NTNU
Norwegian University of
Science and Technology

Nytt konfigurasjonssystem for Linux-servere

Forfattere
Audun Huseby
Stein Ove Jernberg
Andreas Osborg

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

16.05.2018

Veileder Eigil Obrestad

Sammendrag av Bacheloroppgaven

Tittel:	**Nytt konfigurasjonssystem for Linux-servere**
Dato:	16.05.2018
Deltakere:	Audun Huseby Stein Ove Jernberg Andreas Osborg
Veiledere:	Eigil Obrestad
Oppdragsgiver:	NTNU, seksjon for IT-drift
Kontaktperson:	Erlend Midttun, (Teamleder Unix v/Seksjon for IT-drift), erlend.midttun@ntnu.no, 73590398 / 92401961
Nøkkelord:	Konfigurasjonsstyring, CM-system, Puppet, Ansible
Antall sider:	59
Antall vedlegg:	10
Tilgjengelighet:	Åpen

Sammendrag:

Unix-gruppen ved NTNUs seksjon for IT-drift skal finne et nytt konfigurasjonssystem for styring av Linux-maskiner. Det skal erstatte dagens implementasjon av CFEngine 2. En oppgradering til versjon 3 fra samme leverandør vil kreve at alle konfigurasjoner uansett må skrives om. De ønsker derfor en vurdering også av andre aktuelle kandidater. Prosjektgruppens problemstilling ble derfor å finne det de mente var den beste kandidaten for Unix-gruppen. For å finne aktuelle kandidater må et utvalg tas blant mange potensielle systemer. Når utvalget er tatt må en nærmere vurdering av de aktuelle kandidatene tas for å skaffe et godt inntrykk. Når vurdering er gjort skal det velges ut to kandidater til et eksempeloppsett som skal vise hvordan de kan tas i bruk og i seg selv driftes. Ansible og Puppet ble tatt med til et eksempeloppsett hver. Puppet endte opp med å bli anbefalt, blant annet fordi systemet oppleves som mer komplett i forhold til Ansible.

Summary of Graduate Project

Title:	**Replacement configuration management system for managing Linux machines**
Date:	16.05.2018
Authors:	Audun Huseby Stein Ove Jernberg Andreas Osborg
Supervisor:	Eigil Obrestad
Employer:	NTNU, seksjon for IT-drift
Contact Person:	Erlend Midttun, (Teamleder Unix v/Seksjon for IT-drift), erlend.midttun@ntnu.no, 73590398 / 92401961
Keywords:	Configuration management, CM System, Puppet, Ansible
Pages:	59
Attachments:	10
Availability:	Open

Abstract:

The Unix team at NTNU's section for IT operations is in the process of finding a replacement for their implementation of CFEngine 2 as their configuration management system for Linux servers. They will end up rewriting every configuration in the case of upgrading to version 3 from the same vendor. The Unix team would like a review to be made of other candidates as well. The project group's research issue is to determine, in their opinion, the best candidate for a new system. In order to find potential candidates, there has to be an election process. When four candidates have been elected, a review of them has to be done in order to gain a sufficient impressoin of each. When all have been reviewed, two of the candidates will be chosen for an example setup. The setup will demonstrate how the system may be used and operated. Ansible and Puppet were each chosen for an example setup. Puppet became the recommended candidate. The main argument behind this recommendation is that Puppet is experienced as a more complete solution compared to Ansible.

Innhold

Figurer

Tabeller

Listings

1 Innledning

1.1 Bakgrunn

Konfigurasjonssystemer for IT-infrastrukturer kan benyttes for å styre servermaskiner med deres installerte tjenester. Slike systemer sørger for å utføre de nødvendige oppgavene for å oppnå en ønsket tilstand, samt stadig verifisere denne [106]. Konfigurasjonssystemer vil være til stor nytte når man administrerer svært mange servermaskiner (fysiske og virtuelle), slik at konfigurasjonen forblir konsistent og i den tilstanden man ønsker.

Unix-gruppen ved NTNUs IT-avdeling i Trondheim benytter per i dag CFEngine[1] versjon 2 for synkronisering av konfigurasjoner på deres Linux-maskiner. Det brukes til å installere programpakker, kopiere konfigurasjonsfiler, sørge for at tjenester kjører og eventuelt kjøre noen script eller kommandoer. For at riktige maskiner får den konfigurasjonen de skal ha, kan de grupperes.

