gruppen bare har måttet sørge for at serverne hele tiden er tilkoblet med strøm.

Gruppen har også prøvd ut skyløsningen på forskjellige hardware plattformer for å teste ut allsidigheten til OpenStack og systemet vårt. De forskjellige plattformene gruppen har testet er følgende: HP ProLiant DL 380 G5, HP ProLiant DL 320 G5, Dell Optiplex 745 og en stasjonær PC med en moderne AMD CPU. Alle disse har PXE-boot og CPU-er som støtter virtualisering, noe som er et krav. Disse har alle fungert uten problemer. Gruppen testet også med en Dell Optiplex 740 maskin, men denne klarte ikke å boote fra PXE i det hele tatt, selv om den støttet dette. Gruppen brukte ikke noe mer tid på å finne ut hvorfor dette ikke fungerte på denne modellen, siden det var utenfor oppgaven. Det kreves at man gjør endringer i /var/lib/tftpboot/pxelinux.cfg/<MAC-addresse> på Nostrand slik at den booter opp igjen automatisk. Dette er noe som det kan trenges litt justeringer til for å få til å fungere på en viss plattform.

7.4 Virtuelle operativsystem

Det har ikke vært noen krav om spesifikke operativsystemer som skal kjøre på løsningen, så derfor har gruppen bare tatt i bruk de som AutoStack hadde før gruppen - nemlig CirrOS og det nyeste Ubuntu server-imaget. Disse har installert og fungert uten problemer, og er klar til å brukes. Et operativsystem som hadde vært interessant å teste, ville vært et Windows-2012-server-image. Dette må man dessverre registrere seg for å få tilgang til en prøveversjon, og gruppen har ikke satt av tid til dette i oppgaven.

Dersom man skal legge til et Windows Server-image senere, kan det være verdt å merke seg at flere andre

OpenStack-brukere har meldt problemer med nettverkstypen GRE og pakkestørrelse[26]. Mer om dette i kapittelet utfordringer.

Andre UNIX-baserte løsninger burde fungere fint, da disse bygger på mye av det samme hva angår nettverksløsninger.

8 Drøftinger/diskusjoner

Systemutviklingsmodell

I oppgaven har gruppen valgt å gå for en agil systemutviklingsmetode. Metoden består av en blanding mellom inkrementell og Scrum. Dette gjør at gruppen har hatt sprint sykluser og testing etter hver iterasjon. Det gjør også at gruppen ikke skrev hele kravspesifikasjonen i begynnelsen, og har derfor kunnet endre på den dynamisk etter hvert i utviklingen. Etter hvert som prosjektet ble ferdig, så gruppen at det ferdige produktet ble litt annerledes enn gruppen hadde sett for seg på begynnelsen av prosjektet. Det var for det meste flere funksjoner lagt til enn fjernet, og den ekstra tiden det tok oss å legge til disse endringene ville blitt brukt på alternativer uansett. Her er noen eksempler:

Skript

Alle utrullingsskriptene skulle implementeres inn i site.pp. Dette ble avklart på møte den 16. februar, og innebærte mange exec-klasser i stedet for ett stort deployskript. Når dette er sagt, ga det en mye mer konsistent og robust metode å rulle ut på, siden man nå kan sjekke hver eneste kommando for feil og legge inn individuelle krav til når disse kommandoene skulle kjøre.

Moduler

Etter demo nummer to fikk gruppen svar fra oppdragsgiver at alt ikke var som han hadde forventet. Han hadde regnet med at gruppen hadde all koden inne i en separat modul, og ikke i én site.pp. Etter en diskusjon med arbeidsgiver om fordelene det ville gi, tok gruppen på seg arbeidet om å flytte all koden. Noen av fordelene vil være at Foreman fungerer som en External Node Classifier (ENC), noe som gir mer fleksibilitet angående navngiving på noder og konfigurasjonsoppsett, og man får mindre duplisering av kode, siden klasser kan inkluderes mer dynamisk. Det tok ca. tre dager å gjøre dette, siden koden bare skulle splittes opp og legges inn i forskjellige filer.

Scenario

Etter avtale med oppdragsgiver og Digitalt øvingsfelt, skulle gruppen rulle ut to av deres scenarier for å blant annet teste kompabilitet og funksjonalitet sammen med løsningen. Etter å ha mottatt ett av disse, oppstod det problemer. Det viste seg at Digitalt øvingsfelt hadde skrevet kode for en nyere versjon av OpenStack enn den gruppen brukte; nemlig OpenStack Kilo. Etter en diskusjon med arbeidsgiver (som også er veileder for Digitalt øvingsfelt-gruppen) den 27. April, fant gruppen ut at de skulle droppe denne oppgaven. Dette ga gruppen mer tid til å fullføre test-rammeverket, og gruppen lå mer i samsvar med tidsplanen.

9 Avslutning

9.1 Resultat

Hovedsaklig har gruppen endt opp med å besvare de slik gruppen så for seg i starten av prosjektet, men hvordan sluttproduktet skulle se ut har endret seg noe etterhvert i prosjektets gang. Det at skriptene fra AutoStack oppgaven i fjor skulle implementeres inn i site.pp, som igjen skulle lages til en egen puppet modul, var ikke gruppen klar over når prosjektet begynte. Dette er noe som oppdragsgiver uttrykte et ønske om underveis i prosjektet og som gruppen tilpasset seg etter. Gruppen har også feiltolket oppgaven der gruppen trodde at de skulle skrive test-rammeverk for scenariene også.

Gruppen er meget fornøyd med det produktet som har blitt realisert. Gruppen har nå et system der man, fra Foreman, installerer noder over PXE, installerer OpenStack Juno på nodene ved hjelp av Puppet, og tester om nettskyen er operativ via test-rammeverket som gruppen har implementert i Foreman - alt automatisk.

Ved å flytte hele site.pp til en egen "AutoStack modul" kan man nå fra Foreman velge om en node skal settes opp som en Compute node, Controller node, Network node eller både Controller og Network i en og samme node.

Et test-rammeverk er på plass i Foreman som skal bekrefte om skyløsningen er klar til bruk eller ikke. Gruppen er veldig fornøyde med måten gruppen har fått implementert dette inn i Foreman på. Brukervennligheten blir betraktlig forbedret ved at man starter test-rammeverket ved å trykke på en knapp og får resultatet skrevet ut fortløpende på samme siden, enn ved å logge seg inn på Nostrand og starte skriptet derfra med medfølgende rapport i terminalen.

Gruppen har også gjort det mulig via Foreman at brukeren kan velge mellom VLAN eller GRE som nettverksstruktur i sin nettsky. Dette gjøres med "overriding" av variabelen "networkType" og ettersom brukeren skriver 'VLAN' eller 'GRE' her, vil gruppens modul konfigurere nodene med riktige nettverksinnstillinger.

I tillegg har Heat-komponenten blitt lagt til i skyløsningen slik at scenarier kan rulles ut med Heat-templates. Dette er en tidsbesparende komponent som kommer til sin rett dersom noen ofte ønsker å sette opp test-scenarier på nytt.

Med små tilpasninger føler gruppen seg trygg på at AutoStack skal kunne fungere i en rekke forskjellige miljøer og oppsett.

9.2 Videre arbeid

Automatisk oppsett av Nostrand

Oppsettet av Nostrand og Foreman kan automatiseres. Det finnes en egen Puppet modul for installering

og oppsett av Foreman[29]. I tillegg kunne all koden som er blitt generert i løpet av prosjektet blitt pakket i en .deb pakke slik at alle filene blir lagt der de skal være.

Oppdatering til neste OpenStack versjon

Oppgradering til Kilo vil gå på samme måte som gruppen har gjort tidligere. Ut i fra OpenStack sine hjemmesider[3] ser det ikke ut som det vil være så altfor mange forskjeller i konfig-filer, men heller mange nye funksjoner på plass. Denne informasjonen er dessverre mangelfull, fordi Kilo er ikke utgitt i skrivende stund (28.4), og versjonsmerknadene er ikke ferdige. Men dersom oppgraderingen går litt som tidligere, kan man merke seg følgende:

- **Puppet moduler må være oppdatert**
  Gjelder alle modulene som omhandler OpenStack. Dette er gruppen fullstendig avhengig av. Det kan være smart å hente modulene direkte fra Git for å få de nyeste funksjonene og oppdateringene.
- **Riktige programvarepakker må installeres**
  Til Juno så har gruppen måttet installere egne ppa-arkiv for å få riktig programvare, dette må oppdateres til Kilo sine arkiv. Dette ligger i Puppet-filen "common.pp" i gruppens modul.
- **Innstillinger**
  Innstillinger og konfigurasjon som blir utdatert med den nye versjonen av OpenStack må fjernes. Der må det først sjekkes om Puppet-modulene er kompatible med dette, det har gruppen hatt problemer med tidligere (referanse?). Nye innstillinger som er påkrevd må være med, mens andre innstillinger kan vurderes om de er nødvendige. Det anbefales å følge en offisiell oppgraderingsguide for å se hvilke innstillinger som må endres/legges til.
- **Alt burde skje i Puppet**
  Nå er hele systemet rundt Foreman og utrulling er ferdig. Det eneste utenfor OpenStack som kunne trengt en oppgradering ville i så fall bli Linux-OSet som blir rullet ut på serverene. Dette endres i Foreman sitt interface, og burde bare være å endre versjonsnummer (eks. fra 14.04 til 15.04). Resten av endringene vil skje i Puppet-modulen.

Alt i alt burde det ikke være så altfor mye jobb, men med erfaring fra tidligere oppgraderinger kommer det overraskelser. I tillegg burde man være kjent med Linux og OpenStack for å klare oppgaven, siden den går så dypt inn i unix-verdenen.

Opprettelse av VM-er i OpenStack gjennom Foreman

Underveis i prosjektet er det registrert at det finnes løsninger der instanser kan opprettes fra webgrensesnittet i Foreman. Det er da rimelig å anta at man også kan opprette nettverk og lignende i Foreman. Dette er noe som kan være interessant å se nærmere på. Denne linken viser en presentasjon av en som jobber ved CERN der de har dette systemet: Sjekk side 31 i denne sliden[30].

Oppsett av Windows instanser

Gjeldene skyløsning bruker bare Linux debian distroer som instanser i skyen. Det kunne vært interessant å teste løsningen med Windows instanser, da det antakelig er behov for å kjøre programvare som bare er laget for Windows plattformen.

StayPuft

StayPuft er en plugin i Foreman som skal sørge for en automatisk utrulling av OpenStack, med andre ord, et lignende system som det representert i denne oppgaven. StayPuft kjører bare på RedHat distrubisjoner, så dersom dette skal være et alternativ må miljøet kjøre RedHat.

9.3 Evaluering av gruppens arbeid

Innledning

Siden ingen i gruppen har drevet med et like stort prosjekt før, måtte det være et system på fordeling og evaluering av arbeid. Arbeidet har foregått på samme rom hvor alle sitter samlet, og dermed har ikke kommunikasjon vært et problem. Det har oppstått særdeles få uenigheter i gruppen, og arbeidsmoralen og samarbeidet har vært godt.

Organisering

Selve organiseringen er beskrevet i vedlegg B (Forprosjekt). Dette har fungert meget godt, og ingen medlemmer av gruppen har hatt noen innvendinger ved dette. Gruppen har også hatt god kontakt med både oppdragsgiver og veileder. Siden de deler kontor så har det vært lett å få innspill fra en av dem dersom det er nødvendig.

Gruppearbeid

Gruppen har sittet på samme rom og jobbet med oppgaven helt fra begynnelsen. Dette har ført til veldig god kommunikasjon mellom medlemmene, og ellers ett godt samarbeid. Problemer som oppstod, men som ingen hadde tid til å gå igjennom, ble lagt på et internt memo kalt ‘arbeidsoppgaver’ for senere oppgavefordeling. Problemløsning har for det meste blitt gjort gruppevis, da visse problemer trenger en hurtig løsning.

Fordeling av arbeidet

I gruppen er det fordelt arbeid fortløpende. Siden problemer skal løses hele tiden, tar den som får ledig tid en kikk på ledige arbeidsoppgaver og begynner på den. Dersom det ikke er noen som tar på seg ansvaret, kommer gruppeleder til å dele ut ansvar. Dette har skjedd i noen tilfeller.

Prosjektet som arbeidsform

Prosjektet som arbeidsform har gruppen god erfaring fra tidligere oppgaver på HiG. Da det er sagt, er det å gjennomføre et så stort prosjekt som dette noe ingen hadde gjort før. Dette prosjektet har gitt gruppen stor innsikt i hvordan det er å jobbe med store prosjekter.

9.4 Konklusjon

Hvis man sammenligner selve oppgaveteksten med det som faktisk har gjort, har nesten samtlige av punktene i oppgaveteksten fullført. Dette innebærer at AutoStack fungerer på 380G5 servere, gruppen har oppdatert AutoStack til OpenStack IceHouse, de har oppdatert AutoStack til OpenStack Juno. Gruppen har også redesignet AutoStack i form av å konvertere koden i skriptene fra AutoStack til puppet kode. Det har blitt skrevet et helt nytt test-rammeverk for å teste om OpenStack-løsningen fungerer, og det er i tillegg blitt integrert i Foreman sitt grensesnitt. På grunn av årsaker beskrevet tidligere [link] kunne dessverre ikke gruppen innfri oppgavetekstens siste punkt om å automatisk rulle ut to “cyber-range” scenarier som Digitalt øvingsfelt hadde laget, men gruppen kan med sikkerhet si at løsningen kan rulle ut scenarier med riktig OpenStack versjon automatisk. I tillegg har gruppen lagt til støtte for å inkludere storage-noder i sin OpenStack-løsning hvis eventuelt noen skulle ha bruk for dette.

Etter å ha jobbet med denne oppgaven har gruppen lært mye om Foremans funksjonalitet, Puppet-kode og automatisk Puppet konfigurasjon, og de har lært mye om hvordan OpenStack som en skyløsning må installeres og settes opp. Med andre ord har denne oppgaven vært veldig givende i form av læringsutbytte.

Bibliography

[1] Puppetlabs. 2015. puppetlabs sin utviklingsside. https://forge.puppetlabs.com/puppetlabs.

[2] StackForge. 2015. Stackforge sin utviklingsside. https://forge.puppetlabs.com/stackforge.

[3] OpenStack. 2015. Openstack sin hjemmeside. http://www.openstack.org/software/.

[4] OpenStack. 2015. Nova-compute. http://www.openstack.org/software/openstack-compute/.

[5] OpenStack. 2015. Vmware støtte i openstack. http://docs.openstack.org/kilo/config-reference/content/vmware.html.

[6] OpenStack. 2015. Hyper-v -støtte i openstack. http://docs.openstack.org/kilo/config-reference/content/hyper-v-virtualization-platform.html.

[7] OpenStack. 2015. Nova. http://docs.openstack.org/developer/nova/.

[8] OpenStack. 2015. Nova. http://www.openstack.org/software/openstack-networking/.

[9] Keystone. 2015. Keystone på git. https://github.com/openstack/keystone.

[10] OpenStack. 2015. Keystone på openstack. https://wiki.openstack.org/wiki/Keystone#What_is_Keystone.3F.

[11] Wikipedia. 2015. Keystone på wikipedia. http://en.wikipedia.org/wiki/OpenStack#Identity_Service_.28Keystone.29.

[12] Wikipedia. 2015. Glance på wikipedia. http://en.wikipedia.org/wiki/OpenStack#Image_Service_.28Glance.29.

[13] Wikipedia. 2015. Heat på wikipedia. http://en.wikipedia.org/wiki/OpenStack#Orchestration_.28Heat.29.

[14] Wikipedia. 2015. Cinder på wikipedia. http://en.wikipedia.org/wiki/OpenStack#Block_Storage_.28Cinder.29.

[15] Teamet, R. 2015. Rabbitmq sin hjemmeside. https://www.rabbitmq.com/features.html.

[16] Wikipedia. 2015. Mysql på wikipedia. http://no.wikipedia.org/wiki/MySQL.

[17] Wikipedia. 2015. Horizon på wikipedia. http://en.wikipedia.org/wiki/OpenStack#Dashboard_.28Horizon.29.

[18] OpenStack. 2015. Rally på openstack sin wiki. https://wiki.openstack.org/wiki/Rally.

[19] OpenStack. 2015. Openstack sin offisielle guide for oppgradering fra havana til icehouse. http://docs.openstack.org/openstack-ops/content/upgrades_havana-icehouse-ubuntu.html.

[20] OpenStack. 2015. Ml2-konfigurasjonen til icehouse. http://docs.openstack.org/icehouse/install-guide/install/apt/content/neutron-ml2-controller-node.html.

[21] OpenStack. 2015. Ovs-installering. http://docs.openstack.org/admin-guide-cloud/content/install_neutron_agent_ovs.html.

[22] Mirantis. 2015. Rally. https://www.mirantis.com/blog/rally-openstack-tempest-testing-made-simpler/.

[23] readthedocs. 2015. Rally. http://rally.readthedocs.org/en/latest/index.html.

[24] Rally. 2015. Rally testskript. https://github.com/stackforge/rally/blob/master/samples/deployments/existing.json.

[25] Foreman. 2015. Hvordan skrive en foreman-plugin. http://projects.theforeman.org/projects/foreman/wiki/How_to_Create_a_Plugin.

[26] Ewald, M. 2015. Problemer med windows når gre brukes. https://ask.openstack.org/en/question/26165/neutron-gre-with-mtu-1454-cause-windows-network-speed-too-low/.

[27] CloneZilla. 2015. Clonezilla homepage. http://clonezilla.org/.

[28] TheForeman. 2015. Foreman1.8. http://projects.theforeman.org/rb/release/28.

[29] TheForeman. 2015. Foreman sin puppet-modul. https://forge.puppetlabs.com/theforeman/foreman.

[30] CERN. 2015. Foreman hos cern. http://www.slideshare.net/PuppetLabs/puppetconfnonotes?qid=610a90a8-cd58-4201-bb52-8d73510f62e7&v=qf1&b=&from_search=20.

A Terminologi

Compute-nodene: Skal tilby sin maskinvare i form av CPU, RAM og HDD for å tjene de virtuelle maskinene i skyen.

DHCP: Står for Dynamic Host configuration Protocol. Er ansvarlig for å dele ut lokale IP-adresser automatisk, sammen med andre IP-adresser som kan være praktisk for en maskin å vite om, f. eks. PXE-boot adressen.

DNS: Står for Dynamic Name service. En tjeneste som bytter ut navn med IP-adresser og vice versa.

GRE: Generic Routing Encapsulation (GRE) er en nettverksteknologi som pakker inn IP-pakker for å lage helt nye pakker med forskjellig ruting informasjon.

Instanse: En virtuell maskin i skyen. Også kalt en "VM".

Interface: Den fysiske nettverksporten på maskinen. innehar gjerne navn som: eth0, eth1 osv.

Iptables: Den innebygde brannmuren i Ubuntu Server.

IP-adresser: Den logiske adressen til et interface.

Kontroller-noden: Er ansvarlig for å administrere skyen.

LAN: Står for Local Area Network. På norsk blir dette lokalt nettverk.

LDAP: Forkortelse for Lightweight Directory Access Protocol brukes til oppslag i en katalogtjeneste på en server.

Nettverks-noden: Skal fungere som en ruter for de virtuelle maskinene der den sørger for kontakt med omverdenen i form av NAT, og virtuelle lokalnettverk med DHCP.

Nostrand: Gruppens hovedserver som har ansvaret for å rulle ut OpenStack konfigurasjon til de andre serverne. Det er også denne maskinen Foreman er innstallert på.

OpenStack: OpenStack er en sky-løsning som kontrollerer store forekomster av databehandlings-, lagrings-, og nettverksressurser.

OS: Forkortelse for operativsystemer.

OVS: Open vSwitch er en open-source plugin i OpenStack, og den fungerer som en virtuell switch.

Puppet: Puppet er et konfigurasjonsstyringssystem som lar gruppen definere en tilstand på sin infrastruktur, for deretter å sørge for at dette automatisk blir konfigurert og vedlikeholdt utover alle servere og maskiner.

PXE boot: I stedet for at bruke harddisk å starte fra, brukes for eksempel en annen server (Nostrand) som man henter data fra over nettverket.

Storage-node: Skal tilby volumer som instanser kan bruke som ekstra lagringsplass.