Versjon 2 av CFEngine ble først lansert i 1998 og leverandørens støtte har opphørt. I dag tilbys CFEngine versjon 3, men denne versjonen er ikke bakoverkompatibel med versjon 2. Problemet til Unix-gruppen de ikke kan oppgradere CFEngine uten å måtte skrive helt nye konfigurasjonsdefinisjoner. De er nødt til å finne et nytt system, og siden det finnes mange bra konkurrenter til CFEngine i dag, bør disse også vurderes.

Basert på denne problemstillingen valgte Unix-gruppen høsten 2017 å komme med et oppgaveforslag til bachelor-prosjekt. Prosjektgruppa fattet interesse for denne, tok kontakt og inngikk kontrakt.

1.2 Oppgaven

1.2.1 Problemområde

Konfigurasjonssystemer kan løse to vanlige problemer med styring av servere, som på engelsk blir kalt “configuration drift” og “snowflake servers” [106]. Førstnevnte beskriver en uønsket tilstand der konfigurasjoner for et sett med maskiner som i utgangspunktet var konfigurert identisk, har blitt konfigurert forskjellig som følge av “ad-hoc”-endringer på enkeltmaskiner over tid. Maskinene gir kanskje lik funksjon, men konfigurasjonen er ikke lenger identisk, slik det var ment. Sistnevnte beskriver en tilstand der alle ad-hoc-endringene har akkumulert i en såpass stor grad at den har endt opp som en spesiell maskin. Eller at maskinen var spesiell og unik i utgangspunktet, noe som er uønsket når man styrer flere hundre, kanskje tusen maskiner. Snowflake-servere er ikke uvanlig, men bør begrenses til de man bruker som kontrollere av andre maskiner eller “jumphost” (en proxy mot et isolert nettverk).

Dette er problemstillinger Kief Morris har beskrevet i sin bok, *Infrastructure as Code*. Konfigurasjonssystemer sørger for å holde konfigurasjoner konsistent, og for å forhindre ad-hoc-endringer på et sett med like maskiner. Noe annet som er viktig med konfigurasjonssystemer er at konfigurasjonene kan skrives i tekstfiler, som YAML, JSON eller DSL

[1] https://cfengine.com/

(domain specific language). Disse filene bør være rene tekstfiler som kan versjonskontrolleres i f.eks Git. Unix-gruppen ved NTNUs IT-avdeling benytter i dag versjonskontroll for konfigurasjoner med CFEngine.

Det å definere elementer av infrastrukturen i tekstfiler, legge tekstfilene i et versjonskontrollsystem, og la et konfigurasjonssystem utføre konfigurasjon, er alle prinsipper fra *Infrastructure as Code*[106], eller *programmerbar infrastruktur* som det heter på norsk (se 2.4).

1.2.2 Oppgavebeskrivelse

Prosjektgruppen skal på vegne av oppdragsgiver gjøre et utvalg og vurdering av ulike konfigurasjonssystemer for Linux-maskiner, utvikle et eksempeloppsett og komme med en konkret anbefaling. Følgende problemstilling ble styrende for arbeidet i prosjektet:

Hva er det beste konfigurasjonssystemet for Unix-gruppen ved NTNUs IT-avdeling?

Det Unix-gruppen trenger er altså den faglige vurderingen av en håndfull aktuelle systemer og en demonstrasjon av hvordan systemene kan rulles ut, brukes i hverdagen og i seg selv driftes. Det krever at arbeidet produserer to ting til oppdragsgiver: Denne rapporten med vurderinger og beskrivelser, og koden fra et eksempeloppsett. Mange konfigurasjonssystemer gjør i hovedsak den samme jobben, men de har blant annet noen tekniske og arkitekturmessige forskjeller som kan ha betydning for bruk og organisering av konfigurasjoner. Da er det prosjektgruppens jobb å vurdere de vesentlige forskjellene.

For å gi en faglig vurdering er det nødvendig å formulere og planlegge ulike ting til prosessen. Først og fremst en kravspesifikasjon og vurderingskriterier for å gjøre et utvalg. Selve utvalgsprosessen må også være nøye planlagt for å gi et upartisk utvalg. En vurderingsfase skal gi et grunnleggende inntrykk av aktuelle kandidater for å kunne velge ut én eller flere kandidater til eksempeloppsett. Beskrivelser i denne rapporten og kode fra et eksempeloppsett skal vise hvordan man kan jobbe med et CM-system

1.2.3 Avgrensing

Hovedfokuset for denne rapporten er å presentere den faglige vurderingen og utvalgsprosessen, og å presentere aspekter og eksempler for bruk av de utvalgte konfigurasjonssystemene. Prosjektgruppen skal ikke utvikle en fungerende løsning for direkte implementering. For vurderingsfasen av aktuelle kandidater, skal bare en overflatisk gjennomgang gjøres for å gi et godt nok inntrykk til å kunne bestemme hvem som skal tas med til et eksempeloppsett.

1.3 Formål

Denne rapporten skal forhåpentligvis være til hjelp når Unix-gruppen gjør et valg av nytt konfigurasjonssystem for styring av Linux-maskiner. Den vil fungere som en utredning av hvilke muligheter som finnes når det gjelder å velge et passende system, samt hvordan det best mulig kan brukes for å styre servermaskiner.

Prosjektgruppen valgte denne oppgaven fordi alle medlemmene hadde litt erfaring med konfigurasjonssystemer fra før, og ønsket spesielt å utvide horisonten for teknisk forståelse av slike systemer.

1.4 Målgruppe

Oppdragsgiver er rapportens og eksempeloppsettets hovedsaklige målgruppe, siden de er ansvarlige for systemer og maskiner styrt gjennom løsningen med CFEngine og har planer om å velge en erstatter. Rapporten skal i første omgang leveres til Institutt for informasjonssikkerhet og kommunikasjonsteknologi[2] under Fakultet for informasjonsteknologi og elektroteknikk ved NTNU. Prosjektgruppen håper rapporten kan være nyttig også for andre aktører som er i en lignende situasjon som Unix-gruppen.

1.5 Prosjektgruppens bakgrunn

De tre gruppemedlemene av prosjektgruppen går samme studie, Drift av nettverk og datasystemer, ved NTNU i Gjøvik. Alle tre har erfaring og kunnskap om styring av Linux- og Windows-maskiner, web-, database og applikasjonstjenester, drift av ulike tjenestearkitekturer, prinsipper for programmerbar infrastruktur, bruk av utviklingsverktøy og systemer for endringskontroll, programmering og nettverk.

Dette prosjektet vil kreve at alle medlemene av prosjektgruppen tilegner seg ny kunnskap om konfigurasjonssystemer, mønstre for styring og organisering av maskiner, samt drift av systemet i seg selv.

1.6 Roller

Oppdragsgiver er seksjon for IT-drift ved NTNU[3], representert ved Erlend Midttun. Oppgaven er mer spesifikt rettet mot Unix-gruppen, som Erlend er leder for. Vår veileder er Eigil Obrestad, ansatt ved IIK under IE-fakultetet. Gruppeleder for prosjektgruppen er Audun Huseby. Øvrige roller for arbeidet har blir bestemt underveis.

1.7 Rapportens innhold

1.7.1 Kapittelinndelinig

Rapporten vil bestå av 4 hovedkapitler og avslutning i tillegg til denne innledningen.

Kapittel 2, Teori, beskriver teorien som er styrende for rapportens sammenheng.

Kapittel 3, Kvalifisering, utvalg og vurdering, beskriver utvalgs- og vurderingsprosessen. Et initielt utvalg ble gjort, så en individuell vurdering av hver aktuelle kandidat, deretter en påfølgende diskusjon og endelig utvalg av kandidat for eksempeloppsett.

Kapittel 4, Utvidet beskrivelse av Ansible og Puppet, gir en dypere innsikt i aspektene ved de utvalgte kandidatene for eksempeloppsett.

Kapittel 5, Eksempeloppsett, gir beskrivelser av eksempeloppsettet som ble gjort for de utvalgte kandidatene, øvrige beskrivelser for drift og vedlikehold av systemene i seg selv, og diskusjon av resultatene.