VLAN: Virtuelle LAN som legger til en VLAN “tag” i ethernet headeren, slik at man kan dele en fysisk switch inn i flere logiske.

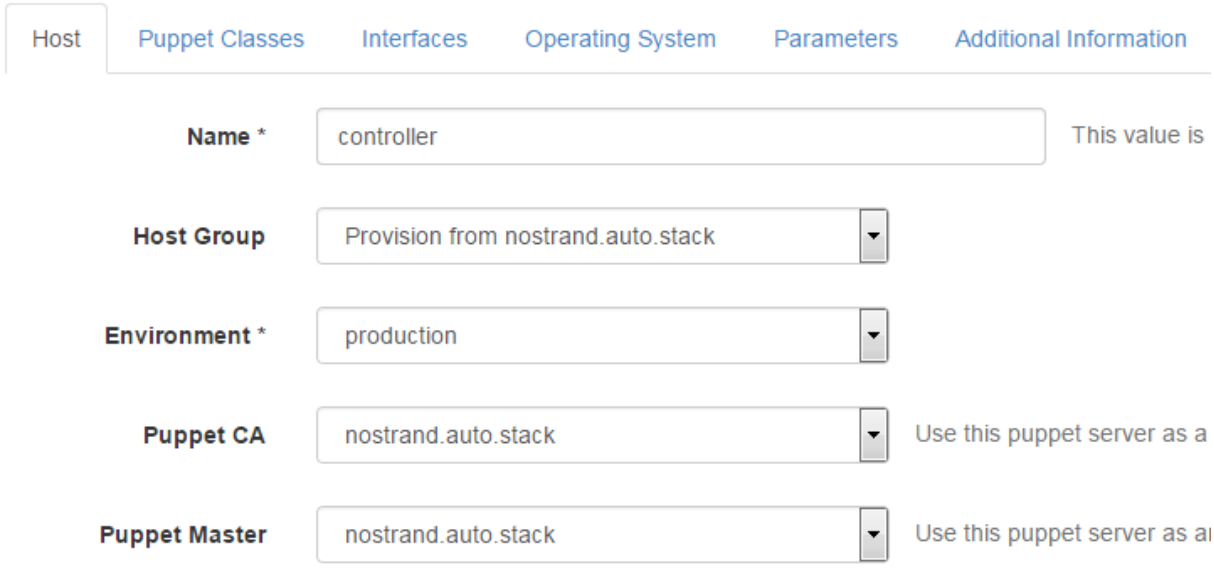
B Brukerveiledning - Foreman

Brukerveiledning - Foreman

Lage en ny Host i Foreman

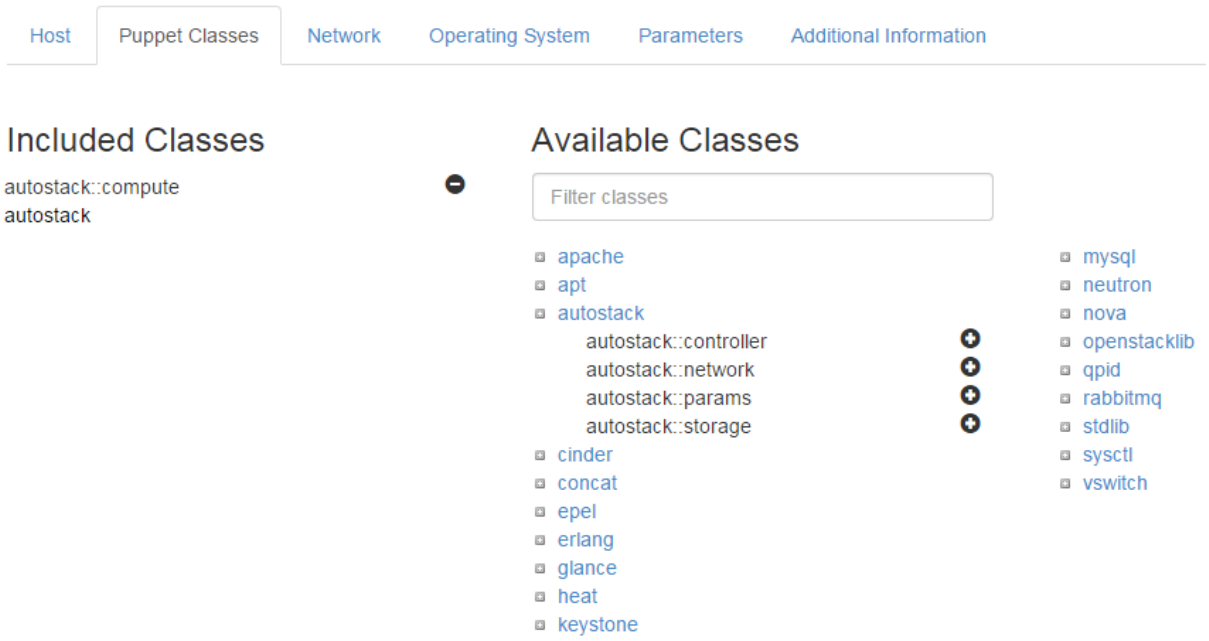
1) Lag en ny host - Host

Hosts -> New host



Name: Navnet til den nye host-maskinen
Host Group: Kan velge en “Provision” slik at alle standard valg blir valgt automatisk
Deploy on: Bare ett valg her, “Bare Metal”
Environment: Valg av hvilket arbeidsmiljø host-en skal inn i
Puppet CA: Valg av Puppet sertifiserings-server
Puppet Master: Valg av Puppet Master som host-en skal rapportere til

3) Lag en ny host - Puppet Classes

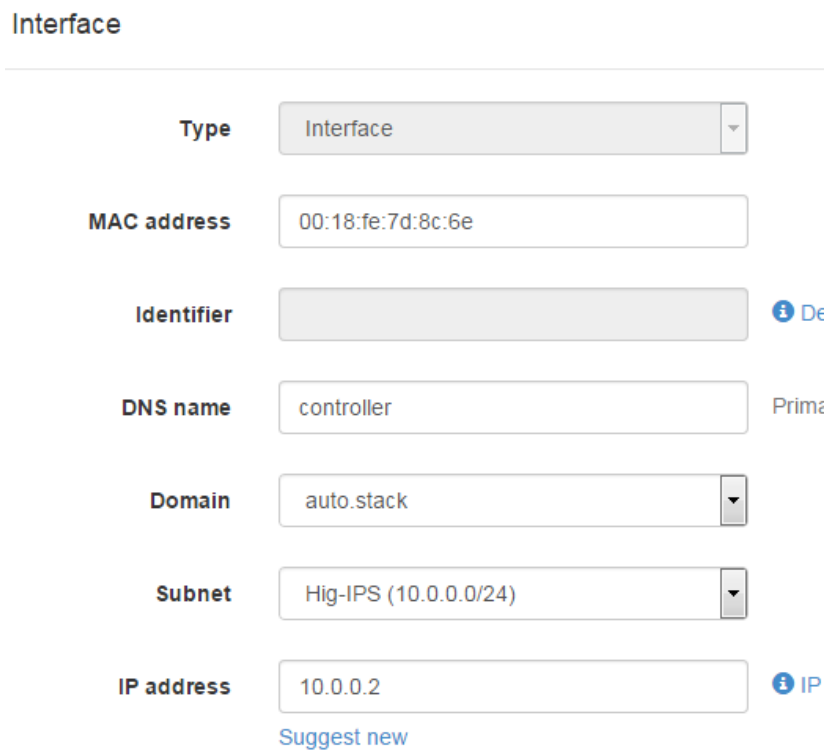


Her kan man legge til Puppet klasser som skal legges til på host-ene slik at disse klassene kjører etter at hosten har startet opp etter installeringen.

For å velge hvilken rolle noden skal ha i OpenStack miljøet velger man en av disse autostack::controller.pp, autostack::network.pp, autostack::storage.pp eller autostack::compute.pp. Man kan også installere controller og nettverk på samme maskin hvis man legger til begge disse klassene.

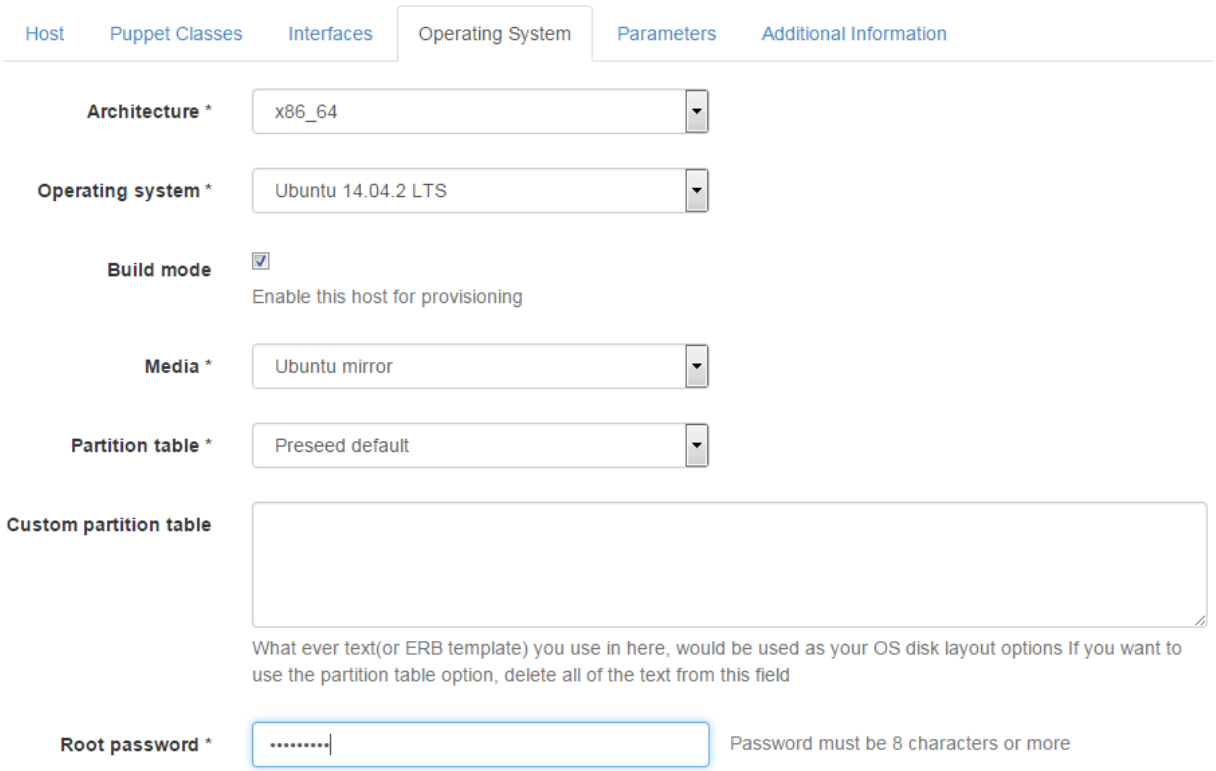
4) Lag en ny host - Interfaces

Velg “Edit” for å få opp menyen.



Type: Valg av type interface
MAC address: MAC adressen til interfacet hoste-en skal boote fra
Identifier: Navnet på interfacet (eksempel eth0)
DNS name: Navnet til host-en
Domain: Domenet vi har satt opp “auto.stack”
Subnet: Subnettet vi har satt opp
IP address: IP addressen fra vår DHCP server

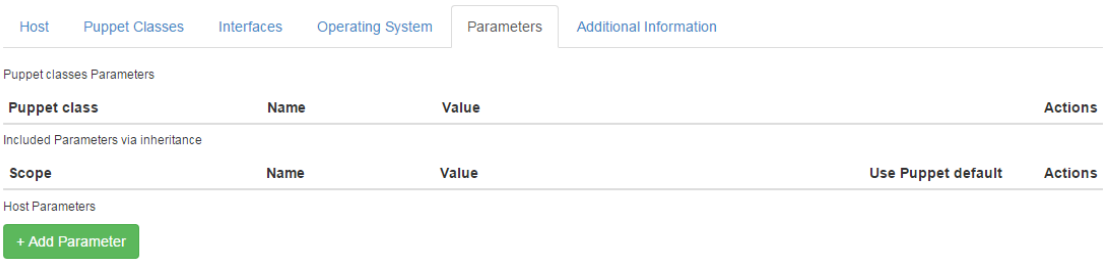
5) Lag en ny host - Operating System



Forklaring:

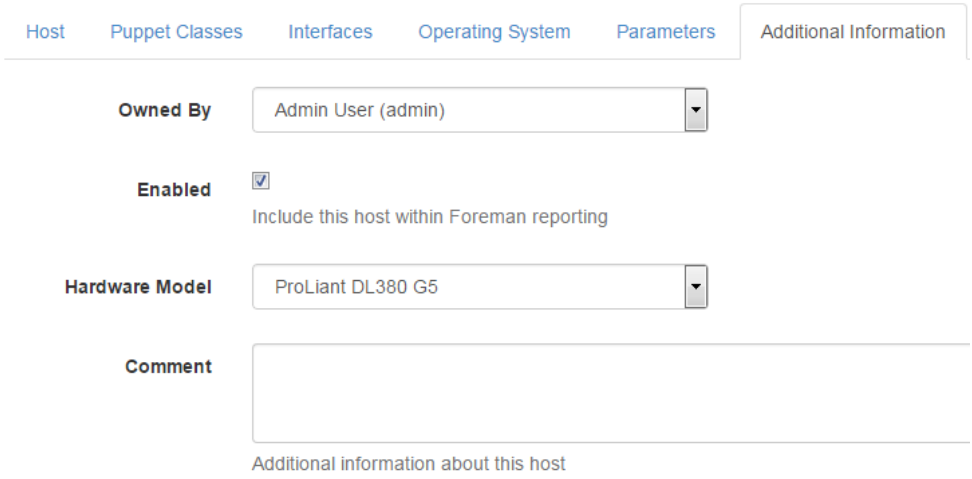
Architecture: Valget mellom 32bit eller 64bits operativsystem
Operating system: Valg av ønsket operativsystem
Build mode: Valg om host er klar til å boote eller ei
Media: Hvor installasjonspakkene skal hentes fra
Partition table: Hvilken partisjonstabell som skal tas i bruk
Custom partition table: Her kan en alternativ partition tabell settes inn.
Root password: Setter passordet på host-en

6) Lag en ny host - Parameters



Dersom man har brukt “Overriding” i Foreman vil disse variablene vises her.

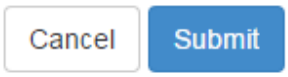
7) Lag en ny host - Additional Information



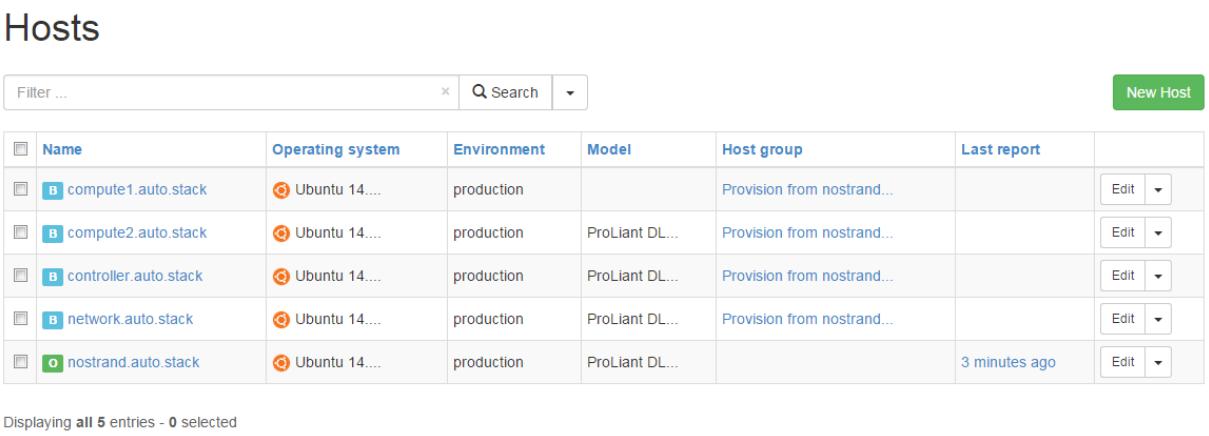
Tilleggsinformasjon slik som Hardware modell, kommentarer og eierskap.

8) Lag en ny host - Submit

Etter at alt er fylt inn er det bare å trykke på “Submit” og host-en er klar.



9) Oversikt



Her ser vi hvordan Foreman lister opp alle hoste-ene. Maskinene merket “B” er de nye hostene vi har satt opp og de er klare for boot over PXE. Nostrand selv er nederst og er merket med en grønn “O” som sier at hosten er oppe å kjøre uten feil.

C Installasjonsguide

Installasjonsguide for AutoStack

Oppsett av DHCP på Nostrand

Installasjon av DHCP:

```
apt-get update
apt-get install isc-dhcp-server
```

Nettverksinstillingene skal settes opp slik at det nettverkskortet som vender mot eksterne nettverk mottar en IP-adresse automatisk, og det nettverkskortet som vender mot skyløsningen skal ha en statisk konfigurert adresse. Dette gjøres ved å endre `/etc/network/interfaces` som vist under:

```
# The primary network interface
auto eth1
iface eth1 inet dhcp

auto eth2
iface eth2 inet static
address 10.0.0.1
netmask 255.255.255.0
network 10.0.0.0
broadcast 10.0.0.255
```

For å starte det interfacet som skal tjene det interne nettverket vårt, kjøres denne kommandoen:

```
ifup eth2
```

Det må gjøres endringer i `/etc/dhcp/dhcpd.conf` og `/etc/default/isc-dhcp-server` for at DHCP serveren skal bli satt opp riktig. Her settes det hvilket nettverk og subnet vi skal bruke, i tillegg til ekstra informasjon slik som bootp-protokollen osv.

`/etc/dhcp/dhcpd.conf`:

```
ddns-update-style interim;
ignore client-updates;
authoritative;
allow booting;
allow bootp;
```

```
}
```

Dette oppsettet har vi hentet fra Foreman sin dokumentasjon (http://theforeman.org/manuals/1.7/index.html#4.3.5DHCP), og tilpasset til vårt oppsett.

Nederst i filen `/etc/default/isc-dhcp-server` setter vi hvilket interface DHCP-serveren skal håndtere DHCP trafikk på. Dette er viktig å sette til det interne nettverksinterfacet, for hvis dette settes til feil nett, så vil alle enheter som trenger tilgang til nettet få svar fra feil DHCP-server. I vårt tilfelle er det interne nettet eth2. Dette kan sjekkes med kommandoen `ip a`.

```
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="eth2"
```

Oppsett av NAT på Nostrand

Det må også aktiveres NAT slik at serverne internt kan få tilgang til omverdenen. Ubuntu server kommer ikke med dette skrudd på som standard, så dette må gjøres manuelt. Husk at eth1 er det eksterne nettet, mens eth2 er det interne. Legg til følgende regler i iptables ved å kjøre disse kommandoene:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth1 -o eth2 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -j ACCEPT
```

IPv4 videresending må også aktiveres for å bruke Nostrand som en ruter. I filen `/etc/sysctl.conf`, sett `net.ipv4.ip_forward` til `1`:

```
net.ipv4.ip_forward=1
```

For at dette skal aktiveres må kommandoen `sysctl -p /etc/sysctl.conf` kjøres.

Rutingtabellen blir tilbakestilt hver gang Nostrand restarter, vi har derfor laget et script som legger tilbake disse innstillingene hver gang maskinen starter opp på nytt.
Først kjører vi kommandoen: `iptables-save > /etc/iptables.rules` som lagrer innstillingene. Deretter lager vi et skript “startopp.sh” som vi legger i `/etc/init.d/` og som inneholder én linje:

```
iptables-restore < /etc/iptables.rules
```

Dette skriptet legger nå til de lagrede innstillingene hver gang det kjøres. Husk også å kjøre kommandoen `chmod +x /etc/init.d/startopp.sh`. for å gjøre skriptet kjørbart. For at skriptet skal kjøre etter hver oppstart, må vi kjøre følgende kommando:

```
update-rc.d -f startopp.sh defaults
```

Dermed gjenstår det bare å starte DHCP tjenesten med denne kommandoen:
`service isc-dhcp-server start`
DHCP-serveren og generell ruting skal da være oppe å kjøre. Det kan være lurt å teste med en bærbar PC eller noe lignende bare for å se om man får internett.

Oppsett av DNS på Nostrand

Installerer bind9:

```
apt-get install bind9
```

For oppsett av DNS på Ubuntu har vi fulgt denne oversiktlige guiden:
http://askubuntu.com/questions/330148/how-do-i-do-a-complete-bind9-dns-server-configuration-with-a-hostname

Følgende filer må endres på:

- /etc/bind/named.conf.options
- /etc/bind/named.conf.local
- /etc/bind/zones/db.auto.stack
- /etc/bind/zones/db.10
- /etc/resolvconf/resolv.conf.d/head
- /etc/hosts
- /etc/apparmor.d/usr.sbin.named

/etc/bind/named.conf.options:
I denne filen skal utvendige DNS-servere oppgis for å få svar ellers fra internett.

```
options {
	directory "/var/cache/bind";
forwarders {
	10.0.0.1;
	128.39.32.2;
	8.8.8.8;
};
	dnssec-validation auto;
	auth-nxdomain no;    # conform to RFC1035
	listen-on-v6 { any; };
};
```

/etc/bind/named.conf.local:
I denne filen definerer vi forward soner og reverse soner. Altså oversettingen mellom navn og IP-adresse, og omvendt.

```
# Our forward zone
zone "auto.stack" {
 type master;
 file "/etc/bind/zones/db.auto.stack";
 allow-update { localhost;};
 };
```

```
# Our reverse Zone
# Server IP 10.0.0.1
zone "0.0.10.in-addr.arpa" {
  type master;
  file "/etc/bind/zones/db.10";
  allow-update { localhost;};
 };
```

Vi lager mappen zones og filene db.auto.stack og db.10.
Etter dette så må disse kommandoene kjøres for å få riktige rettigheter på de opprettede filene:

```
cd /etc/bind
chown bind zones
chown bind zones/db.auto.stack
chown bind zones/db.10
```

/etc/bind/zones/db.auto.stack:
Forward lookup sone. Her blir alle “navn-spørringer” oversatt til en IP-adresse.

```
$TTL   604800
@      IN     SOA    nostrand.auto.stack. foreman.auto.stack. (
                             3      ; Serial
                             604800         ; Refresh
                             86400          ; Retry
                             2419200        ; Expire
                             604800 )       ; Negative Cache TTL
;
auto.stack.          IN  NS  nostrand.auto.stack.
auto.stack.          IN  A   10.0.0.1
nostrand             IN  A   10.0.0.1
gateway              IN  A   10.0.0.1
www                  IN  CNAME   auto.stack.
```

/etc/bind/zones/db.10:
Reverse lookup sone. Her blir alle “IP-adresse-spørringer” oversatt til et hostname.

```
$TTL   604800
@      IN     SOA    nostrand.auto.stack. foreman.auto.stack. (
                             1      ; Serial
                             604800         ; Refresh
                             86400          ; Retry
```

```
                        2419200       ; Expire
                        604800 )      ; Negative Cache TTL
;
        IN  NS  nostrand.
1       IN  PTR gateway.auto.stack.
1       IN  PTR nostrand.auto.stack.
```

/etc/dhcp/dhclient.conf:

Denne filen inneholder konfigurasjonen til den lokale dhcp-klienten. Denne legger informasjonen sin inn i den lokale dnsmasq-serveren, som Ubuntu bruker bl. a. som cache når den skal se etter navneservere. Disse valgene overstyrer hvor maskinen skal se etter navn, og gjør det dermed mulig å koble direkte til serverene med navn:

```
supersede domain-name "auto.stack";
prepend domain-name-servers 127.0.0.1;
```

Kjør så denne kommandoen for å starte nettverksporten på nytt og laste inn de nye instillingene for DNS-løsing:

```
ifdown eth1 && ifup eth1
```

/etc/hosts:

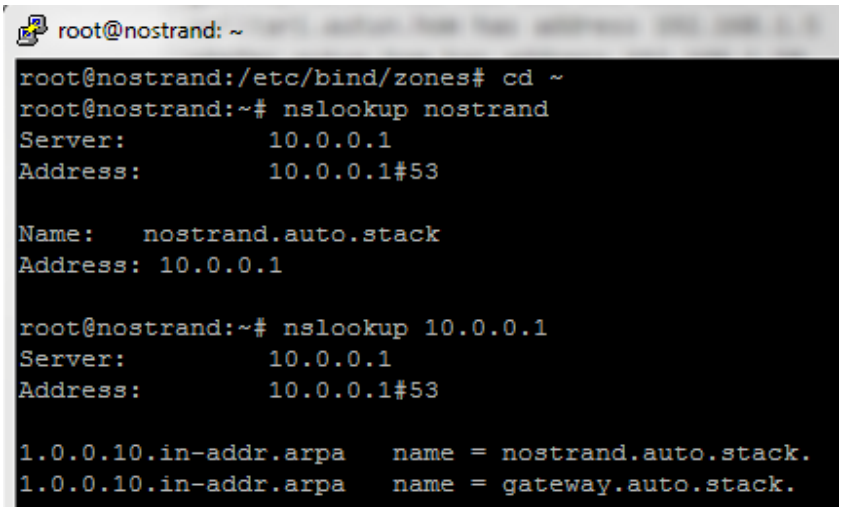
Endre `/etc/hosts` til 10.0.0.1 nostrand.auto.stack nostrand

/etc/apparmor.d/usr.sbin.named:

AppArmor er et sikkerhetssystem i Linux som, blant annet, håndterer tilgang til filer. Denne hindrer Bind å gjøre endringer på konfigurasjonsfilene sine, og vi må derfor endre en innstilling i AppArmor:
Åpne filen `/etc/apparmor.d/usr.sbin.named` og endre linjen
`/etc/bind/** r,`
til
`/etc/bind/** rw,`

Etter at alle disse innstillingene er på plass må tjenesten restartes: `service bind9 restart,` og DNS skal være oppe å kjøre!

Test til slutt at det fungerer ved å kjøre `nslookup`:



Fungerer begge veier!

Installasjon av Foreman

Installasjon av Foreman går gjennom et eget installasjonsprogram. Det lastes ned på den offisielle nettsiden og kjøres i kommandolinjen. Først må man legge til og pakke ut alle de nødvendige arkivene:

```
apt-get -y install ca-certificates
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb
dpkg -i puppetlabs-release-trusty.deb
echo "deb http://deb.theforeman.org/ trusty 1.7" > /etc/apt/sources.list.d/foreman.list
echo "deb http://deb.theforeman.org/ plugins 1.7" >>
/etc/apt/sources.list.d/foreman.list
wget -q http://deb.theforeman.org/pubkey.gpg -O- | apt-key add -
apt-get update && apt-get -y install foreman-installer
```

Merk at trusty er kodenavnet til Ubuntu 14.04, og må byttes dersom annet operativsystem benyttes. 1.7 er versjonen av Foreman, og den mest stabile og best testede versjonen for løsningen vår.

Etter at dette er gjort så skal selve installasjonen kjøres. Her er det viktig at kommandoen

```
ping $(hostname -f)
```

viser en ekstern ip-adresse, ikke 127.0.0.1 og at . Dersom det ikke stemmer, er det bare å endre innholdet i `/etc/hosts`, slik at det vises riktig. Da alt stemmer, skal det bare være å kjøre kommandoen

```
foreman-installer
```

Denne prosessen er automatisk, og sier i fra når den er ferdig med å kjøre. Den vil også skrive ut den initielle administrator-brukerkontoen med passord til slutt, så terminalutskriften er viktig å få med seg. Dette passordet kan endres senere i web-grensesnittet til Foreman.

Konfigurasjon av Foreman-proxier

Foreman kommer med det meste ferdigkonfigurert, men noen ting mangler for å få distribueringssystemet vi trenger helt automatisert. Dette er smart-proxyer. Disse håndterer nettverkstjenester for oss, slik at det er minst mulig manuell jobb med konfigurasjon av disse tjenestene ved utrulling. Her støtter Foreman både TFTP, DNS, DHCP, Puppet og Puppet CA (sertifikatshåndtering). Ved installasjon er det bare TFTP, Puppet og Puppet CA som er aktivert. DHCP og DNS (Som er nødvendig å styre fra Foreman ved deployment) må aktiveres manuelt. Dette gjøres gjennom en dedikert tjeneste; foreman-proxy.

I mappen `/etc/foreman-proxy/settings.d/` ligger alle innstillingene for denne tjenesten. Det er her de konfigureres og settes opp til de ulike funksjonene.

I filen `/etc/foreman-proxy/settings.d/dhcp.yml` skal det se slik ut:

```
:enabled: true
:dhcp_config: /etc/dhcp/dhcpd.conf
:dhcp_leases: /var/lib/dhcp/dhcpd.leases
```

I filen `/etc/foreman-proxy/settings.d/dns.yml` skal det se slik ut:

```
:enabled: true
#:dns_key:
```

Til slutt må denne kommandoen kjøres for å laste inn de nye innstillingene og skru på tjenestene:

```
service foreman-proxy restart
```

Da dette er gjort, er det bare å gå på fanen “Infrastructure -> Smart proxies” inne i foreman og trykke på “Refresh features” ved siden av Nostrand for å oppdatere listen, så skal de nye tjenestene dukke opp.

Puppet management

Etter at installasjonen av Foreman er ferdig skal det være satt opp en Puppetmaster på Nostrand, som er fullt integrert med Foreman. For å sende den første statusrapporten til Foreman må man først kjøre Puppet kommandoen:

```
puppet agent --test
```

Når Puppet agent kommandoen kjører vil det komme opp en warning første gangen kommandoen kjøres. Denne meldingen forsvinner andre gangen man kjører kommandoen, og varselmeldingen kan derfor bare ignoreres.

I Foreman gå under '*Hosts'* fanen og klikk på '*All hosts'*. Når Puppet agent har kjørt ferdig skal hosten dukke opp her med statusen "O". Dette viser at din host er "OK" og at ingen endringer har blitt gjort siden siste statusrapport.

PXE boot etter første installering
Under installeringen av en ny node bruker Foreman forskjellige templates for å konfigurere serveren. Vi bruker de fleste default-templatene, men en av de må endres: "Preseed default" under "Partition tables". Her må linjen "`d-i partman-auto/disk string /dev/sda /dev/vda`" endres til "`d-i partman-auto/disk string /dev/sda /dev/vda /dev/cciss/c0d0`". Mer om dette i [*link/referanse til utfordringer /dev/cciss, testing*]

PXELinux localboot må endres inne i foreman -- provisioning templates til dette:
Når den automatiske installasjonen av andre noder er ferdig, vil ikke våre maskiner kunne starte opp fra harddisk automatisk etterpå. Det er på grunn av at `/var/lib/tftpboot/pxelinux.cfg/<MAC-addresse>` ikke er riktig (en fil for hver host/node). Den inneholder følgende:

```
DEFAULT menu
PROMPT 0
MENU TITLE PXE Menu
TIMEOUT 200
TOTALTIMEOUT 6000
ONTIMEOUT local

LABEL local
MENU LABEL (local)
MENU DEFAULT
LOCALBOOT 0
```

Dette innholdet kan endres inne i Foreman så det gjelder alle hosts. Via "Hosts" -> "Provisioning templates" -> "PXELinux default localboot" endres dette til:

```
DEFAULT menu
PROMPT 0
```

```
MENU TITLE PXE Menu
TIMEOUT 100
TOTALTIMEOUT 6000
ONTIMEOUT local

LABEL local
    MENU LABEL (LocalBoot)
    MENU DEFAULT
    COM32 chain.c32
<% if @host.shortname == 'compute3' || @host.shortname == 'compute4' ||
@host.shortname == 'compute5' -%>
    APPEND hd0
<% else -%>
    APPEND hd0 0
<% end -%>
```

Denne koden skal sørge for at de to ulike hp serverne DL380 og DL320 skal klare å boote automatisk etter installering av OS. Den fungerer også på andre maskiner, da vi har testet flere forskjellige i oppsettet vårt.

Den eneste forskjellen mellom de to er: APPEND hd0 0 eller APPEND hd0. Forklaringen på dette kan være så enkel som at DL380 kjører RAID mens DL320 maskinen bare har en harddisk. Dette har vi ikke fått bekreftet.

http://theforeman.org/manuals/1.7/index.html#2.2PuppetManagement
http://projects.theforeman.org/projects/foreman/wiki/TemplateWriting
Dette er siden vi fant syntaksen for å lage dynamisk oppstartskode på.

Klargjøre utrulling i Foreman

For å starte opp en maskin gjennom nettverket, trenger den et OS å installere. Foreman bruker flere forskjellige 'templates' for å gjøre dette på riktig måte. De templatene man trenger for å installere et fullverdig operativsystem er provision template, finish template og PXELinux template. Disse ligger i fanen '***Hosts*** → ***Provisioning templates***'. Her må templatene '*Preseed default*', '*Preseed default finish*' og '*Preseed default PXELinux*' oppdateres. Inne i hver av dem, gå til fanen '***Association***' og velg '*Ubuntu 14.04 LTS*'. Trykk på knappen '***Submit***'.

Nå skal Operativsystemet være oppdatert, og neste steg er å ordne resten av instilingene til OSet. For å gjøre dette, går man til '***Hosts*** → ***Operating systems***' og velger '*Ubuntu 14.04 LTS'*. Kryss av '*x86_64*' under '***Architectures***' og gå til fanen '***Partition table***'. ' Velg '*Preseed default*' og gå til fanen '***Installation media***'. Velg '*Ubuntu mirror*' og gå til fanen '***Templates***'. Fyll alle boksene med valg, og trykk på knappen '*Submit*'. Da skal Operativsystemet være klart.

Nå som selve OSet er ferdig konfigurert, skal utrullingen klargjøres. Gå til fanen '***Infrastructure*** → ***Provisioning setup***'. Velg det nettverksinterfacet som har adressen 10.0.0.0/24 (burde være eth2) og trykk submit. På neste side fyll ut dette:

Name	→ AutoStack
Gateway address	→ 10.0.0.1
Secondary DNS server	→ 8.8.8.8
Start of IP range	→ 10.0.0.2
End of IP range	→ 10.0.0.99
Boot mode	→ DHCP

Trykk på '***Nex***t' og '***Submit***'.

Det siste som må gjøres er å skru på automatisk DNS-registrering. Gå til '***Infrastructure*** → ***Domains***' og velg '*auto.stack'*, sett DNS Proxy til '*nostrand.auto.stack'* og trykk '***Submit***'. Nå er alt klart for provisjonering.

Oppdatering til OpenStack IceHouse

15.8.1 Oppdatering av AutoStack til å bruke OpenStack IceHouse

Det første vi gjorde var å installere IceHouse versjonen av Puppetlabs-OpenStack modulene: (Puppetlabs sine OpenStack-moduler stoppet opp med versjonen IceHouse, og vil sannsynligvis alltid være gjeldende.)

```
puppet module install puppetlabs-ntp
puppet module install puppetlabs-nova
puppet module install puppetlabs-neutron
puppet module install puppetlabs-cinder
puppet module uninstall --force puppetlabs-rabbitmq
puppet module install puppetlabs-rabbitmq --version 2.1.0
puppet module uninstall --force puppetlabs-mysql
puppet module install puppetlabs-mysql --version 0.9.0
puppet module install puppetlabs-heat
puppet module install saz-ssh --version 2.4.0
```

I mappen `/etc/puppet/environments/production/manifests/` skal gruppens site.pp legges inn for IceHouse.
Utrullingsskriptene for nettverk (deployNetwork.sh), kontroller (deployController.sh) og (deployCompute.sh) compute-nodene skal lagres i `/etc/puppet/modules/nova/files`.

Endringer gjort fra Havana til IceHouse

Fra OpenStack Havana til OpenStack IceHouse er det blitt lagt til kode for å oppgradere eksisterende funksjonalitet, eller legge til ny funksjonalitet. Når gruppen skulle oppgradere fra Havana til IceHouse var det de følgende endringene som ble gjort fra AutoStack til IceHouse sin site.pp.

node controller:

Gjør at noden kan nås via SSH

```
class { 'ssh::server':
       options => {
       'PermitRootLogin' => 'yes',
       }
}
```

Skrevet inn faktiske NTP adresser

```
class { '::ntp':
       servers => [ '0.debian.pool.ntp.org', '1.debian.pool.ntp.org' ],
}
```

Extra configuration of the api file (IceHouse)

```
class { 'glance::config':
       api_config => { 'DEFAULT/rpc_backend' => { value => 'rabbit'},
                       'DEFAULT/rabbit_host' => { value => '10.0.0.2'},
                       'DEFAULT/rabbit_password' => { value => 'SRVPW'},
              }
}
```

Endret `rpc_backend` fra `'nova.rpc.impl_kombu'` til `'rabbit'`

```
class { 'nova':
       sql_connection => false,
       database_connection => 'mysql://nova:SRVPW@10.0.0.2/nova',
       rpc_backend => 'rabbit',
}
```

Lagt til `core_plugin => 'ml2'` og `service_plugins => ['router']`

```
class { '::neutron':
       enabled => true,
       bind_host => '0.0.0.0',
       rabbit_host => '10.0.0.2',
       rabbit_user => 'guest',
       rabbit_password => 'SRVPW',
       rabbit_hosts => false,
       verbose => false,
       debug => false,
       core_plugin => 'ml2',
       service_plugins => ['router'],
}
```

Lagt til `neutron::server::notifications` klassen

```
class { 'neutron::server::notifications':
       notify_nova_on_port_status_changes => 'True',
       notify_nova_on_port_data_changes => 'True',
       nova_url => 'http://10.0.0.2:8774/v2',
       nova_admin_auth_url => 'http://10.0.0.2:35357/v2.0',
       nova_admin_username => 'nova',
       nova_admin_tenant_name => 'services',
       nova_admin_password => 'SRVPW',
       nova_region_name => 'regionOne',
}
```

Lagt til `neutron::plugins::ml2` klassen

```
class { 'neutron::plugins::ml2':
       type_drivers        => ['vlan'],
       tenant_network_types => ['vlan'],
       mechanism_drivers   => ['openvswitch'],
       network_vlan_ranges  => ['physnet1:2000:2099'],
}
```

Fjernet `tenant_network_type => ‘vlan’`

```
class { 'neutron::plugins::ovs':
       network_vlan_ranges => 'physnet1:2000:2099',
}
```

Fjernet `firewall_driver =>` siden denne blir lagt inn by default

```
class { 'neutron::agents::ovs':
}
```

Legger til exec admin_tenant_id som sørger for at tenant id blir satt i nova_admin_tenant_id i /etc/neutron/neutron.conf

```
exec { 'admin_tenant_id':
       command => "tenantID=\$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})
              sed -i \"s|# nova\\_admin\\_tenant\\_id =|nova\\_admin\\_tenant\\_id
=\$tenantID|g\" /etc/neutron/neutron.conf
      service neutron-server restart",
       onlyif  => “[ -n \”$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})\” ] && cat /etc/neutron/neutron.conf | grep '#
nova_admin_tenant_id’”,
       provider => 'shell'
}
```

Lagt til Heat-klasser

```
#######################
######## HEAT ########
#######################

class { 'heat::db::mysql':
       host          => '10.0.0.2',
       password      => 'SRVPW',
}

class { 'heat':
       rabbit_host         => '10.0.0.2',
       rabbit_password     => 'SRVPW',
       sql_connection      => 'mysql://heat:SRVPW@10.0.0.2/heat',
       keystone_host       => '10.0.0.2',
       keystone_port       => '35357',
       keystone_protocol   => 'http',
       keystone_user       => 'heat',
       keystone_tenant     => 'services',
       keystone_ec2_uri    => 'http://10.0.0.2:5000/v2.0',
       keystone_password   => 'SRVPW',
}

class { 'heat::keystone::auth':
```

```
        password            => 'SRVPW',
        public_address      => '10.0.0.2',
        admin_address       => '10.0.0.2',
        internal_address    => '10.0.0.2',
        region              => 'regionOne',
}

class { 'heat::engine':
        auth_encryption_key        => 'notgood but just long enough i think',
        heat_metadata_server_url   => 'http://10.0.0.2:8000',
        heat_waitcondition_server_url => 'http://10.0.0.2:8000/v1/waitcondition',
}

class { 'heat::api': }
```

Lagt til `'heat'`, `'heat::db::mysql'`, `'heat::engine'`, `'heat::api'` og `'heat::keystone::auth'` som skal være på plass før scriptet får lov til å kjøre.