Til slutt i kapittel 6, Avslutning, blir prosjektets resultater diskutert og anbefalingen av konfigurasjonssystem drøftet. I vedlegg finnes blant annet Kravspesifikasjon og Initielt utvalg.

[2] https://www.ntnu.no/iik
[3] https://www.ntnu.no/adm/it

1.8 Terminologi

Konfigurasjonsstyring

Konfigurasjonsstyring, slik uttrykket blir anvendt i denne rapporten, beskriver hvordan servermaskiner blir konfigurert med programvare for å kunne tilby en tjeneste.

CM-system

En forkortelse for "konfigurasjonssystem"/"konfigurasjonsstyringssystem".

Roles, Modules, Cookbooks og Formulas

Ulike leverandørers navn på gruppering av konfigurasjoner. Disse navnene betyr i hovedsak det samme: Kode for konfigurasjonsdefinisjoner samlet i en katalog med en spesiell struktur.

Tjener, server, maskin, noder

En datamaskin installert med programvare som kan nås over nettverket. Disse ordene brukes litt om hverandre i denne rapporten, men refererer til det samme.

Virtuell maskin

Programvare som emulerer fysisk maskinvare for et operativsystem å kjøre isolert i.

HTTP-API

En tjeneste som tillater å bruke HTTP-protokollen for klienter å kommunisere og hente informasjon.

Domenespesifikke språk

Fra engelske "Domain-Specific Language" (DSL), der et programmeringsspråk har en spesiell tilknytning til et datasystem.

YAML/JSON

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

Ad-hoc

Gjøre endringer utenfor konfigurasjonsverktøyet (i sammenheng med denne rapporten).

CI/CD

Det å kunne utvikle og teste ny funksjonalitet i programvare, ofte og inkrementelt.

CMDB

Engelsk forkortelse for "configuration management database". En database for å holde orden på metadata om inventar i en IT-infrastruktur, som regel maskiner og datasystemer.

SkyHiGh

Navnet på implementasjonen av skytjenestesystemet OpenStack ved NTNU i Gjøvik.

Stack

En samling ressurser som opprettes i Openstack, ofte i virtuelle maskiner, nettverk og datalagring.

FQDN

Fullt kvalifisert domenenavn på maskiner i en IT-infrastruktur/domene. Eksempelvis “maskin01.domene.net”.

Repo

Forkortelse for det engelske ordet “repository”. Det er et oppbevaringssted for kode.

Refactoring

Vurdering, forbedring og fornying av eksisterende kode.

Runtime-dokumentasjon

Et uttrykk prosjektgruppen valgte å ta i bruk for å beskrive dokumentasjon som kan leses på kommandolinjen på en maskin.

2 Teori

2.1 Konfigurasjonsstyring

Når man driver en prosess med stadige endringer, flere involverte personer og mange elementer å styre, bør man etablere et system for å håndtere dette på systematisk vis. Konfigurasjonsstyring er en standardisert praksis og rammeverk for håndtering av ansvar, planlegging, identifisering, styring av endringer, overvåking og revisjon av en bedriftsprosess [92]. Det er et administrativt og teknisk rammeverk for å vedlikeholde framgangen og livssyklusen for et produkt eller tjeneste med dokumentasjon og overvåking av prosessen rundt. Målet er å spore *konfigurasjon*, *konfigurasjonsinformasjon*, *konfigurasjonselementer* og *statusinformasjon*.

I *konfigurasjonsstyring for programvare* finner man mange av temaene som ISO10007-standarden abstrakt beskriver, mer spesifikt rettet mot programvareutvikling. I boken *Software Engineering: a practitioner's approach* beskriver Roger S. Pressman og Bruce R. Maxim konfigurasjonsstyring for programvare (*software configuration management*, SCM) som en prosess for endringskontroll i programvare. Det blir også omtalt som *change management* [113].

SCM er et sett med aktiviteter for å styre endringer i livssyklusen for utvikling av et programvaresystem.