```
########################
#### deploy script ###
########################
# The controller deploy script is copied to the controller
file {'/etc/puppet/deployController.sh':
        ensure => directory,
        owner => 'root',
        group => 'root',
        mode => '0755',
        source => 'puppet:///modules/nova/deployController.sh',
        # Listing requirements to ensure that the script is not
        # copied and executed before depended service are in place.
        require => Class[ '::neutron',
                        'nova',
                        'neutron::server',
                        'nova::api',
                        'glance::api',
                        'nova::rabbitmq',
                        'heat',
                        'heat::db::mysql',
                        'heat::engine',
                        'heat::api',
                        'heat::keystone::auth' ],
}
```

node network:

Gjør at noden kan nåes via ssh

```
class { 'ssh::server':
       options => {
       'PermitRootLogin' => 'yes',
       }
}
```

Skrevet inn faktiske NTP adresser

```
class { '::ntp':
       servers => [ '0.debian.pool.ntp.org', '1.debian.pool.ntp.org' ],
}
```

Lagt til `core_plugin => 'ml2'` og `service_plugins => ['router']`

```
class { '::neutron':
       enabled => true,
       bind_host => '0.0.0.0',
       rabbit_host => '10.0.0.2',
       rabbit_user => 'guest',
       rabbit_password => 'SRVPW',
       rabbit_hosts => false,
       verbose => false,
       debug => false,
       core_plugin => 'ml2',
       service_plugins => ['router'],
}
```

Lagt til `neutron::server::notifications` klassen

```
class { 'neutron::server::notifications':
       notify_nova_on_port_status_changes => 'True',
       notify_nova_on_port_data_changes => 'True',
       nova_url => 'http://10.0.0.2:8774/v2',
       nova_admin_auth_url => 'http://10.0.0.2:35357/v2.0',
       nova_admin_username => 'nova',
       nova_admin_tenant_name => 'services',
       nova_admin_password => 'SRVPW',
       nova_region_name => 'regionOne',
}
```

Lagt til `neutron::plugins::ml2` klassen

```
class { 'neutron::plugins::ml2':
       type_drivers        => ['vlan'],
       tenant_network_types => ['vlan'],
       mechanism_drivers   => ['openvswitch'],
       network_vlan_ranges  => ['physnet1:2000:2099'],
}
```

Fjernet `tenant_network_type => ‘vlan’`

```
class { 'neutron::plugins::ovs':
       network_vlan_ranges => 'physnet1:2000:2099',
}
```

Lagt til `interface_driver` og `use_namespaces` i klassen `neutron::agents::l3`

```
# Configuration of l3_agent.ini (IceHouse).
class { 'neutron::agents::l3':
       interface_driver => 'neutron.agent.linux.interface.OVSInterfaceDriver',
       use_namespaces => 'True',
}
```

Fjernet `firewall_driver =>` siden denne blir lagt inn by default

```
class { 'neutron::agents::ovs':
}
```

Legger til exec admin_tenant_id som sørger for at tenant id blir satt i nova_admin_tenant_id i /etc/neutron/neutron.conf

```
exec { 'admin_tenant_id':
       command => "tenantID=\$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})
             sed -i \"s|# nova\\_admin\\_tenant\\_id =|nova\\_admin\\_tenant\\_id
=\$tenantID|g\" /etc/neutron/neutron.conf
      service neutron-server restart",
       onlyif  => “[ -n \”$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})\” ] && cat /etc/neutron/neutron.conf | grep '#
nova_admin_tenant_id’”,
       provider => 'shell'
}
```

node compute:

Gjør at noden kan nåes via ssh

```
class { 'ssh::server':
       options => {
       'PermitRootLogin' => 'yes',
       }
}
```

Skrevet inn faktiske NTP adresser

```
class { '::ntp':
       servers => [ '0.debian.pool.ntp.org', '1.debian.pool.ntp.org' ],
}
```

Endret `rpc_backend` fra `'nova.rpc.impl_kombu'` til `'rabbit'`

```
class { 'nova':
       database_connection => 'mysql://nova:SRVPW@10.0.0.2/nova',
       rabbit_userid => 'guest',
       rabbit_password => 'SRVPW',
       image_service => 'nova.image.glance.GlanceImageService',
       glance_api_servers => '10.0.0.2:9292',
       verbose => false,
       rabbit_host => '10.0.0.2',
       rpc_backend => 'rabbit',
}
```

Endret `core_plugin` fra `'neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2'` til `'ml2'`, og la til `service_plugins => ['router']`

```
class { '::neutron':
       enabled => false,
       rabbit_host => '10.0.0.2',
       rabbit_user => 'guest',
       rabbit_password => 'SRVPW',
       core_plugin => 'ml2',
       service_plugins => ['router'],
}
```

Lagt til `neutron::server::notifications` klassen

```
class { 'neutron::server::notifications':
       notify_nova_on_port_status_changes => 'True',
       notify_nova_on_port_data_changes => 'True',
       nova_url => 'http://10.0.0.2:8774/v2',
       nova_admin_auth_url => 'http://10.0.0.2:35357/v2.0',
       nova_admin_username => 'nova',
       nova_admin_tenant_name => 'services',
       nova_admin_password => 'SRVPW',
       nova_region_name => 'regionOne',
}
```

Lagt til `neutron::plugins::ml2` klassen

```
class { 'neutron::plugins::ml2':
       type_drivers        => ['vlan'],
       tenant_network_types => ['vlan'],
       mechanism_drivers   => ['openvswitch'],
       network_vlan_ranges  => ['physnet1:2000:2099'],
}
```

Fjernet `tenant_network_type => ‘vlan’`

```
class { 'neutron::plugins::ovs':
       network_vlan_ranges => 'physnet1:2000:2099',
}
```

Fjernet `firewall_driver =>` siden denne blir lagt inn by default

```
class { 'neutron::agents::ovs':
}
```

Legger til exec admin_tenant_id som sørger for at tenant id blir satt i nova_admin_tenant_id i /etc/neutron/neutron.conf

```
exec { 'admin_tenant_id':
       command => "tenantID=\$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})
              sed -i \"s|# nova\\_admin\\_tenant\\_id =|nova\\_admin\\_tenant\\_id
=\$tenantID|g\" /etc/neutron/neutron.conf
      service neutron-server restart",
```

```
      onlyif  => “[ -n \”$(keystone --os-tenant-name openstack --os-username admin
--os-password SRVPW --os-auth-url http://10.0.0.2:5000/v2.0 tenant-list | grep
'services' | awk {'print \$2'})\” ] && cat /etc/neutron/neutron.conf | grep '#
nova_admin_tenant_id’”,
      provider => 'shell'
}
```

Oppdatering til OpenStack Juno

Først sletter vi de gamle modulene. Disse er utdaterte og kan ikke brukes lenger:

```
puppet module uninstall puppetlabs-nova --force
puppet module uninstall puppetlabs-neutron --ignore-changes
puppet module uninstall puppetlabs-cinder
puppet module uninstall puppetlabs-glance
puppet module uninstall puppetlabs-heat
puppet module uninstall puppetlabs-vswitch
puppet module uninstall puppetlabs-keystone
puppet module uninstall puppetlabs-mysql
puppet module uninstall puppetlabs-rabbitmq
```

Så installerer vi de nye modulene for Juno:

```
puppet module install stackforge-nova
puppet module install stackforge-neutron
puppet module install stackforge-heat
puppet module install puppetlabs-rabbitmq --version 2.1.0 --force
```

Så sletter vi grunnkomponentene for oppdatering:

```
cd /etc/puppet/modules
rm -rf nova neutron glance heat cinder
```

Vi installerer git som brukes for å hente siste oppdatering fra Github:

```
apt-get install git
```

Vi hentet de nyeste puppet-modulene direkte fra Github, siden de som ligger i Puppet Forge er utdaterte. Vi må fremdeles installere fra Puppet Forge først, siden det er viktig å få med seg avhengigheter til andre moduler.
Modulene legges fortsatt i `/etc/puppet/modules.`

```
cd /etc/puppet/modules/
git clone https://github.com/stackforge/puppet-nova.git nova
git clone https://github.com/stackforge/puppet-neutron.git neutron
git clone https://github.com/stackforge/puppet-glance.git glance
git clone https://github.com/stackforge/puppet-heat.git heat
git clone https://github.com/stackforge/puppet-cinder.git cinder
```

I sine vedlegg har gruppen Puppet-koden for sin modul.
For å implementere dette må man lage en ny modul-mappe under /etc/puppet/modules:

```
cd /etc/puppet/modules
mkdir autostack
```

I `/etc/puppet/modules/autostack` kopierer man inn de syv forskjellige .pp filene som utgjør modulen; params.pp, init.pp, common.pp, controller.pp, network.pp, compute.pp, og storage.pp.

Endringer fra IceHouse til Juno

Gruppen begynte med en automatisk implementasjon av OpenStack Juno gjennom Puppet. Slik som tidligere beskrevet i den automatiske implementasjonen i IceHouse, tar gruppen også her for seg bare de endringene som er gjort fra site.pp for IceHouse til site.pp for Juno.

I begynnelsen i site.pp for Juno har gruppen lagt til variabler som brukes ofte gjennom site.pp. Dermed kan oppdragsgiver bare endre variabelene slik han ønsker i stedet for å gå igjennom hele site.pp, og endre hver ting flere ganger. Ikke bli forvirret av at `IP settings` variablene står i flertall. Dette er fordi navn som “netmask” og “gateway” allerede er i bruk:

```
# IP address of the nodes.
$controller     = "10.0.0.2"
$network        = "$::ipaddress"
$compute        = "$::ipaddress"

# IP settings for OpenStack.
$netmasks       = "255.255.255.0"
$gateways       = "10.0.0.1"
$nameservers    = "10.0.0.1 8.8.8.8"
$networks       = "10.0.0.0"
$broadcasts     = "10.0.0.255"

# Interface ports.
$eth0 = "eth0"
$eth1 = "eth1"
$eth2 = "eth2"

# Credentials.
$pass = "SRVPW"
$tenantName = "services"

# Region.
$regionName = "regionOne"

# Network type (VLAN or GRE).
$networkType = "GRE"
```

Vi har laget en “common classes” seksjon i site.pp for Juno, hvor puppet-kode som må kjøres på alle noder (kontroller, nettverk, og compute) ligger. Vi gjorde dette for å unngå så mye duplisering av kode som mulig:

```
########################
### COMMON CLASSES ###
########################
```

```
# Configured classes that all the three nodes have in common.

################
### APT-GET ###
################
```

Vi har laget en klasse kalt “init” som inneholder den programvaren og pakkene som kreves for å oppgradere OpenStack fra IceHouse til Juno. Dette gjøres på alle noder, og er derfor i “common classes” seksjonen:

```
class init {
# Installs the ubuntu-cloud-keyring.
exec { 'ubuntu-cloud-keyring':
  command  => '/usr/bin/apt-get install ubuntu-cloud-keyring',
  unless   => '/usr/bin/dpkg --get-selections | /bin/grep ubuntu-cloud-keyring',
}

# Adds the Juno repositories.
exec { 'echo-rep':
  command => '/bin/echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu
trusty-updates/juno main" > /etc/apt/sources.list.d/cloudarchive-juno.list',
  unless  => '/bin/grep "juno main" /etc/apt/sources.list.d/cloudarchive-juno.list',
}

# Installs OpenStack packages.
exec { 'apt-update':
  require => Exec['echo-rep','ubuntu-cloud-keyring'],
  command => '/usr/bin/apt-get update',
  unless  => '/usr/bin/find /etc/neutron/',
}
}
```

Vi lager en stage som vi kaller for “first”. Dette sørger for at klassen “init” ovenfor kjører sin kode før alt annet i site.pp:

```
# Makes sure that the class 'init' runs before everything else.
stage { 'first':
before => Stage['main'],
}
```

All Neutron konfigurasjon var satt i hver node i site.pp for IceHouse. Nå har en stor del av Neutron konfigurasjonen (ikke absolutt alt) blitt flyttet inn i “common classes” seksjonen, og lagt det inn i en klasse vi har kalt “basic_neutron”. Vi har lagt til “allow_overlapping_ips” for GRE-nettverk, “rpc_backend”, og “rabbit_port”. Vi har fjernet “bind_host”, “rabbit_hosts”:

```
###############
### NEUTRON ###
###############

# Installation and basic configuration of the Neutron service.
class basic_neutron {
if $networkType == 'GRE' {
  $var = 'true'
}
else {
  $var = 'true'
}

class { '::neutron':
  allow_overlapping_ips       => $var,                 #GRE
  enabled                     => true,
  verbose                     => true,
  debug                       => true,
  core_plugin                 => 'ml2',
  service_plugins             => ['router'],
  rpc_backend                 => 'neutron.openstack.common.rpc.impl_kombu',
  rabbit_host                 => $controller,
  rabbit_port                 => '5672',
  rabbit_user                 => 'guest',
  rabbit_password             => $pass,
}
}
```

ML2 konfigurasjonen var også satt på alle noder, og ble flyttet til “common classes” seksjonen. Vi har fjernet klassen “neutron::agents::ovs”, og klassen “neutron::plugins::ovs”. Vi har lagt til “neutron::agents::ml2::ovs”, “bridge” variabel, støtte for både VLAN og GRE nettverk, “require” atributt, “tunnel_id_ranges”, exec “echo-ovs” som konfigurerer resten av “ovs” seksjonen i ml2:

```
############
### ML2 ###
############

# Installation and confiugration of the ML2-plugin and the [ovs] section for Neutron
(VLAN).
if $controller == $::ipaddress {
$bridge = []
}
else {
$bridge = ['physnet1:br-eth1']
}
```

```
class ml2_vlan {
# Installation an configuration of the ML2-plugin for Neutron.
class { 'neutron::plugins::ml2':
  require                 => Class['neutron'],
  type_drivers          => ['vlan'],
  tenant_network_types  => ['vlan'],
  tunnel_id_ranges        => [],
  mechanism_drivers     => ['openvswitch'],
  network_vlan_ranges   => ['physnet1:2000:2099'],
}

# Adds and configures the [ovs] section for the ml2 plugin.
class {'neutron::agents::ml2::ovs':
  bridge_mappings => $bridge,
  tunnel_types    => false,
  local_ip        => $::ipaddress,
}

# Further configures the [ovs] section for the ml2 plugin.
exec { 'echo-ovs':
  command => '/bin/echo "network_vlan_ranges = physnet1:2000:2099" >>
/etc/neutron/plugins/ml2/ml2_conf.ini;
      /bin/echo "tenant_network_type = vlan" >> /etc/neutron/plugins/ml2/ml2_conf.ini',
  require => Class['neutron::plugins::ml2'],
  unless  => '/bin/grep "tenant_network_type = vlan"
/etc/neutron/plugins/ml2/ml2_conf.ini',
}
}

# Installation and configuration of the ml2 plugin and the [ovs] section for Neutron
(GRE-TUNNELS).
class ml2_gre {

# Installation and configuration of the ML2-plugin for Neutron.
class { 'neutron::plugins::ml2':
  type_drivers                => ['flat','gre'],
  tenant_network_types        => ['gre'],
  mechanism_drivers           => ['openvswitch'],
  tunnel_id_ranges            => ['1:1000'],
}

class {'neutron::agents::ml2::ovs':
  bridge_mappings             => ['external:br-ex'],
  local_ip                    => $::ipaddress,
  enable_tunneling            => 'true',
  tunnel_types                => ['gre'],
}
```

```
}
```

SSH og NTP konfigurasjon ble også flyttet til “common classes” seksjonen. Vi laget en egen klasse kalt “ssh” for SSH, og en klasse kalt “ntp” for NTP:

```
############
### SSH ###
############

# Configures sshd_config to permit root login.
class ssh {
class { 'ssh::server':
  options => {
    'PermitRootLogin' => 'yes',
  }
}
}

##############
#### NTP ####
##############

# To synchronize services across multiple machines, NTP is installed.
# The compute node fetches time data from the controller in first instance.
class ntp {
class { '::ntp':
  servers => [ $controller, '0.debian.pool.ntp.org', '1.debian.pool.ntp.org' ],
}
}
```

Kontroller-noden:

I Keystone sin konfigurasjon av "admin-roles" påloggingsinformasjon, la vi til en atributt kalt "admin_roles". Vi måtte ha med denne atributten for at Heat tjenesten kunne få tilgang til ressursene:

```
# The Admin-role credentials is set.
class { 'keystone::roles::admin':
  email       => 'example@example.com',
  password    => $pass,
  admin_roles => ['heat_stack_owner','admin'],
}
```

I Glance sin konfigurasjon av sitt API, har vi gjort noen få endringer. Vi byttet ut "sql_connection" med "database_connection", fjernet "auth_host" siden den er utdatert i Juno, la til "auth_uri", og la til "identity_uri":

```
# Configuration of the api file for Glance.
# Ensures Glance is installed.
class { 'glance::api':
  verbose             => true,
  keystone_tenant     => $tenantName,
  keystone_user       => 'glance',
  keystone_password   => $pass,
  database_connection => "mysql://glance:$pass@$controller/glance",
  pipeline            => 'keystone',
  auth_uri            => "http://$controller:5000/v2.0",
  identity_uri        => "http://$controller:35357",
}
```

La til litt ekstra konfigurasjon av Glance sitt API enn det som var der fra før:

```
# Extra configuration of the api file.
class { 'glance::config':
  api_config => { 'DEFAULT/rpc_backend'                       => { value => 'rabbit'},
                  'DEFAULT/rabbit_host'                       => { value =>
$controller},
                  'DEFAULT/rabbit_password'                   => { value => $pass},
                  'glance_store/default_store'                => { value => 'file'},
                  'glance_store/filesystem__store_datadir'    => { value =>
'/var/lib/glance/images/'},
                }
}
```

I konfigurasjonen for Glance sin "registry" fil, la vi til "identity_uri", la til "auth_uri", fjernet "auth_host", og byttet ut "sql_connection" med "database_connection":

```
# Configuration of the registry file for Glance.
class { 'glance::registry':
  identity_uri        => "http://$controller:35357",
  verbose             => true,
  keystone_tenant     => $tenantName,
  keystone_user       => 'glance',
  keystone_password   => $pass,
  auth_uri            => "http://$controller:5000/v2.0",
  database_connection => "mysql://glance:$pass@$controller/glance",
}
```

Vi installerer RabbitMQ:

```
# Installs rabbitmq.
class {'nova::rabbitmq':
 password            => $pass,
}
```

Under konfigurasjon av Nova, har vi fjernet “sql_connection”, endret “rpc_backend”, og rabbit innstillinger:

```
# Install and configure Nova.
class { 'nova':
 database_connection => "mysql://nova:$pass@$controller/nova",
 rpc_backend         => 'nova.openstack.common.rpc.impl_kombu',
 rabbit_userid       => 'guest',
 rabbit_host         => $controller,
 rabbit_password     => $pass,
 rabbit_port         => '5672',
}
```

Fra klassen “nova::api” har vi fjernet “auth_strategy”. I stedet for “auth_host”, “auth_port”, “auth_protocol” har vi lagt til valget “identity_uri”:

```
# Install and configure nova-api.
class { 'nova::api':
 enabled                              => true,
 admin_password                       => $pass,
 neutron_metadata_proxy_shared_secret => 'METADATA_PASS',
 auth_uri                             => "http://$controller:5000/v2.0",
 admin_tenant_name                    => $tenantName,
 admin_user                           => 'nova',
 identity_uri                         => "http://$controller:35357",
}
```

Fjernet klassen “nova::rabbitmq’. Rabbitmq settes nå opp manuelt:

```
# Install Nova components.
class { 'nova::scheduler': enabled   => true }
class { 'nova::cert': enabled        => true }
class { 'nova::vncproxy': enabled    => true }
class { 'nova::consoleauth': enabled => true }
class { 'nova::conductor': enabled   => true }
```

Lagt til kodelinjen for å kjøre “basic_neutron” klassen:

```
# Installation of the Neutron service with configuration
# of message-queue server.
class { 'basic_neutron': }
```

I klassen “neutron::server” har vi fjernet “sql_connection”, “connection”, og “auth_host”. Vi har lagt til “sync_db”, “identity_uri”, “auth_uri”, og “database_connection”

```
# Configures authentication and database connection for the Neutron service.
class { 'neutron::server':
 sync_db             => 'false',
 identity_uri        => "http://$controller:35357",
 auth_uri            => "http://$controller:5000/v2.0",
 auth_password       => $pass,
 database_connection => "mysql://neutron:$pass@$controller/neutron",
}
```

Kjører ML2 konfigurasjon for VLAN eller GRE:

```
# Runs the configuration for the ML2-plugin.
if $networkType == 'GRE' {
 class { 'ml2_gre': }
}
else {
 class { 'ml2_vlan': }
}
```

Fjernet hele “admin_tenant_id” exec, siden den ikke lenger er nødvendig i Juno.

I klassen “heat” fjernet vi “keystone_host”, “keystone_port”, og “keystone_protocol” som ble bytttet ut med “identity_uri”. Vi la også til “auth_uri”:

```
# Installs and configures the Heat service.
class { 'heat':
 identity_uri        => "http://$controller:35357",
 auth_uri            => "http://$controller:5000/v2.0",
```

```
 rabbit_host         => $controller,
 rabbit_password     => $pass,
 sql_connection      => "mysql://heat:$pass@$controller/heat",
 keystone_user       => 'heat',
 keystone_tenant     => $tenantName,
 keystone_ec2_uri    => "http://$controller:5000/v2.0",
 keystone_password   => $pass,
}
```

Kjører NTP konfigurasjonen:

```
# Runs the ntp configurations.
class { 'ntp':}
```

Kjører “init” klassen med stage “first”:

```
# Runs the apt-get configurations.
class { 'init':
 stage => first,
}
```

“deplyController.sh” er nå integrert inn i site.pp for Juno, så “Deploy script” seksjonen i site.pp fra IceHouse er fjernet i Juno.

Nettverks-noden:

Kjørte SSH konfigurasjonen:

```
# Runs the ssh configurations.
class { 'ssh':}
```

Kjørte NTP konfigurasjonen:

```
# Runs the ntp configurations.
class { 'ntp':}
```

Kjørte "init" klassen med stage "first":

```
# Runs the apt-get configurations.
class { 'init':
 stage => first,
}
```

Kjørte klassen "basic_neutron":

```
# Installation of the Neutron service with configuration
# of message-queue server.

class { 'basic_neutron': }
```

I klassen "neutron::server" fjernet vi "auth_host", "sql_connection", og "connection". Vi la til "identity_uri", "auth_uri", og "database_connection":

```
# Configures authentication and database connection for the Neutron service.
class { 'neutron::server':
 auth_uri      => "http://$controller:5000/v2.0",
 auth_password => $pass,
 identity_uri  => "http://$controller:35357",
 database_connection => "mysql://neutron:$pass@$controller/neutron",
}
```

Så satte vi opp slik at konfigurasjonen kjørte med VLAN eller GRE konfigurasjon:

```
# Runs the configuration for the ML2-plugin, based on network type
if $networkType == 'VLAN'{
 class { 'ml2_vlan':}
}
else {
 class {'ml2_gre':}
}
```

Fjernet klassen “neutron::plugins::ovs”, og “neutron::agents::ovs”.
La til GRE støtte i klassen “neutron::agents::l3”:

```
# Configuration of l3_agent.ini.

 if $networkType == 'GRE' {
   $var3 = 'true'
 }
 else {
   $var3 = 'false'
 }

class { 'neutron::agents::l3':
 interface_driver => 'neutron.agent.linux.interface.OVSInterfaceDriver',
 use_namespaces   => 'True',
 router_delete_namespaces    => $var3,
}
```

Vi la til GRE støtte i klassen “neutron::agents::dhcp”:

```
# Configuration of dhcp_agent.ini with default settings.

 if $networkType == 'GRE' {
   class { 'neutron::agents::dhcp':
     dhcp_delete_namespaces => 'true',
     dnsmasq_config_file    => '/etc/neutron/dnsmasq-neutron.conf',
   }
 }
 else {
   class { 'neutron::agents::dhcp':}
 }
```

Så fjernet exec “admin_tenant_id”, siden den ikke lenger er nødvendig i Juno.
La til exec som installerte “OVS-agent”:

```
# Installs the OVS agent.
exec { 'apt-get-ovs':
 require => Class[ 'neutron::plugins::ml2' ],
 command => '/usr/bin/apt-get install -y neutron-plugin-openvswitch-agent',
 unless  => '/bin/ps -aux | /bin/grep openvswitch',
}
```

La til en exec for dnsmasq hvis man bruker GRE nettverk:

```
# Adding the dnsmasq-neutron.conf-file for GRE networking
```

```
if $networkType == 'GRE' {
 exec { 'add-MTU-conf':
   command => "/bin/echo 'dhcp-option-force=26,1454' >
/etc/neutron/dnsmasq-neutron.conf",
   unless  => "/usr/bin/file /etc/neutron/dnsmasq-neutron.conf",
 }
}
```

La til exec for konfigurasjon av “bridger” både for VLAN og GRE nettverk, og starter nødvendige tjenester på nytt:

```
# Adding and configuration of the bridges for the OpenStack network-node, restart
necessary services.
if $networkType == "VLAN" {
 exec { 'add-ovs-conf':
   command => "/usr/bin/ovs-vsctl add-br br-ex; /usr/bin/ovs-vsctl add-port br-ex eth2;
     /usr/bin/ovs-vsctl add-br br-eth1; /usr/bin/ovs-vsctl add-port br-eth1 eth1;
     /usr/bin/service openvswitch-switch restart;
     /usr/bin/service neutron-plugin-openvswitch-agent restart;
     /usr/bin/service neutron-l3-agent restart",
   require => Exec['apt-get-ovs','interfaces'],
   unless  => "/usr/bin/ovs-vsctl show | /bin/grep eth2",
 }
}
else {
 exec { 'add-ovs-conf':
   command => "/usr/bin/ovs-vsctl add-br br-ex; /usr/bin/ovs-vsctl add-port br-ex eth2;
     /usr/bin/service openvswitch-switch restart;
     /usr/bin/service neutron-plugin-openvswitch-agent restart;
     /usr/bin/service neutron-l3-agent restart",
   require => Exec['apt-get-ovs','interfaces'],
   unless  => "/usr/bin/ovs-vsctl show | /bin/grep eth2",
 }
}
```

“deployNetwork.sh” er integrert i site.pp, og “Deploy script” er derfor fjernet fra site.pp for Juno-konfigurasjonen.

Compute-node:

Kjørte SSH konfigurasjon:

```
# Runs the ssh configurations.
class { 'ssh':}
```

Kjørte NTP konfigurasjon:

```
# Runs the ntp configurations.
class { 'ntp':}
```

Kjørte klassen “init” med stage “first”:

```
# Runs the apt-get configurations.
class { 'init':
 stage => first,
}
```

I klassen “nova” har vi endret “rpc_backend”, og lagt til “rabbit_port”:

```
# Installation and configuration of the Nova service.
class { 'nova':
 database_connection => "mysql://nova:$pass@$controller/nova",
 rabbit_userid       => 'guest',
 rabbit_password     => $pass,
 rabbit_port         => '5672',
 image_service       => 'nova.image.glance.GlanceImageService',
 glance_api_servers  => "$controller:9292",
 verbose             => false,
 rabbit_host         => $controller,
 rpc_backend         => 'nova.openstack.common.rpc.impl_kombu',
}
```

I klassen “nova::api” har vi fjernet “auth_host”. Vi har lagt til “identity_uri” og “auth_uri”:

```
# Installation and authentication configuration for the nova-api.
class { 'nova::api':
 enabled             => true,
 identity_uri        => "http://$controller:35357",
 auth_uri            => "http://$controller:5000/v2.0",
 admin_password      => $pass,
 sync_db             => 'false',
}
```

Kjørte “basic_neutron” klassen:

```
# Installation of the neutron service with message-queue and plugin configuration.
class { 'basic_neutron': }
```

I klassen “neutron::server” har vi fjernet “auth_host”, “connection”, og “sql_connection”. Vi har lagt til “identity_uri” og “database_connection”:

```
# Configuration of the neutron database connection and authentication to the
controller.
class { 'neutron::server':
 enabled             => 'false',
 identity_uri        => "http://$controller:35357",
 database_connection => "mysql://neutron:$pass@$controller/neutron",
 auth_password       => $pass,
}
```

Fjernet klassene “neutron::plugins::ovs” og “neutron::agents::ovs”.
Kjørte ML2 konfigurasjon med enten VLAN eller GRE nettverk:

```
# Runs the configuration for the ML2-plugin, based on network type
if $networkType == 'VLAN'{
 class { 'ml2_vlan':}
}
else {
 class {'ml2_gre':}
}
```

Fjernet exec “admin_tenant_id” siden den ikke er nødvendig i Juno.
“deployCompute.sh” er integrert i site.pp for Juno, og “Deploy script” seksjonen er derfor fjernet fra site.pp.
Det er alle endringer som har blitt gjort fra site.pp fra IceHouse til site.pp fra Juno. Merk at dette er eksludert koden fra alle deploy skriptene. Koden fra deploy skriptene må inn i “exec”-er i tillegg til dette.

Resultatet av alle endringene finnes i filene juno/site.pp, juno/deployController.sh, juno/deployCompute.sh og juno/deployNetwork.sh.

Oppsett av test-rammeverk-nettside i Foreman

Det første vi gjorde var å kopiere en eksempelplugin inn i Foreman, slik at vi kunne endre den selv. Naviger til mappen `/var/lib/foreman/plugins/` og klon templaten inn i egen mappe:

```
cd /var/lib/foreman/plugins
git clone https://github.com/theforeman/foreman_plugin_template rammeverk
```

Denne pluginen kommer med et lite skript som fikser navngivingen på pluginen.Gå inn i rammeverk-mappen som nå er laget og kjør skriptet:

```
cd rammeverk
./rename.rb rammeverk
```

Nå må pluginen aktiveres i Foreman. Gå til mappen `/usr/share/foreman/bundler.d/` , lag en fil kalt gemfile.local.rb og legg til disse linjene:

```
cd /usr/share/foreman/bundler.d/
nano gemfile.local.rb
```

Innhold:

```
# Gemfile.local.rb
gem 'rammeverk', :path => '/var/lib/foreman/plugins/rammeverk'
```

Deretter kjøres denne kommandoen for å installere pluginen:

```
bundle install
```

Etter at dette er gjort, så må diverse navn og instillinger endres på. I filen `/var/lib/foreman/plugins/rammeverk/lib/rammeverk/engine.rb`

- Bytt ut alle ‘my_plugin’ med ‘rammeverk’,
- Linjer som ellers skal byttes ut:

```
:url_hash => {:controller => :'rammeverk/hosts', :action => :new_action },
:caption  => 'Test-rammeverk',
:parent   => :monitor_menu,
:after => :audits
```

Nå er pluginen på plass, men vi mangler fremdeles selve nettsiden som skal vise oss resultatene. Vedlegget `new_action.html.erb` kopieres inn i mappen `/var/lib/foreman/plugins/rammeverk/app/views/rammeverk/hosts/`

Lag mappene i stien `/etc/local/apache2/cgi-bin`.

Alle test-rammeverk-skript(link) skal kopieres inn i mappen `/etc/local/apache2/cgi-bin/`. Naviger til denne mappen og kjør disse kommandoene for å fikse riktige rettigheter:

```
cd /etc/local/apache2/cgi-bin/
chown www-data *
chmod 777 *
```

```
chmod 0400 nokkel
```
www-data er brukeren til apache. Siden skriptet skal kjøres gjennom en nettside er det viktig at rettighetene er til stede.

Fjern alt fra “known_hosts”:

```
ssh-keygen -f "/root/.ssh/known_hosts"
```

Etter å ha kjørt kommandoen ovenfor. Skriv “y” for “yes”, og trykk enter to ganger.

Dersom det er IceHouse som er installert, må denne kommandoen kjøres i tillegg:
`ssh-copy-id -i nokkel 10.0.0.2`

Naviger til mappen `/var/www` og kjør disse kommandoene for å fikse rettigheter. Disse er spesielt viktige for SSH-tilkoblingen sin del.

```
cd /var/www
mkdir .ssh
chown www-data .ssh
chmod 777 .ssh
```

Naviger til mappen `/var/lib/foreman/public/` og kjør disse kommandoene for å ordne resultatfilene til testrammeverk.skriptet:

```
cd /var/lib/foreman/public
mkdir testResultater
cd testResultater
touch testrapport
chmod 666 testrapport
chown www-data testrapport
```

Siden nettsidetjeneren som kom med Foreman ikke støtter PHP, bruker vi apache sin cgi-bin funksjon for å kjøre programmer, eller i vårt tilfelle, test-rammeverk skriptet.
I filen `/etc/apache2/sites-enabled/05-foreman-ssl.conf` skal dette legges til:
(http://httpd.apache.org/docs/2.2/howto/cgi.html)

```
  ScriptAlias /cgi-bin/ /etc/local/apache2/cgi-bin/
  <Directory "/etc/local/apache2/cgi-bin">
       Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
  </Directory>
```

I tillegg kan det ta lang tid å kjøre igjennom hele test-rammeverket, så vi endrer timeout-variabelen i apache sine konfigurasjonsfiler for å støtte det:
I filen `/etc/apache2/apache2.conf`:

```
Timeout 300
```
Apache må nå restartes
```
service apache2 restart
```
og menyen vil dukke opp i Foreman sine nettsider.

D Prosjektplan

Prosjektplan

OpenStack testing

Forfattere:

Terje Pedersen	12HBDRA
Harald Sletten	12HBDRA
Kevin Abadi Barhaugen	12HBDRA

22. Januar 2015

Innhold

1. MÅL OG RAMMER

1.1. Bakgrunn

Enkelte bedrifter har i dag et behov for å drive sikkerhetstrening på et eget lukket nett med ulike scenarioer, der man ikke risikerer å forårsake nedetid på systemer som allerede er i produksjon. Et eksempel på dette er Forsvarets Ingeniørhøgskole/Cyberforsvaret. De trenger realistiske omgivelser der de kan øve og trene på cyber defence-operasjoner. Til dette behøves et skalerbart system der man, hvis ønskelig, kan bruke egen hardware for å sette opp løsningen raskt og automatisk.

Det er her OpenStack kommer inn i bildet. OpenStack er en åpen kildekode skyløsning som brukes for å lage en privat eller offentlig virtuell serverpark. Løsningen samler all CPU, lagring og nettverksressurser fra ulike maskiner i et nettverk og gjør alt dette tilgjengelig for administrasjon. OpenStack er et av de raskest voksende open-source prosjektene i verden og det er også blitt tatt i bruk ved HIG der det har fått navnet SkyHIGh, hvor det brukes til undervisning og forskning.

Det ble våren 2014 levert en suksessfull Bacheloroppgave ved navn AutoStack[1] som handlet om automatisk installering av OpenStack, samt automatisk utrulling av et OpenStack-miljø. Det er ønskelig at man bygger videre på denne oppgaven. Dette medfører at AutoStack blir tilpasset nyeste versjon av OpenStack (Juno), samt innføring av et test-rammeverk for å verifisere at den automatiske installasjonen/utrullingen ble utført korrekt.

1.2. Prosjektmål

Effektmål:

- Ved hjelp av test-rammeverket, gjøre feilsøking enklere og mindre tidskrevende ved utrulling av selve skyløsningen og nye OpenStack-miljøer.
- Ha et tilbud til bedrifter/skoler som ønsker å sette opp virtuelle miljøer for å drive sikkerhetstrening, eller bruke det til andre formål.
- Oppnå bedre sikkerhet[2] og funksjonalitet[3] med nyeste versjon av OpenStack.
- Vil være et kostnadseffektivt alternativ for bedrifter siden man kan benytte egen hardware og løsninger basert på åpen kildekode.

Resultatmål:

- Tilpasse og oppdatere AutoStack til de nyeste versjonene av OpenStack, IceHouse og Juno.
- Lage et test-rammeverk for kvalitetssikring av utrullingen til skyløsningen og scenarioer.
- Sjekke at AutoStack fungerer på to ulike hardware sammensetninger.
- Etter at de øvrige punkt er en realitet ønsker vi å automatisk rulle ut to “cyber range “ scenarioer i samarbeid med gruppen “Digitalt Øvingsfelt”.

Læringsmål:

- Mer komfortable med oppsett av fysiske servere i stedet for virtuelle servere.
- Bli mer kjent med OpenStack.
- Bli bedre med Puppet, scripting.
- Øving i å planlegge et prosjekt over lengre tid. Erfaring i å se sammenhengen mellom planlegging og faktisk utførelse.

2. OMFANG

2.1. Fagområde

Vi som skal jobbe med dette prosjektet går alle sammen linjen 'bachelor i drift av nettverk- og datasystemer'. Dette gjør at den faglige bakgrunnen er forholdsvis lik hos alle tre, og vi kjenner derfor hverandres faglige styrker og svakheter. Utenom dette så har Terje fagbrev i IKT-servicefag, Kevin har tatt IT-fag, og Harald gått folkehøgskole med fokus på data, noe som også vil komme til nytte.

Oppgaven passer veldig godt til linja vår fordi den omhandler fysisk interaksjon med servere, bruk av skripting og koding på flere forskjellige nivåer, og tar utgangspunkt i flere av de fagene vi har hatt, slik som Operativsystemer, Systemutvikling, og Database- og applikasjonsdrift. Nettverk vil det også være en del av, og der kommer faget Nettverksadministrasjon godt til nytte.

2.2. Avgrensning

Når det gjelder utstyr som skal brukes, holder vi oss kun til de maskinene vi får låne av skolen. Disse maskinene befinner seg på rom K202, og blir derfor av praktiske årsaker vår arbeidsplass. Av eksterne IP-addresser kommer vi til å få utdelt noen få for bruk i prosjektet, frem til da så holder vi oss til DHCP.

Prosjektet er definert som å oppgradere AutoStack, og dermed er det kun denne løsningen vi skal gå ut ifra. Da det er sagt, så kommer løsningen til å bli redesignet, og det ferdige produktet kan bli ganske så forskjellig fra det vi begynte med, men funksjonen forblir den samme. Vi kommer ikke til å lage fullverdige scenarioer som skal rulles ut i skyløsningen. Dette er det en annen bachelorgruppe som har i oppgave å realisere.

2.3. Oppgavebeskrivelse

Oppgaven omhandler å bygge videre på bacheloroppgaven 'AutoStack' fra våren 2014, og den kan deles inn i tre deler.

I den første delen skal vi oppdatere AutoStack fra dens nåværende OpenStack-versjon Grizzly frem til Icehouse. Det er et sprang på to versjoner, og de fleste endringene blir i puppet-modulene for å få det til å fungere. I tillegg så har vi blitt bedt om å bytte ut PXE bootserveren Cobbler og distribueringssystemet vårt Puppet med den komplette og mer oversiktige løsningen Foreman. Dette vil gjøre AutoStack både lettere å bruke og lettere å teste for oss senere i oppgaven.

Etterpå følger videre oppdatering av AutoStack. Denne gangen oppdaterer vi den videre til den aller nyeste versjonen av OpenStack, Juno. I denne versjonen kom det blant annet støtte for IPv6, og en rekke ting har endret seg i konfigurasjonsfiler. Her er det dog et lite problem; siden vi bruker puppet-moduler som vår hovedkonfigurasjonsmetode, så er vi avhengig av at disse er oppdatert. Disse er det Puppetlabs som er ansvarlige for. I skrivende stund (16.1.2015) så er de

ikke det, men vi håper de kommer ut tidlig nok. Hvis ikke så må vi fullføre oppgaven vår med Icehouse-versjonen av OpenStack.

Etter vi har oppdatert AutoStack så skal vi teste ut AutoStack på to forskjellige hardwaresammensetninger. Den ene består av kun HP proliant 380 G5-servere, den andre er ikke spesifisert av oppdragsgiver enda. Dette er for å sikre stabilitet på en type hardware, og forsikre seg om at det også fungerer på en annen type hardwaresammensetning.

I del to så skal vi lage et test-rammeverk for skyløsningen AutoStack. Her skal det forsikres at alle noder og tjenester er oppe og kjører etter utrulling, og det skal rapporteres tilbake til den som har satt opp systemet gjennom Foreman. Puppet og OpenStack sin manager, Nova, har allerede mange innebygde funksjoner for testing. Planen er derfor å utnytte disse så mye som mulig før vi skriver våre egne metoder.

Del tre består av å rulle ut to scenarioer og skrive et test-rammeverk for disse. Et scenario består av en rekke maskiner, rutere og programvare som skal rulles ut virtuelt i OpenStack-løsningen. Her samarbeider vi med en annen gruppe, "Digitalt øvingsfelt", hvor de er hovedansvarlig for å finne opp disse scenarioene. Vår oppgave blir å implementere de inn i OpenStack-utrullingen, og sjekke at de fungerer som de skal.

3. PROSJEKTORGANISERING

3.1. Ansvarsforhold og roller

Innad i gruppa har vi valgt Terje Pedersen til å være vår prosjektleder, og vil også derfor ha ansvaret for å være kontaktperson mot både oppdragsgiver og veileder, for eksempel angående avtalte møter osv. Arbeidsfordelingen i gruppen er bestemt slik at oppgaver fordeles fortløpende mellom alle studentene i gruppen slik at samtlige får erfaring med systemet. Oppgavene som blir fordelt mellom studentene skal være så jevnt fordelt som mulig, slik at alle gjør omtrent like mye. For denne bacheloroppgaven "OpenStack Testing", er det Erik Hjelmås som er oppdragsgiver. Vår tilegnede veileder for denne oppgaven er Eigil Obrestad.

3.2. Rutiner og regler i gruppa

- Alle studentene i gruppen skal respektere tidsfrister og det å møte tidsnok til bestemte avtaler og møter. Dersom noen blir syke eller ikke kan komme til avtalt tid, skal man si i fra så tidlig som mulig.

- Gruppen som helhet har et mål om å etterstrebe 30 timers arbeid med oppgaven per uke totalt. Disse timene vil helst brukes i hverdagene, slik at man kan ta en velfortjent pause på helgedager. Vi kan selvfølgelig arbeide i helgene hvis det blir nødvendig, men vi skal prøve å unngå dette så mye som mulig.

- Dersom det skulle oppstå uenigheter som ikke kan løses innad i gruppen, skal dette tas opp med veileder/oppdragsgiver.

Når det gjelder arbeidsmetoder så kommer vi til å samarbeide nesten hele tiden. Vi har planlagt en rutine på at vi møtes nesten hver ukedag for å jobbe med bacheloroppgaven. Det kommer til å bli noe individuelt arbeid, men det blir mest sannsynlig under arbeid med prosjektplan, og endelig rapportskriving på slutten av semesteret.

4. PLANLEGGING, OPPFØLGING OG RAPPORTERING

4.1. Hovedinndeling av prosjektet

Karakteristika med oppgaven

- Vi er tre stykker som skal jobbe med oppgaven. Vi har alle litt erfaring med OpenStack siden vi i høst arbeidet sammen på et lite prosjekt som handlet om AutoStack. Denne erfaringen tror vi blir veldig verdifull i hvert fall i oppstartfasen siden det går med mye tid på å sette seg inn i systemet, og få satt det opp de første gangene.

- Bacheloroppgaven skal leveres 15. mai så vi har totalt ca. fem måneder på å få til løsningen samt skrive rapport.

- Måten vi kommer til å jobbe på blir å sette oss små delmål hele tiden slik at vi legger stein på stein (inkrementelt) og til slutt ender med det ferdige resultatet.

- Endringer vil sannsynligvis komme hele tiden gjennom oppgavens gang. Dette på grunn av at det er problemer vi ikke vet hvordan skal løses, slik at vi hele tiden må tilpasse oss etter vår progresjon og måten vi løser problemene på.

- Vanskelig å planlegge framover i en slik oppgave siden mye er nytt og usikkert. Risikoen ved prosjektet er at mye er basert på OpenStack modulene til PuppetLabs, og de har, per dags dato, ikke kommet med noen ny versjon til Juno. Vi vet heller ikke hvordan vi skal lage test-rammeverket.

- Vi er avhengige av å kunne prøve og feile for å kunne løse de problemene vi vil møte på.

- Dokumentasjon vil være veldig viktig i et slikt prosjekt. Det er helt avgjørende at vi dokumenterer alt vi gjør underveis slik at dette ikke blir glemt eller forårsaker en hel masse unødvendig jobb i slutten av prosjektet, der man må søke fram og sette sammen den nødvendige informasjonen.

Valg av systemutviklingsmodell

Når vi skulle diskutere utviklingsmodell var vi ganske samstemt på at det er riktig å gå for en agil utviklingsmetode. I agile prosesser så er planleggingen inkrementell og det er lettere å endre prosesser etter kundens ønsker. Vi har valgt å bruke en blanding mellom inkrementell og Scrum som systemutviklingsmodell.

Her er hvorfor vi mener dette er et riktig valg:

- Egner seg for små utviklingsgrupper.
- Vi kommer til å måtte gjøre endringer ofte. Vår modell legger opp til dette, og vil derfor gjøre det mindre kostbart å gjøre endringer.
- Lettere for oss å bedømme kjørende kode, kontra design dokumenter.
- Vi kan begynne å teste deler av løsningen fortere enn i en plan-drevet utviklingsmodell.
- Vi ser for oss at vi kommer til å jobbe i «sprinter», og sette sammen del for del.
- Alle vil være med på å bestemme videre progresjon.
- Modellen legger opp til at arbeidet dokumenteres underveis.

Anvendelse av systemutviklingsmodell

Sprint cycles

- Vi ser for oss at vi kommer til å jobbe i små sykluser der vi prøver å implementere del for del av systemet hvor vi på forhånd planlegger hva vi ønsker å oppnå(backlog). Disse kommer til å bli en del kortere enn de man vanligvis kjører i Scrum og vil heller ikke ha noe fast tidsintervall. Det vil dog være en grense på hvor lang sprintene kan være uten tilstrekkelig progresjon. Her ser vi for oss at to uker er maksimal tidsbruk .

Sprint retrospective

- Neste «sprint» vil bli planlagt i forhold til hvordan den forrige gikk, og vi kommer til å ha daglige møter der vi diskuterer status siden vi kommer til å sitte i det samme rommet og arbeide mesteparten av tiden.

Inkrement

- Siden vi bygger løsningen inkrementelt kommer vi til å utføre testing etter hver iterasjon for å se at det vi har gjort fungerer på en tilfredsstillende måte. Her kan man kjapt finne bugs fordi det er blitt gjort få endringer siden siste iterasjon.

Kravspesifikasjon og dokumentasjon

- Vi kan håndtere endringer som skulle forekomme underveis i arbeidet på en god måte siden vi ikke trenger å dokumentere og spesifisere alt før vi begynner å kode. Fullstendig kravspesifikasjon og dokumentasjon blir laget underveis.

4.2. Plan for statusmøter og beslutningspunkter

Nå i planleggingsfasen møtes vi et par ganger i uken der vi blir enige om hva som må gjøres, og fordeling av arbeid. Vi setter oss deretter ned på et avtalt tidspunkt, der vi setter sammen det vi har gjort hver for oss. Når vi så begynner på oppgaven kommer vi for det meste til å sitte sammen og jobbe. Vi kommer derfor til å ha små møter først på dagen om hva som er gjort, og hva som skal gjøres. Det vil også være naturlig at vi har en kontinuerlig dialog gjennom dagen om oppgavens progresjon. I tillegg til dette har vi avtale med arbeidsgiver og veileder om å samles til et møte hver mandag.

5. ORGANISERING AV KVALITETSSIKRING

5.1. Dokumentasjon, standardbruk og kildekode

I gjennomføringen av bacheloroppgaven skal alt dokumenteres; linker, endringer i kode/konfigurasjon, og loggføring for hver arbeidsdag. Kildekoden skal være ryddig med innrykk og gode/forklarende kommentarer, slik at koden er mulig å forstå både for de andre studentene, for oppdragsgiver, og eventuelt for fremtidig bruk av andre. Gjennom arbeidet kommer vi til å gi kommentarer til hverandres kildekode, og vil foreslå forbedringer der det er nødvendig. Gruppen bruker Google Drive og Google Docs som standardverktøy for å dele og skrive dokumenter/konfigurasjonsfiler/skript seg i mellom. For å lage modeller og figurer brukes Microsoft Visio. Vi kommer også til å bruke Latex ved innlevering av rapporten. Vi mener at et konfigurasjonsstyringsverktøy ikke er nødvendig i måten vi skal arbeide på. Vi kommer stort sett til å arbeide med separate filer. Når vi en sjelden gang skal jobbe med samme filer, kommer vi mest sannsynlig til å jobbe i samme rom.

5.3. Risikoanalyse (identifisere, analysere, tiltak, oppfølging)

Prosjektrisiko:

1. Release av Juno moduler fra Puppetlabs tidsnok.
2. Redesign av AutoStack kan bli en for stor oppgave.
3. Cyber-range scenarioer blir ikke ferdige tidsnok.
4. Hardwarefeil på utstyr.
5. Sykdom.
6. Mangelfull dokumentasjon.
7. Tidsfrister blir ikke overholdt.
8. Design av test-rammeverk

Av punktene nedenfor har vi valgt å se nærmere på punktene 1, 2, 4, og 8, siden vi mener disse er de største/viktigste risikoene i bacheloroppgaven. Tabellen nedenfor viser hvor store konsekvenser, og hvor sannsynlig det er at prosjektrisikoene som er nevnt over inntreffer:

	Lav	Middels	Høy	Katastrofal
Svært sannsynlig	_	_	_	_
Meget sannsynlig	_	2	1	_
Sannsynlig	3	5,6	7,8,4	_
Lite sannsynlig	_	_	_	_

Figur 1 - Risikotabell

Plan for håndtering av de viktigste risikoene

Analyse av Juno moduler fra Puppetlabs:

I bacheloroppgaven skal vi oppdatere OpenStack til nyeste versjon; OpenStack Juno. Vanligvis pleier release av OpenStack moduler fra Puppetlabs å komme ut rundt Desember måned. I skrivende stund (16.01) er disse modulene fortsatt ikke ute, men kan komme ut i de senere påfølgende uker.

Planlegging av Juno moduler fra Puppetlabs:

Vi har bestemt oss for at hvis ikke modulene kommer tidsnok, må vi bruke nest nyeste versjon OpenStack IceHouse. Vi kommer ikke til å prøve å lage disse modulene selv, siden dette blir for tidkrevende å gjøre. Vi har konkludert med dette på bakgrunn av en diskusjon som ble gjort mellom oss og oppdragsgiver.

Analyse av redesign av AutoStack:

Hvis det viser seg at redesign av AutoStack kreves basert på utvidelsen av Puppet-implementasjonen, så kan dette bli en stor oppgave som krever tid. Redesign av AutoStack innebærer å flytte mer kode fra eksisterende skript til site.pp. Vi har også fått beskjed fra oppdragsgiver om å bruke Foreman istedet for Cobbler som initielt hadde vært brukt.

Planlegging av redesign av AutoStack:
Tiltakene vi kan gjøre er å bruke helgedagene til å arbeide med oppgaven i tillegg til hverdager. Hvis dette fortsatt ikke viser seg å være nok, må vi prioritere de viktigste tingene som må endres på.

Analyse av hardwarefeil på utstyr:
Hardwarefeil på utstyr som skal brukes kan være et stort problem hvis det ikke oppdages tidlig. I en prosjektoppgave i høst hadde vi nettopp et slikt problem, noe som førte til større tidsbruk enn det vi forventet.

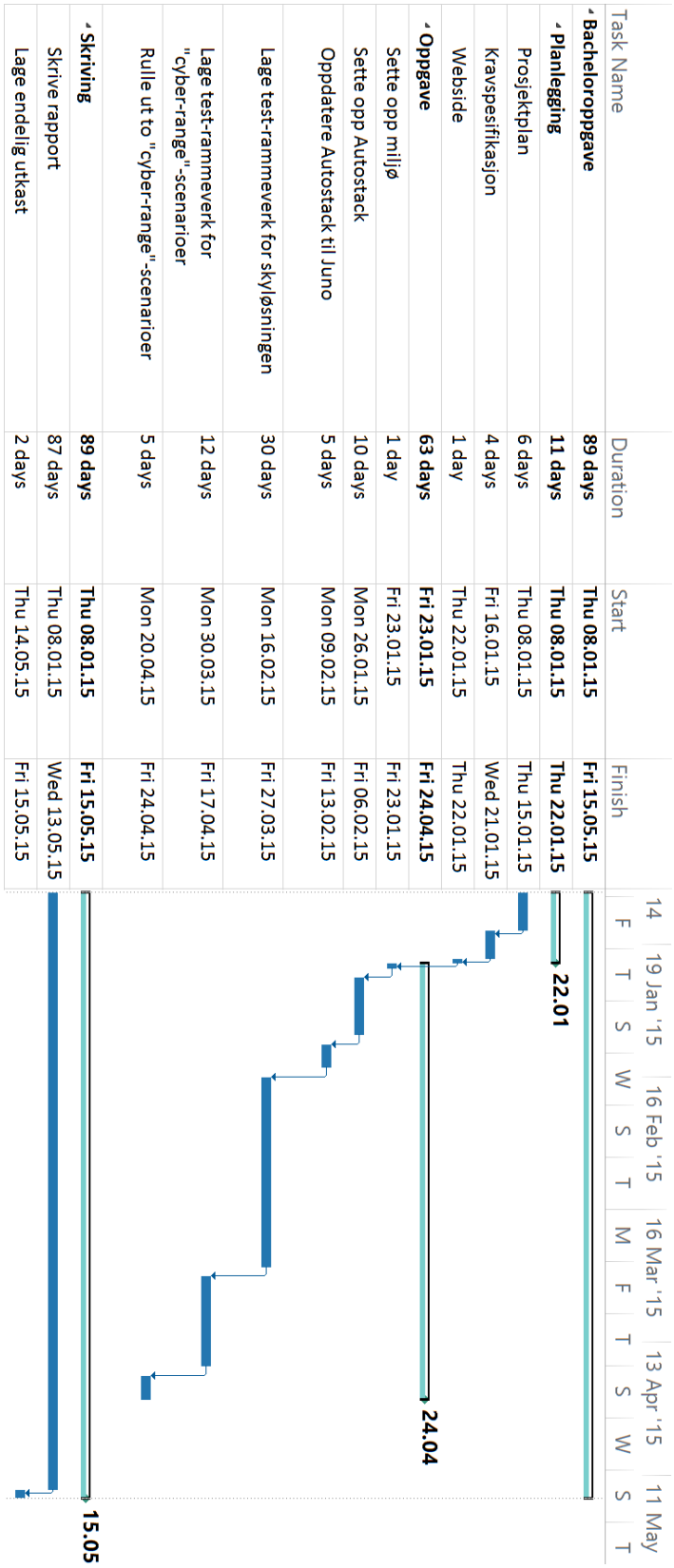
Planlegging av hardwarefeil på utstyr:
Vi vil da sjekke tidlig om maskinene vi har fått tildelt, fungerer slik de skal. Hvis de ikke gjør det må vi spørre oppdragsgiver om å få tildelt en annen maskin som er ledig, eventuelt få tildelt nye hardware-deler som kan byttes ut.

Analyse av design av test-rammeverk:
Realiseringen av Puppets test-rammeverk for å teste AutoStack kan ta en god del tid. Hvis det viser seg at Puppets test-rammeverk ikke kan brukes for å teste vårt system, må vi i verste fall skrive et nytt test-rammeverk, noe som vil ta enda lenger tid. Ikke minst er det også teknisk vanskelig å lage dette fra bunnen av.

Planlegging av design av test-rammeverk:
De eneste tiltakene vi har tilgjengelig mot dette er de samme tiltakene som i redesign av AutoStack. Vi må eventuelt bruke ekstra tid i helgene, og hvis det ikke er nok, må vi her også prioritere de viktigste tingene som bør testes, og implementere dette.

6. PLAN FOR GJENNOMFØRING

Figur 2 - Ganttskjema



Task Name	Duration	Start	Finish
Bacheloroppgave	**89 days**	**Thu 08.01.15**	**Fri 15.05.15**
Planlegging	**11 days**	**Thu 08.01.15**	**Thu 22.01.15**
Prosjektplan	6 days	Thu 08.01.15	Thu 15.01.15
Kravspesifikasjon	4 days	Fri 16.01.15	Wed 21.01.15
Webside	1 day	Thu 22.01.15	Thu 22.01.15
Oppgave	**63 days**	**Fri 23.01.15**	**Fri 24.04.15**
Sette opp miljø	1 day	Fri 23.01.15	Fri 23.01.15
Sette opp Autostack	10 days	Mon 26.01.15	Fri 06.02.15
Oppdatere Autostack til Juno	5 days	Mon 09.02.15	Fri 13.02.15
Lage test-rammeverk for skyløsningen	30 days	Mon 16.02.15	Fri 27.03.15
Lage test-rammeverk for "cyber-range" -scenarioer	12 days	Mon 30.03.15	Fri 17.04.15
Rulle ut to "cyber-range"-scenarioer	5 days	Mon 20.04.15	Fri 24.04.15
Skriving	**89 days**	**Thu 08.01.15**	**Fri 15.05.15**
Skrive rapport	87 days	Thu 08.01.15	Wed 13.05.15
Lage endelig utkast	2 days	Thu 14.05.15	Fri 15.05.15

Forklaring av Ganttskjema:
Planen vår er delt opp i tre deler: planlegging, oppgaveløsning og skriving. Vi bestemte oss for å komme fort i gang med prosjektplanen, slik at vi fikk så mye tid som mulig på løsing av selve oppgaven.

Når det gjelder oppgaveløsningen, ble det en utfordring å fordele tid på de forskjellige oppgavene. Det er vanskelig i planleggingsfasen å avgjøre hvor omfattende hver oppgave kan bli, men vi har etter beste evne forsøkt å fordele mest tid på de oppgavene vi er mest usikre på, og mindre tid på de vi er mer sikre på. Derfor er det test-rammeverket som vi har bestemt å bruke mest tid på, siden dette er det største usikkerhetsmomentet.