SCM-prosessen består av følgende aktiviteter:

- Identifisere objekter i konfigurasjonen (modeller/dokumenter, kodeelementer, osv.)
- Versjonskontroll for sporing av endringer
- Endringskontroll som en organisatorisk prosess for endringer fra de blir foreslått til de blir utviklet og implementert, samt dokumentere hvem, når, hva og hvorfor endringen ble gjennomført.
- Kvalitetssikring, eksempelvis ved bruk av testing og gjennomgang av kode før implementasjon

2.1.1 Konfigurasjon

Konfigurasjonen i konfigurasjonsstyring for programvareutvikling er elementene som inngår i hele utviklingsprosessen. Altså programkoden, kompilert program, data som inngår i kjøringen av programmet, dokumentasjon og prosedyrer, og verktøy som benyttes for å utvikle programkode og gjøre konfigurasjonsstyring.

Versjonskontrollsystemer, eksempelvis Git[1], sørger for å håndtere endringer i prosessen. Git har versjonskontroll, oppbevaring av kode og endringskontroll [195].

2.1.2 Konfigurasjonsstyring for infrastruktur

Hva er poenget med å beskrive prinsipper og elementer i en prosess for utvikling av programvare, når denne rapporten handler om styring av server-maskiner i en IT-infrastruktur? Konfigurasjonssystemer, slik de blir omtalt i denne rapporten, har blitt utviklet for styring

[1] https://git-scm.com/

av infrastruktur som tillater at dens elementer og tilstand kan deklareres i kode. Dette kalles *programmerbar infrastruktur* (*infrastructure as code*) og er en regelrett anvendelse av prinsippene fra konfigurasjonsstyring for programvareutvikling, i styringen av infrastruktur [106]. Endringer gjort i en infrastruktur kan også gå gjennom endringskontroll og versjonskontroll. Man kan også teste deklarasjonskoden og implementasjonen.

Forvirring kan oppstå når man snakker om konfigurasjonsstyring for infrastruktur. Slik *konfigurasjon* er definert i [113] har den litt forskjellig betydning fra det som oftest kalles *konfigurasjonsfiler*. Konfigurasjonsfiler er en del av *konfigurasjonen*. Konfigurasjon er som nevnt alle elementer som inngår i *prosessen* for konfigurasjonsstyring. Teoretisk sett kan eksempelvis designdokumenter og rutinebeskrivelser være en del av konfigurasjonen.

I resten av denne rapporten omtales konfigurasjon i hovedsak som definisjonsfiler i et CM-system, eller konfigurasjonsfiler til en eller annen programvare installert på en maskin.

2.2 Automatisering

Fundamentals of modern manufacturing : materials, processes, and systems (Mikell P. Groover, 4. utgave, 2010), en bok om materialproduksjon, definerer automatisering på følgende vis:

> Automation can be defined as the technology by which a process or procedure is performed without human assistance [89].

Denne definisjonen er universell, og ikke forbeholdt materialproduksjon. Likevel kan det hjelpe med en definisjon som er mer rettet mot bedriftsprosesser der automatisering ved hjelp av IT-systemer kan være svært nyttig. Viktigst av alt, automatisering kan være helt avgjørende for verdiskapning i en moderne bedrift. Gartner definerer automatisering for bedriftsprosesser slik:

> Business process automation (BPA) is defined as the automation of complex business processes and functions beyond conventional data manipulation and record-keeping activities, usually through the use of advanced technologies. It focuses on 'run the business' as opposed to 'count the business' types of automation efforts and often deals with event-driven, mission-critical, core processes. BPA usually supports an enterprise's knowledge workers in satisfying the needs of its many constituencies [88].

Det kan antas at "count the business" handler om digitalisering av informasjon, og "run the business" handler om automatisering av bedriftsprosesser. CM-systemer bidrar i allefall til å automatisere styring av IT-infrastruktur ved bruk av prinsipper for programmerbar infrastruktur.

2.3 Modelldreven utvikling

Abstraksjon blir brukt til å skjule underliggende kompleksitet. Ved abstraksjon kan man fokusere på det som er relevant der eksempelvis operativsystemets styring av maskinvareressurser er irrelevant for å definere en kodesekvens som behandler noen enkle data. John V. Guttag beskriver abstrahering slik i sin bok, *Introduction to Computation and Programming Using Python*: "The essence of abstraction is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context" [90].