Til slutt kommer rapportskrivingen. Denne går parallelt med hele prosjektet, hvor planen er å bruke minst en halv time på slutten hver dag med å dokumentere det vi har gjort. Siden vi vil ha fokus på å dokumentere underveis, vil vi spare tid når det kommer til selve rapportskrivingen. Vi har derfor kun satt av tre uker til ren skriving av rapport.

KILDER

[1] AutoStack - Automatisk og adaptiv utrulling av Openstack
http://brage.bibsys.no/xmlui/handle/11250/216816

[2] OpenStack - Security/Juno
https://wiki.OpenStack.org/wiki/Security/Juno

[3] OpenStack - New Features in Juno
http://www.OpenStack.org/software/juno/

E E-post

Ser at vi har hakket nyere versjoner jevnt over. Innbiller meg at devstack
drar ned fra GIT master branch når man kjører opp. Vi har:
Nova: 2.20.0
Heat: 0.3.0
Neutron: 2.3.10
Openstack: 1.0.2 (Usikker på hvor jeg finner dato-versjon)

Har lagt merke til at noen properties i heat er deprecated (Usikker på om
vi har brukt noen av de) og at noen er bare tilgjengelige i versjon x og
oppover (usikker på hvem av disse vi kan ha brukt også). Her er en oversikt
over det meste: http://docs.openstack.org/developer/heat/template_guide/openstack.html

Skal se litt nærmere på de spesifikke errorene dere fikk nå snart.

- Andreas og Mats

27. april 2015 kl. 10.05 skrev <harald.sletten@fjellnett.no>:

> Nova: 2.19.0
> Heat: 0.2.10
> Neutron: 2.3.8
>
> Openstack: 2014.2.2
>
> Hvis versjonene ikke stemmer, ta kontakt så vi kan prøve å oppdatere til
> nyere. Dersom dere ikke finner noe, kan det være lettere å feilsøke i
> samme rom som oss. Vi sitter på K202.
>
> Takk,
> Openstack testing
>
> > Hei igjen,
> >
> > Dette høres ikke ut som kjente feilmeldinger for vår del. Kan du sende
> > over
> > litt versjonsinformasjon (openstack, heat, nova osv). Skal ta en titt på
> > det i løpet av dagen og se om jeg kan finne ut hva som kan lage disse

> error
> meldingene.
>
> - Mats og Andreas
>
> 27. april 2015 kl. 09.24 skrev <harald.sletten@fjellnett.no>:
>
>> Hei.
>>
>> Da fikk vi til pluginen extraroute (Det var den samme feilen som dere
>> hadde, .openstack skulle ikke være med), men nå sitter vi fast på et
>> annet
>> område. Hver gang vi prøver å lage en stack så får vi bare enten
>> 'Unknown
>> Property Port' eller 'Property range-name not assigned'. Har dere også
>> fått disse feilene, eller er det spesielt hos oss? Dette skjer på
>> template-valideringsnivå, så det er ingen andre feilmeldinger i loggene,
>> og Google har ingen svar.
>>
>> Ser frem til svar fra dere,
>>
>> vennlig hilsen Openstack-testing.
>>
>> > Hey igjen,
>> >
>> > Beklager litt sent svar. Dere har kanskje funnet ut av dette nå?
>> > ExtraRoute
>> > kom ikke med default installasjonen. Vi fant ExtraRoute filen(e) i
>> Heat
>> > sin
>> > master branch på github:
>> > https://github.com/openstack/heat/tree/master/contrib/extraroute
>> >
>> > Installasjonen er enkel nok, men ikke helt intuitiv kanskje..
>> > Filen extraroute.py i
>> >
>> https://github.com/openstack/heat/tree/master/contrib/extraroute/extraroute/resources
>> > plaseres i /opt/stack/heat/heat/engine/resources/neutron eller hvor nå
>> enn
>> > dere har installert heat. Deretter restart neutron.
>> >
>> > Mulig ekstra feil:
>> > Hvis dere får error når dere prøver dette kan det hende at dere har
>> samme
>> > feilen vi hadde i extraroute.py. Endre "from
>> > heat.engine.resources.openstack.neutron import neutron" til "from

>> > heat.engine.resources.neutron import neutron" i en av de øverste
>> linjene.
>> >
>> > Skulle dere ikke få det til får dere bare sende oss en melding om det
>> så
>> > kan vi sikkert bistå litt :)
>> >
>> >
>> > Mvh
>> > Andreas og Mats
>> > Digitalt Øvningsfelt
>> >
>> > 15. april 2015 kl. 08.38 skrev <harald.sletten@fjellnett.no>:
>> >
>> >> Hei igjen,
>> >>
>> >> Vi fikk kopiert over templates og byttet ut ID til public nett, men
>> vi
>> >> mangler en komponent kalt 'extraroute'. Hvordan har dere installert
>> den?
>> >> Kom den gjennom default Juno installasjon eller installerte dere den
>> >> selv?
>> >> Vi prøvde å installere pluginen som best vi kunne, man har ikke
>> lykkes
>> >> med
>> >> dette enda.
>> >>
>> >> Vennlig hilsen
>> >>
>> >> Autostack-testing
>> >>
>> >> > På tegningen betyr M "Main router", MA "Management router", AT
>> "Attack
>> >> > router", AD "Admin router", RS "Ranges router", RN "Range router"
>> >> >
>> >> > 14. april 2015 kl. 13.59 skrev Andreas Ringstad
>> >> > <andreas.ring92@gmail.com>:
>> >> >
>> >> >> Hei igjen,
>> >> >>
>> >> >> Da har vi fått til ett grunnscenario. Legger ved nettverksbilde og
>> >> zip
>> >> >> av
>> >> >> templates. "core.yaml" er den som skal lastes opp/kjøres. De andre
>> >> >> inkluderes dynamisk fra URL. Dere kan bruke de URLene som er der
>> >> eller

>> >> >> laste de opp selv et sted.
>> >> >>
>> >> >> Er litt dårlig kommentert/dokumentert. Men vi kan sende dere
>> >> >> oppdateringer
>> >> >> etterhvert som vi nærmer oss ett litt finpusset produkt.
>> >> >>
>> >> >> Med vennlig hilsen
>> >> >> Andreas og Mats
>> >> >> Digitalt Øvningsfelt
>> >> >>
>> >> >> 14. april 2015 kl. 11.13 skrev <harald.sletten@fjellnett.no>:
>> >> >>
>> >> >>> Så flott :)
>> >> >>>
>> >> >>> > Hey igjen,
>> >> >>> >
>> >> >>> > Vi er nesten ferdig med ett grunnscenario, en type hello world
>> av
>> >> >>> > scenarioer. Det eneste som mangler er litt routing, og det blir
>> vi
>> >> >>> nok
>> >> >>> > ferdig med i dag. Kan sende dere det i løpet av dagen. Vi har
>> enn
>> >> så
>> >> >>> lenge
>> >> >>> > ikke begynnt på noen andre scenarier, da vi vil bli ferdig med
>> et
>> >> >>> > grunnoppsett først. Skal se om vi ikke kan få ordnet ett
>> >> >>> nettverkskart
>> >> >>> til
>> >> >>> > dere også, da det er ganske vanskelig å få oversikt uten det.
>> >> >>> >
>> >> >>> > Mvh
>> >> >>> > Andreas og Mats
>> >> >>> > Digitalt Øvingsfelt
>> >> >>> >
>> >> >>> > 14. april 2015 kl. 09.50 skrev <harald.sletten@fjellnett.no>:
>> >> >>> >
>> >> >>> >> Hei!
>> >> >>> >>
>> >> >>> >> Oppgaven vår sier vi skal skrive et testrammeverk for minst 2
>> >> >>> >> cyber-range
>> >> >>> >> scenarioer, og med den lille tiden vi har igjen så er det
>> greit å
>> >> >>> >> begynne
>> >> >>> >> så tidlig som mulig. Derfor hadde det vært fint om dere kunne

>> >> sende
>> >> >>> oss
>> >> >>> >> templates for 2 scenarioer, uavhengig om den er ferdig eller
>> >> ikke,
>> >> >>> slik
>> >> >>> >> at
>> >> >>> >> vi får en fin oversikt og kan begynne å programmere. De enkle
>> >> >>> templatene
>> >> >>> >> vi fikk i går fungerte helt fint, og verifiserte at Heat
>> fungerte
>> >> >>> hos
>> >> >>> >> oss.
>> >> >>> >>
>> >> >>> >> Vennlig hilsen
>> >> >>> >>
>> >> >>> >> Autostack Testing
>> >> >>> >>
>> >> >>> >>
>> >> >>> >> > Ok!
>> >> >>> >> >
>> >> >>> >> > Vi ser om vi får disse til å fungere (vi må kanskje bytte ut
>> >> noen
>> >> >>> >> IDer)
>> >> >>> >> > med vår HEAT-installasjon og så mailes vi senere når vi er
>> klar
>> >> >>> til å
>> >> >>> >> > teste big time. :)
>> >> >>> >> >
>> >> >>> >> >> Ooooops! Der var det ingen filer. Legger ved her ;)
>> >> >>> >> >>
>> >> >>> >> >> PS: Er ikke helt enkelt å bare skjønne åssen dette funker,
>> så
>> >> >>> hvis
>> >> >>> >> dere
>> >> >>> >> >> står fast og trenger litt forklaring får dere si ifra så
>> kan
>> >> vi
>> >> >>> se
>> >> >>> >> hva
>> >> >>> >> >> vi
>> >> >>> >> >> får gjort ;)
>> >> >>> >> >>
>> >> >>> >> >> 13. april 2015 kl. 11.33 skrev Andreas Ringstad
>> >> >>> >> >> <andreas.ring92@gmail.com>:
>> >> >>> >> >>
>> >> >>> >> >>> Hei!

>> >> >>> >> >>>
>> >> >>> >> >>> Vi har ikke så veldig mange enkle templates etter lang tid
>> >> med
>> >> >>> >> prøving,
>> >> >>> >> >>> feiling, testing og videreutvikling. Men fant et par som
>> >> kanskje
>> >> >>> er
>> >> >>> >> av
>> >> >>> >> >>> interesse. Legger ved disse.
>> >> >>> >> >>>
>> >> >>> >> >>> Vi driver å skriver litt mer avanserte og større templates
>> >> for
>> >> >>> div.
>> >> >>> >> >>> scenarioer og grunnoppsett nå. Disse er kanskje av
>> interesse
>> >> å
>> >> >>> se
>> >> >>> på
>> >> >>> >> >>> når
>> >> >>> >> >>> de
>> >> >>> >> >>> nærmer seg ferdige? Kunne vært spennende for oss å
>> prøvekjøre
>> >> de
>> >> >>> på
>> >> >>> >> >>> deres
>> >> >>> >> >>> oppsett også.
>> >> >>> >> >>>
>> >> >>> >> >>>
>> >> >>> >> >>> Med vennlig hilsen
>> >> >>> >> >>> Andreas og Mats
>> >> >>> >> >>> Digitalt øvingsfelt
>> >> >>> >> >>>
>> >> >>> >> >>> 13. april 2015 kl. 08.26 skrev
>> <harald.sletten@fjellnett.no
>> >:
>> >> >>> >> >>>
>> >> >>> >> >>>> Hei digitalt øvingsfelt!
>> >> >>> >> >>>>
>> >> >>> >> >>>> Vi er ferdig med å sette opp OpenStack Juno og er nå
>> langt
>> >> inne
>> >> >>> i
>> >> >>> >> >>>> testingfasen, som betyr at vi trenger noen heat-templates
>> >> for å
>> >> >>> teste.
>> >> >>> >> >>>> I

>> >> >>> >> >>>> tillegg til å teste heat skal vi også skrive et
>> >> test-rammeverk
>> >> >>> for
>> >> >>> >> >>>> templaten, så det hadde vært fint å få forklart litt hva
>> som
>> >> er
>> >> >>> >> >>>> bakgrunnen
>> >> >>> >> >>>> og meningen med templaten. Har dere ett klart for oss? Vi
>> >> >>> sitter
>> >> >>> på
>> >> >>> >> >>>> K202.
>> >> >>> >> >>>>
>> >> >>> >> >>>> Vennlig hilsen
>> >> >>> >> >>>>
>> >> >>> >> >>>> OpenStack Testing, mvh. Harald Sletten

F Statusrapporter

Statusrapport 20.2 Openstack Testing

Vi føler vi har kommet godt i gang med bacheloroppgaven. Vi begynte tidlig med planleggingsfasen slik at vi tidlig kunne begynne å jobbe med det praktiske. Per i dag har vi fått erstattet Cobbler med Foreman og kommet i mål med automatisk installering av AutoStack med IceHouse (oppgradering fra Havana) og modulen Heat som en ekstra komponent. Med tilgang på noen ekstra servere kan derfor gruppen Digitalt Øvingsfelt nå begynne å teste sine templates på vårt system.

Vi har brukt en del tid på feilsøking av neutron, siden det her skjedde et plugin skifte fra Havana til IceHouse. Det har også gått med litt ekstra tid på oppsett av DNS og DHCP i Foreman siden dette var nytt for oss og naturlig nok ikke beskrevet i AutoStack oppgaven fra i fjor.

I forhold til tiddsskjema ligger vi godt ann. Det har gått med litt mere tid på oppsettet av AutoStack enn det vi forutså, så vi ligger litt over en uke etter skjema, men dette stresser oss ikke nevneverdig.

Arbeidsmessig har vi kommet inn i en god flyt og vi jobber ca 30 timer med oppgaven hver uke. Samarbeidet oss i mellom går bra og det er lite uenigheter. Vi prøver hele tiden å dokumentere så mye som mulig underveis, slik at alle endringer blir ført opp som en slags hurtig manual for oss selv.

Neste steg for oss nå blir å oppdatere AutoStack til Juno, for deretter å begynne med test-rammeverket.

Statusrapport 20.03.15 - Openstack Testing

Siden forrige statusrapport har fokuset vårt vært på å få oppdatert AutoStack til å kjøre Juno. Vi begynte å se på hva som måtte gjøres for å få oppdatert fra IceHouse til Juno, og gjorde dette manuelt først for å se om det fungerte. Etter at vi fikk dette til å fungere tømte vi site.pp, for deretter å sette inn alt på nytt fra scratch. Vi tok for oss hver vår node og implementerte alt som krevdes for at nodene skulle bli satt opp riktig. Samtidig tok vi også å gjorde om alle bash scriptene til puppet kode slik at all koden vår nå er i en fil, site.pp. Dette var en tidskrevende prosess som vi brukte litt over en uke med arbeid på. Etter møte med arbeidsgiver og veileder den 9.mars ble vi oppmerksomme på at OpenStack-puppet-modulene kanskje ikke var helt oppdatert, siden vi meldte om at vi måtte gjøre flere endringer i modulene for å få de til å fungere med den offisielle OpenStack guiden. Det vi fant ut var at modulene som ligger ute på Puppet Forge (https://forge.puppetlabs.com), de er rett og slett utdaterte, og har ikke blitt oppdatert selv om modulene har blitt oppdatert på prosjektenes github sider (https://github.com/stackforge). Etter at vi hentet de oppdaterte filene fra hvert prosjekt, så fant vi ut at flere av de feilene/manglene som vi hadde oppdaget, var blitt fikset. Til slutt har vi da fått til automatisk utrulling av Juno uten noen bash script eller endring av kode i moduler.

Videre har vi brukt en god del tid på å få både VLAN og GRE nettverk til å fungere i Juno. Vi har særlig slitt med å få kontakt med instansene etter at de har startet opp. Dette skyldes ting vi har oversett i OpenStack guiden eller at overgangen fra script til puppet kode ikke har vært 100% vellykket.

Ellers har vi god arbeidsflyt og føler vi gjør fremskritt. I forhold til tidsskjemaet ligger vi ca èn måned etter planen, men føler fortsatt at vi har kontroll siden vi har til den 17. april til å sette opp test-rammeverket i følge vårt Gannt skjema, og det er heller ikke spesifisert hvor stor dette rammeverket skal være.

Så for å oppsummere: Automatisk utrulling av Juno med både VLAN og GRE er på plass. Veien videre blir å se på test-rammeverket.

Andre ting som er blitt gjort:

- Testet og vellykket bruk av HP DL320 G5-server og en Dell Optiplex 745 som compute noder.
- Har prøvd å gjort optimalisering av koden i site.pp, samt sørge for at ting blir så automatisk som mulig.
- Har begynt å se på hvordan test-rammeverket kan settes opp.
- Har også meldt inn en bug til utviklerene av puppet-neutron modulen hvor vi kommer med et forbedringsforslag: https://bugs.launchpad.net/puppet-neutron/+bug/1431192

Statusrapport 24.04.15 - Openstack Testing

Siden forrige statusrapport har vi jobbet med følgende:

Vi har flyttet all koden fra site.pp og lagd en “Autostack” modul slik at man nå inne i Foreman kan velge om en node skal bli en controller, nettverk, eller compute node bare ved å legge til en klasse. Man kan også bruke “Override” funksjonen i Foreman for å endre på variablene i .pp filene, slik at at man kan velge mellom VLAN og GRE som nettverkstyper for eksempel.

Siden automatisk installering av OpenStack mer eller mindre er i boks begynte vi å se på test-rammeverket og hvordan dette kunne implementeres. Vi utforsket et eksisterende verktøy for testing som heter Rally (http://rally.readthedocs.org/en/latest/) og klarte å kjøre en del tester med dette verktøyet. Etter litt testing med Rally kom vi fram til at dette ikke fungerte helt som vi ville ha det. Vi fant ut at Rally nesten er mer et benchmarking-verktøy enn et rent test-verktøy, og vi følte oss litt låst med tanke på rapporteringen fra testen.
Vi bestemte oss derfor for å begynne å scripte et testrammeverk fra bunnen av. Da får vi full kontroll over hva som testes, og rapporteringen som skrives tilbake. Test-rammeverk-scriptet vårt setter opp et lite scenario som tester ut hver OpenStack-komponent og gir svar tilbake i form av en rapportfil. Dette test-scriptet for selve utrullingen av AutoStack er nå så godt som ferdig. Vi har implementert dette test-rammeverket inn i Foreman slik at man kan starte scriptet fra Foreman, og får skrevet ut rapporten fortløpende inne i Foreman.

Ellers har vi begynt å skrive på rapporten.

Det neste for oss blir å samarbeide med Digitalt Øvingsfelt gruppen, der vi skal rulle ut deres scenarier, og skrive et test-rammeverk for disse. Vi har alt begynt å se på et av deres scenarier.

G Dagslogg

Dagslogg

8-9. januar

- Begynte arbeidet med prosjektplanen

12. januar

- Fordelte arbeid med prosjektplanen
- Møte med Erik

16. januar

- Møttes for å sette sammen prosjektplanen. Ble så godt som ferdig.

19. januar

- Mer pussing på prosjektplanen.
- Avklarte arbeidsfordeling på kravspesifikasjonen
- Møte med veileder der vi planla møter fremover

22. januar:

- Endelig ferdig med prosjektplan
- Satt sammen og ferdige med kravspesifikasjonen

23. Januar

- Webside er oppe.
- 14.04 Ubuntu-installering på Nostrand og tatt backup (clonezilla).
- Installert Foreman på nostrand

26. Januar

- Prøvde å få foreman til å erstatte Cobbler. Så langt ingen suksess.
- Møte med Erik og Eigil.

28. Januar

- Fortsatte med deployment via foreman, og har i tillegg satt opp DNS på foreman som det ser ut som at kreves. Gjør små fremskritt.

29. Januar

Strevd veldig med forman i hele dag:

- Først fikk vi ikke tak i nostrand. (Fikset dette med dhclient -r)
- Som et resultat av dette blir /etc/resolv.conf overskrevet, slik at vi må fylle inn den på nytt.
- Iptables måtte inn på nytt siden vi ikke kom på internett.
- Deretter i installasjonsprosessen fikk vi problemer med partisjoneringen. Der ble løsningen å legge til en parameter 'install-disk' byttes ut fa /dev/sda til /dev/cciss. Gjøres i foreman under partition tables eller under maskinen sine parametere i opprettelsesprosessen.
- Til slutt slet vi med at serveren ikke ville boote fra hdd etter at PXE boot var gjennomført. Fikset ved å endre noen linjer i /var/lib/tftpboot/pxelinux.cfg/<MAC>

30. Januar

- Har sliti med at host-ene blir lagt til med feil navn i foreman. De blir lagt til som "navn".nostrand.auto.stack i stedet for "navn".auto.stack. Dette viste seg å være en skrivefeil fra våres side i /etc/dhcp/dhcpd.conf, der "`option domain-name`" var satt som "`nostrand.auto.stack`" i stedet for "`auto.stack`".

02. Februar

- Siden Foreman er i boks, begynte vi å legge inn skriptene fra "AutoStack-gutta". Det vil si gjøre små endringer i skriptene som trengs slik at de blir tilpasset vår situasjon.
- Møte med Erik og Eigil.

04. Februar

- Slet litt med annen versjon (1.2.0) av rabbitmq (Feil i dokumentasjon fra i høst). Installerte rabbitmq versjon 2.1.0 i stedet.
- Hadde ikke kommentert bort alt av munin i deploy-skriptet (site.pp).
- La til node default i site.pp fordi puppet agent -t fant ikke nostrand.
- Sletta ntp puppet modul i /etc/puppet/production fordi den var på feil sted.
- Kan logge inn i Horizon etter disse endringene hadde blitt gjort.
- Nova er ikke oppe i Horizon.
- DHCP fungerer ikke på kontroller??

05. Februar !

- Fant problemet med DHCP, viste seg at interfacene hadde bytta plass (eth0 ble til eth1, eth2 ble til eth0, osv.). Løste ved å endre statisk ipconfig sitt interface.
- Compute1 ville ikke kjøre puppet riktig, PXE-booter installasjon på nytt.
- Korrekturrettet litt på deployskript, det viste seg at flere linjer hadde delt seg under kopieringen. Doh.
- VT-X var ikke på, ga oss libvirt-feil og sa at prosessoren ikke støttet OSet.
- NTP-servere er endret til noe fornuftig, corp.com var visst ikke en virkelig addresse.

06. Februar

- Klarer ikke å lage et instance. Har forsøkt i hele dag med forskjellige små konfigurasjoner i .conf filer osv., men uten hell.

09. Februar

- Demo med oppdragsgiver og veileder. BRA DEMO! :) (oppdragsgiver approves!)
- Endret noen linjer i nova.conf, neutron.conf og ml2.conf.ini hos kontroller, network, og compute (Fulgte Openstack IceHouse installation guide). http://docs.openstack.org/icehouse/install-guide/install/apt/content/neutron-ml2-controller-node.html

10. februar

- Prøvde igjen å endre på konfiger, men fulgte denne gangen oppdateringsguiden til openstack for både havana og IceHouse.
- http://docs.openstack.org/openstack-ops/content/ops_upgrades_grizzly_havana-ubuntu.html
- Endret to linjer i site.pp: byttet ut rpc backend til bare 'rabbit'.
- Skal prøve igjen i morgen, med puppetmaster avskrudd og manuell konfig.

11. februar (8 timer)

- Har prøvd igjen i dag, med puppet master avskrudd, å endre konfigfilene til nodene ved å se på havana->icehouse upgrade manual (http://docs.openstack.org/openstack-ops/content/upgrades_havana-icehouse-ubuntu.html), og iceHouse manual (http://docs.openstack.org/icehouse/install-guide/install/apt/content/neutron-ml2-control

ler-node.html), men uten hell.
- Sliter med at server.log til neutron gir oss: network_type=”gre” not supported
- Skal prøve å installere alle noder på nytt og prøve det samme, men vi skal ikke aktivere “gre” denne gangen.

12. Februar (7 timer)
- Endret sshd_config slik at flere kan jobbe på samme maskin samtidig.
- Fortsetter med å feilsøke problemer i Horizon. Fant ut at notify_nova osv. i /etc/neutron/neutron.conf var kommentert bort, og ikke satt på alle noder bortsett fra controllernoden. Da ble instansen satt opp!
- Satt opp ruter i Horizon. Lagt til floating ip til en instans. Instansen har nett, og dns.
- Vi har også funnet ut at BIOS innstillingene til Compute1 resettes hver gang strømmen kuttes på maskinene, antakelig på grunn av defekt/ikke eksisterende BIOS-batteri. Det som skjer det er blant annet at HW virtualisering disables og at Boot orderen endres.

13. februar (7.5 timer)
- Begynte dagen med å undersøke hva som måtte endres for at puppet kunne kjøre, på både server og klienter, og at vi kunne gå inn i Horizon uten å få error meldinger. Det viste seg at det var nok å legge til: core_plugin => ‘ml2’ i neutron klassen på alle nodene i site.pp.
- Vi begynte så å gjøre endringer i site.pp slik at den automatisk legger til de nødvendige endringene fra Havana til Icehouse.
- Men, så fant vi ut at vi ikke klarte å spawne nye instancer! Etter en lang og intens feilsøkingsprosess fant vi ut at tenant_id i neutron.conf var feil. Av en eller annen mystisk grunn så hadde vi c5693ffed4de4916b**9**369bd9f19a3c4d, mens det korrekte var: c5693ffed4de4916b**6**369bd9f19a3c4d. Når denne endringen ble gjort, så fungerte alt som dagen før, men nå med puppet aktivert på alle maskiner!

16. Februar (8 timer)
- Fortsetter med å automatisere Openstack IceHouse med puppet. Lager også et skript for å håndtere automatisering av tenant_id. Kjører alt på nytt for å se om puppet gjør det den skal.
- Lage ny instans i Horizon fungerte ikke. Fant ut at variabelen bridge_mappings ikke var riktig satt.
- Hadde møte med oppdragsiver og veileder.

17. Februar (3 timer)
- Reinstallert på nytt, og håpet på at alt gikk knirkefritt. Det viste seg at tjenester ikke ble restartet riktig etter at deployskriptet var kjørt, og en reboot på hele systemet vart løsningen for i dag. I morgen skal vi fikse dette, og alt skal gå automatisk.
- Heat er installert manuelt med suksess, og i morgen kommer også arbeidet med å få dette inn i puppetskriptet.

18. Februar (8 timer)
- Reinstallert to ganger i dag; den ene gangen med heat sånn halvveis fungerende men med feil region (RegionOne i stedet for regionOne). Dette gjorde at systemet vårt ikke fungerte helt som det skulle og vi måtte reinstallere enda en gang. Denne gangen fungerte alt strålende etter opptil flere puppet agent-runs og en reboot på alle maskiner.
- SSH-server er nå skrudd på gjennom puppet, og burde fungere etter den siste puppet-kjøringen.

19. Februar (2.5 timer)
- begynt på oppgradering til Juno og sett litt på integrering av skript inn i site.pp
- Det som er viktig nå framover: oppgradere til Juno og integrere shell-skript inn i puppet-konfigen.

24. Februar (8 timer)

- Oppdaterte AutoStack til Juno manuelt, og dette fungerte. Vi kunne spawne instances og heat så også ut til å fungere.
- Begynte så å se på puppet koden å skriptene for å få til dette automatisk.

25. Februar (8 timer)

- Fant ut at vi ikke hadde oppdaterte skript, kjørte dermed inn backupen av newman.
- Etter backup fikk vi problemer med PXE da vi prøvde å reinstallere systemet
- Dette fikset seg ved å avinstallere alle smart-proxyer i Foreman for så å installere på nytt.
- Nå vil ikke ting installere seg, det stopper på ~6% i basesystemet eller når flere maskiner kjører installasjon samtidig.

26. Februar (8 timer)

- Nostrand fikk en ny IP adresse i dag, å etter å ha lagt inn imaget på nytt, så fungerte det. Eneste forskjellen var at vi ikke forandret modulene før installasjon var igang……...rart.
- Igang med å fikse site.pp og skriptene slik at de gjør de riktige tingene mot Juno.
- Feilet litt i dag med at det sannsynlighvis var IceHouse pakkene som var installert og ikke Juno, fikk oppdatert pakkene på slutten av dagen.

2. Mars (8 timer)

- Reinstallerte OpenStack, og tømte hele site.pp. Vi tok hver vår node, og satte sammen site.pp helt fra scratch.
- Mens vi bygde opp site.pp igjen la vi i tillegg til Juno oppgraderinger, og å få mest mulig fra deploy-skriptene inn i puppet.

3. Mars (8 timer)

- Fortsettelse fra gårsdagen med å implementere OpenStack Juno i Puppet sin site.pp, i tillegg til å få mest mulig kode fra deploy-skriptene inn i site.pp.

6. Mars (4 timer)

- Fortsettelse fra tirsdagen med å implementere OpenStack Juno i Puppet sin site.pp, i tillegg til å få mest mulig kode fra deploy-skriptene inn i site.pp.

9. Mars (6 timer)

- Fortsatte å slite med puppet, og sette deploy-skript kode inn i site.pp.
- Fikk opp Horizon igjen.
- Møte med arbeidsgiver og veileder.
- Fant ut at vår "versjon" av Openstack Juno kanskje ikke var oppdatert, eller at utviklerne ikke var 100% ferdige med å utvikle modulene.

10. Mars (5 timer)

- Kjørte git clone for å hente ut de siste oppdateringene i Openstack Juno modulene.
- Gjorde nødvendige endringer i site.pp på grunn av oppdateringene (eks identity_uri). Fikk fjerna noen exec-er på grunn av dette.
- Gjorde om mer kode fra deploy-skript til puppet kode.

11. Mars (8 timer)

- Flyttet det siste av skript over i site.pp, og etter litt bug fiksing så fikk vi til å rulle ut AutoStack med Juno automagisk!
- Begynte så å fordele litt små arbeid videre, rydding av site.pp, backups, begynt på å skrive inn en patch til puppet-neutron modulen.

12. Mars (7 timer)

- Småfikset på diverse ting i site.pp, blant annet setting av rabbitmqpassord.

- testet på HP DL 320 G5-server, boot fungerte ikke helt riktig
- Tatt Juno image, backup er fikset
- Dokumentasjon og litt rapport er skrevet
- begynt på GRE-implementering i stedet for VLAN.
- Testet at heat fungerer på Juno

13. Mars (8 timer)
- Fikset boot på HP DL 320 G5-server
- Testing av GRE i stedet for VLAN (implementert i site.pp)
- Begynt så vidt å sjekke ut/teste Rally

16. Mars (6 timer)
- Harald tok med seg et par servere med Intel og AMD prosessor, som ble brukt som compute-noder.
- Fortsatte testing av GRE i stedet for VLAN. Kan lage instanser, men instansene kan ikke ssh til andre og omvendt.
- Fiksa bug med glance images i site.pp. Fiksa et par bugs med enable IPv4 forwarding på network, og disable packet filtering på compute i site.pp.

17. Mars (5 timer)
- Fant ut av problemet med at vi ikke fikk koblet til instansene via SSH. Vi måtte endre MTU størrelsen på pakkene til 1454 på grunn av at GRE lager litt overhead. Dette var beskrevet i OpenStack guiden, så vi fikk implementert dette i site-en vår.
- Installerte hele oppsettet på nytt, denne gangen med VLAN, slik at vi også her kan være 100% sikre på at vi får kontaktet instansene via SSH, og at installeringen er 100% vellykket. Det var det ikke….vi sliter også her med å få kontakt med vm-ene, får ikke pinget dem engang.

18. Mars (8 timer)
- Fortsatt å feilsøke med VLAN oppsettet og tilkobling til VM-ene via SSH. Har funnet et par feil og rettet disse, men det har ikke hatt noe utslag mens systemet kjører. Vi prøvde derfor å installere alt på nytt.

19. Mars (7 timer)
- Fortsatte feilsøking og fant ut at de to Exec-ene i site-en som skal sette inn bridgene på nettverks og compute nodene, ikke har gjort dette ordentlig. Alt ble nemmelig ikke lagt inn.
- Etter at dette var på plass brukte vi litt tid på å feilsøke at det eksterne nettet ikke fungerte. Fant til slutt ut at vi hadde koblet br-ex(eth2) fra nettverksnoden i feil vlan i switchen, eller vi hadde "bridget" fra et vlan til et annet, som egentlig var feilen. Etter at vi fikk koblet denne i samme vlan som det eksterne nettet, så fikk instansene nett og vi kunne koble oss til med SSH.
- Installerer alt på nytt for å forsikre oss om at alt skal settes opp automatisk med VLAN.
- Noen småting som ikke gikk automatisk, måtte restarte nettverk og compute nodene i forbindelse med å legge til bridger og porter. Satser på å fikse dette gjennom restart av services i siten.

20. Mars (8 timer)
- Skrev statusrapport for måneden til oppdragsgiver og veileder.
- Lagt til en ekstra service restart for å få bridger til å fungere i site.pp.
- Verifisert at full reinstallering med GRE fungerer, i tillegg til at alle bridger blir satt riktig. Verifiserer at OpenStack Juno fungerer 100% automagisk etter reinstallering.
- Tok nytt image av Nostrand.

24. Mars (5 timer)
- Hadde en liten brainstorming for å lage en liste over ting som må sjekkes i test-rammeverket.

- Brukte Rally for å implementere sjekk-listen for test-rammeverket.
- Litt mer rydding i site.pp (NTP og SSH har bare en klasse).

25. Mars (4,5 timer)
- Fortsatte jobbing med test-rammeverk for VM-er.
- Litt mer rydding i site.pp.
- Fiksa noen bugs i site.pp (en linje var kommentert bort, ugyldig navn på en klasse).
- Oppdatering av rootwrap førte til at Heat ikke fungerer.

26. Mars (7 timer)
- Jobbet hardt med Heat, ingen framgang
- Mer komprimering av site.pp, mindre duplikasjon av felles kode.
- testing av Rally med kommadnoer på VMer: Cinder klager litt. Vi her ikke Cinder fra før av, og installeringen går ikke så bra.
- Ser ut som om rally skaper noen konflikter med python-biblioteket vårt og lager noen avhengighetsproblemer der. Installerer kontroller på nytt uten rally og ser om det hjelper til i morgen.

27. Mars (8 timer)
- Fortsatte med test-rammeverk og Rally.
- Hadde installasjonsdemo med oppdragsgiver og veileder.
- Fikk beskjed om at Puppet skulle integreres i Foreman, altså dele opp site.pp til en modul som du la til i Foreman når du oppretter en node.
- Begynt å skrive rapport

28. Mars (5 timer)
- Har klart å flytte hele site.pp inn i en egen modul slik at vi nå kan legge til hva som skal kjøres av puppet kode i Foreman når du skal oprette en node.
- To ting som mangler før dette blir helmaks, et automatisk interface oppsett og 'overiding' av variabler i Foreman - noe vi ikke har fått til å fungere, man må fortsatt endre dette i .pp filene.

29. Mars (8 timer)
- Reinstallerte alt etter at site.pp ble til "autostack" modulen.
- Fiksa stage 'first' som ikke fungerte (måtte være utenfor klassen). Reinstallerte alt på nytt.
- Fiksa noe DNS greier. Reverse zone fungerte ikke, måtte slette /etc/bind/zones/db.10.jnl.
- Installerte Rally på Nostrand slik at vi nå kan kjøre tester mot OpenStack deploymenten derfra. Har også installert Cinder på Controlleren siden noen spesielle tester krevde dette.
- La til dns-nameservers i /etc/resolvconf/resolv.conf.d/head i stedet for rett inn i /etc/network/interfaces.
- Laget et skript /root/node_startup.sh som bytter interfacenavn på 10.0.0.0/24 -nettet til eth0 og rebooter maskina. Burde fikse kabelswapping.
- Skrevet mer på rapporten.

30. Mars (8 timer)
- Fikset et DNS-problem som vi ikke så før vi hadde restartet alle maskinene
- Fikset ethX-navngivinga; nå forblir det nettet som man booter fra eth0. Kanskje emX-nettverk fungerer, men gjenstår å teste.
- Enda mindre duplisering av kode; eth0 ligger i common-klassa takket være DNS-fix.
- Fikk ENDELIG til Foreman sin overstyring av variabler. Praktisk å ha nå når vi skal teste
- nettverk og controller på samme maskin: i testingfasen.

7. April (8 timer)

- Gjort testing(reinstallering) med den nye "autostack-modulen" vår, både når det gjelder en samlet nettverks og kontroller-node, og hver for seg. Har funnet småfeil her og der

som vi har rettet på.

8. April (7.5 timer)
- Samme som i går.

9. April (8 timer)
- Skrevet mye på test-rammeverket. Fullført interne openstack-funksjoner som oppretting av nettverk, modifisering av regler, oppretting av instance, heat-funksjonalitet, osv. Ting som gjenstår: kjøring av skript på VM og testing av tjenester ellers.
- Skrevet mer på rapporten.

10. April (8 timer)
- Samme som i går (jobbing med test-rammeverket).
- Laget et skript med all funksjonalitet (ALT FUNKER!!!!!! bortsett fra ssh-bug??).

13. April (8 timer)
- Har rettet SSH-buggen i scriptet slik at hele scriptet nå kjører uten feil.
- Har gjort slik at test-scriptet nå kan kjøres fra nostrand og ikke fra controlleren slik vi hadde gjort det.
- Begynt å sett på oppsett av en storage node til bruk av cinder.
- Ferdig med en "testrammeverk webside" som oppdateres automagisk.
- Fikk også to enkle templates fra Digitalt Øvingsfelt-gutta som vi skal se nærmere på etter hvert.

14. April (8 timer)
- Fikset cinder manuelt, jobber fremdeles med puppet-implementasjon. 'storage' vil da bli en egen klasse inne i foreman.
- Fått test-rammeverket inn i foreman gjennom en egen plugin
- Testet små templates fra Digimon.
- Skrevet mer på rapporten.

15. April (8 timer)
- Cinder er nå implementert i autostack-modulen vår og vi har nå en egen storage klasse hvis det skulle være behov for å bruke volumes.
- Lagt til test av Cinder i test-rammeverket + forbedringer(hva skjer når det er en error osv.).
- Skrevet mer på rapporten.

16. April (8 timer)
- Ny reinstallasjon av alle maskiner.
- ssh og ntp var ikke deklarert i common.pp. Dette ble fikset.
- Fiksa en del bugs i test-rammeverk (både output og funksjonalitet).

27. April (8 timer)
- Har vært i kontakt med Digital Øvingsfelt gruppa på mail fordi vi har hatt problemer med å rulle ut templaten deres. Først å fremst var det en ekstra plugin i heat som måtte på plass, og denne er nå i orden. Men så dukket et nytt problem opp. Vi fikk denne feilmeldingen når vi skulle kjøre heat stack-create: "ERROR: unknown property port". Vi fant etter hvert ut at de i sin template hadde brukt en property "port" som er innført med nyeste versjon av OpenStack, nemlig Kilo. Vi har sendt mail om dette til de men ikke fått noe svar enda.
- Etter samtale med veileder og oppdragsgiver har vi fått klarhet i om test-rammeverket til scenarioer som skal rulles ut, skal skrives av oss eller ikke. Det skal vi ikke! Dette er gode nyheter siden ikke hadde jobbet veldig mye med dette, og at vi slipper å gjøre det (har dårlig tid). Bedre å heller bruke tiden som er igjen til å pusse/polere på det vi har.
- Ellers har vi gjort små forbedringer på test-rammeverket på skyløsningen; vi har lagt til sjekk om det er et VLAN eller et GRE nettverk som skal opprettes, og vi har lagt til litt mer sjekk for et heat scenario, slik at man kan få en grunnleggende sjekk om templaten faktisk fungerer.

- Har også skrevet litt på rapporten.

28. April (8 timer)
- Skrevet masse på rapporten. Implementering, utfordringer og introduksjon.
- Fikset på test-rammeverket, skrevet om til engelsk, fikset output og endret på noen bugs. Direkte på nostrand fordi nano ikke er så glad i hermetegn og SSH.

29. April (8 timer)
- Re-innstallerte alt med VLAN konfigurasjon.
- Fiksa på bugs og utseende i test-rammeverk, og i puppet-modulen.
- Re-installerte igjen alt igjen etter å ha fikset på ting.

04. Mai (8 timer)
- Har begynt å sette opp Foreman på fra scratch, der vi følger implementerings-dokumentasjonen som vi har skrevet. På denne måten finner vi ut om det vi har skrevet holder mål, siden vi ønsker at det nesten bare skal være å copy/paste fra dokumentasjonen å sette opp Foreman.
- Fortsatt skriving av rapport.
- Møte med oppdragsgiver og veileder. Fikk i dag bekreftet at vi ikke skal rulle ut Digitalt Øvingsfelt sine scenarioer siden de har benyttet seg av mye Kilo funksjonalitet.

05. Mai (8 timer)
- Nesten ferdig med å gå igjennom oppsett av Foreman og Nostrand fra scratch.
- Fortsatt skriving av rapport.

06. Mai (8 timer)
- Fortsatt skriving av rapport.
- Gikk igjennom rapporten etter feil og mangler før det skal sendes til veileder.
- Instancer vil ikke spawne eller få nett pga. "tenant id-setter", og nova/neutron ikke helt fungerer i IceHouse. Denne feilen har vi ikke fått før...

07. Mai (8 timer)
- Gikk igjennom resten av rapporten etter feil/mangler. Sendte utkast til veileder.
- Fortsatte å feilsøke etter "tenant id-setter" melding fra Puppet.

08. Mai (8 timer)
- Fortsatt skriving av rapport.
- IceHouse fungerer! dnsmasq skulle ikke være med i VLAN konfig på nettverksnoden.

09. Mai (8 timer)
- Fortsatt skriving av rapport.
- Sjekker fortsatt om automatisk utrulling av IceHouse fungerer etter dokumentasjonen i rapporten. Har problemer med "nova-tenant-id-setter" i IceHouse. (Fikset ved å legge til en exec i scriptet)

10. Mai (8 timer)
- Fortsatt skriving av rapport.
- IceHouse fungerer bare ved å følge rapporten vår!
- Fiksa en del bugs og skjønnhetsfeil i test-rammeverk skriptet (er nå kompatibelt med IceHouse, måtte bare endre på "versjons dato" øverst i heat teamplaten).

11. Mai (8 timer)
- Rapport
- Testet Juno installering
- Møte med veileder der vi fikk tilbakemelding på utkast til rapport

12. Mai (10 timer)
- Rapport

13. Mai (11 timer)

- Rapport

14. Mai (10 timer)
- finpussing og sammensetting av rapport

H Prosjektavtalen

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

ERIK HJELMÅS

(oppdragsgiver), og

HARALD SLETTEN,
TERJE PEDERSEN, Kevin A. Barhaugen (student(er))

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

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 8.1.2015 til 15.05.2015.

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

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

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

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

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

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

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

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

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

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

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): EIGIL OBRESTAD

Oppdragsgivers kontaktperson (navn): ERIK HJELMÅS

Student(er) (signatur): _____ dato _____

Kevin A. Barhaugen dato 26/1-15

Eirik Pedersen dato 26/1-15

Harald Sletten dato 26/1-15

Oppdragsgiver (signatur): _____ dato 26/1-15

IMT Dekan/prodekan (signatur): _____ dato _____

Versjon Januar 2011hb