Modelldreven utvikling tar sikte på å abstrahere programmeringsspråk ved å deklarere en modell med de ønskede elementene og attributtene man trenger [194]. Dette vil rette fokuset mer over på arkitektur, gi større muligheter for gjenbruk av kode og la utviklere tenke mer på effektiviteten av systemet i sin helhet. I modelldreven utvikling kombineres to ting: *Domenespesifikke modelleringsspråk* (domain-specific modeling languages, DSML) og *transformasjonsmotorer* eller *generatorer*. I [194] beskriver Douglas C. Schmidt transformasjonsprosessen med generatorer slik:

> This automated transformation process is often referred to as 'correct-by-construction,' as opposed to conventional handcrafted 'construct-by-correction' software development processes that are tedious and error prone

Når man i DSMLen deklarerer de ønskede elementene og attributtene, vil generatoren sørge for at underliggende kode, data, og andre nødvendige elementer blir generert. Dette gjelder for programvareutvikling. I denne rapportens sammenheng er tema rettet mot styring av infrastruktur og server-maskiner. Da vil man først deklarere et ønsket element (programpakke, konfigurasjon, tjenestestyring, e.l.), som deretter blir generert og rullet ut på en maskin.

Modelldreven utvikling brukes av konfigurasjonssystemer for IT-infrastruktur og servere, ved at man i klasser eller roller deklarerer de ressursene og elementene man vil ha på plass i infrastrukturen. Bruk av modelldreven utvikling forutsetter at utviklerne stoler og støtter seg på abstraksjonslaget i DSMLen og generatorsystemet. De store aktørene som tilbyr systemer for konfigurasjonsstyring av infrastruktur og servere har antakeligvis oppnådd en høy grad av robusthet i deres produkter.

2.4 Programmerbar infrastruktur

Når man har modelldreven utvikling ved bruk av CM-systemer, brukes også en del prinsipper som er løst definert under det som kalles *programmerbar infrastruktur* (*Infrastructure as Code*). Kief Morris har diskutert dette i sin bok *Infrastructure as Code: managing servers in the cloud*.

Det som menes med programmerbar infrastruktur er at prinsipper og arbeidsmetodikk fra programvareutvikling kan brukes for endringer gjort i en infrastruktur [106]. Styring av maskiner i en infrastruktur deklareres i kode, mates i et CM-system, og setter endringene i drift.

Ifølge Morris vil man kunne oppnå mer effektiv tidsbruk, færre feil, gjøre forbedringer og stadige endringer, og kunne respondere mer forberedt på feil som oppstår i infrastrukturen. Å ha en programmerbar infrastruktur vil legge til rette for og normalisere hyppige endringer.

Maskiner og systemer bør bli behandlet som "Cattle, Not Pets" (kveg i stedet for kjæledyr). De bør altså være helt like der bare domenenavn og IP-adresse skiller de (popkulturnavn i domenenavn bør unngås). Konfigurasjon vil bli mer konsistent, og maskinene kan da bli reprodusert i andre oppsett.

Når infrastrukturen deklareres som kode i tekstfiler, kan de også enkelt versjonskontrolleres og testes. På sett og vis kan man si at styringen av infrastrukturen blir et utviklingprosjekt.

2.4.1 Styring av maskiner

Ved bruk av programmerbar infrastruktur skal server-maskiner kunne automatisk provisjoneres, uten at mennesker gjør manuelle endringer på maskinen. Denne prosessen skal kunne være repeterbar, konsistent, selvdokumenterende og transparent. Alle endringer gjort i konfigurasjonsdeklarasjonen skal kunne versjonskontrolleres og testes. Endringer skal ikke gjøres utenom den automatiserte prosessen med CM-systemet (altså ingen manuelle endringer på maskinene).

Noen kjente navn på verktøy for styring av server-maskiner (CM-systemer) er Puppet, Chef, Ansible, og Salt. Disse gjør i hovedsak det samme for konfigurasjon, men er litt forskjellige når det kommer til arkitektur og funksjonalitet. Hensikten med slike systemer er at de kan styre maskiner fra en sentral plass i en typisk klient-tjener-arkitektur, der tjeneren ofte blir kalt *master* eller lignende. De bruker ulike språk og syntaks for deklarering, har systemer for håndtering av konfigurasjonsdata, statiske data om maskiner, gruppering av maskiner, og distribuering av konfigurasjon.

Selv om slike CM-systemer kan ta seg av mange oppgaver, kan det likevel være behov for å kjøre vanlige script. Det er anbefalt å unngå dette så lenge det lar seg gjøre. I tillegg må det bemerkes at CM-systemene styrer bare det de er bedt om. Det vil si at områder på en maskin som et CM-system ikke styrer vil kunne bli inkonsistent.

Det finnes ulike måter å gruppere maskiner som i hovedsak kjører samme tjeneste. Et vanlig mønster er å gruppere disse i *roller* der en rolle beskriver tjenesten de kjører. En rolle kan ha flere underroller (web-rolle og databaserolle som sammen utgjør en tjeneste). En maskin kan i prinsippet ha flere roller, men det kan være lurt å strukturere dette fornuftig.

2.4.2 Praksis og arbeidsflyt

Som nevnt handler programmerbar infrastruktur om å ta i bruk prinsipper og metodikk fra programvareutvikling til styring av IT-infrastrukturer. Ett av hovedmålene med en strukturert programvareutviklingprosess er å sikre god kvalitet på det som produseres og mer effektiv bruk av tid i utvikling og administrering.

Alle endringer bør spores, noe et versjonskontrollsystem kan gjøre. Man trenger ikke bare versjonskontrollere kode, men også andre filer, script, dokumentasjon, og testkode. Testing av deklarasjonskoden kan også gjøres ved bruk av et system for *CI/CD*, som Jenkins, TravisCI, med flere.

Hvis deklarasjonskode for CM-systemet skal testes, kan en *testing pipeline* benyttes, slik at flere ulike automatiserte tester kan gjennomføres før den er godkjent til å settes i produksjon. En pipeline kan teste syntaks, enhetstesting og utrulling i et testmiljø. Da kan man i størst mulig grad kunne verifisere lesbarhet, krav, og om det faktisk fungerer når det blir provisjonert på server-maskiner.

3 Kvalifisering, utvalg og vurdering

3.1 Beskrivelse av utvalgsprosess

Kandidatene til denne utvalgsprosessen tok utgangspunkt i Wikipedia sin oversikt over CM-systemer med åpen kildekode, som listet opp 23 forskjellige systemer [84]. I tillegg ble det brukt nettsøk til å finne eventuelle andre systemer som ikke var nevnt i artikkelen. Dette ga én ekstra kandidat og til sammen ble 24 kandidater tatt med videre i utvalgsprosessen. Se vedlegg B.1 for den komplette listen.

Kriteriene for utvalgsprosessen baserte seg på oppdragsgivers kriterier og gruppens egen kravspesifikasjon til et CM-system (se A.1).

3.2 Initielt utvalg

3.2.1 Utgivelsesdato

Dato for CM-systemets seneste utgivelse var første punkt som ble vurdert. Denne informasjonen ble hentet fra systemets offisielle nettside. Hvis informasjonen ikke var tilgjengelig på offisiell nettside ble informasjonen hentet der koden var utgitt, eksempelvis GitHub eller SourceForge. Kriteriet for å gå videre var at det hadde vært en utgivelse i løpet av siste året (dvs. nyere enn 06.02.2017). Basert på på utgivelsesdato ble det tatt ut 9 av 24 systemer, dvs. at 15 kandidater fikk være med videre. Se vedlegg B.2 for liste sortert på siste utgivelsesdato.

3.2.2 Trender blant brukere

Neste steg gikk ut på å se på hvor mye oppmerksomhet hvert produkt har fått på Internett, i form av tagger på StackOverflow, vedlegg B.3.1, stjerner på GitHub, vedlegg B.3.2, og søkemengde/søketrend på Google, vedlegg B.3.3. Tallene fra hver kategori ble plassert i et regneark og sortert fra høy til lav. Systemet med høyest tall fikk poengsummen 15, nummer to fikk 14, osv, ned til den med lavest poengsum som fikk kun ett poeng. Hvis det ikke var mulig å finne en verdi i en gitt kategori i det hele tatt ble poengsummen satt til null. Poengsummen for hver kategori ble summert og kandidatene ble sortert etter total poengsum fra høy til lav.

Alene er ingen av disse tallene hundre prosent pålitelig. Eksempelvis når det kommer til søketrender, der den prosentvise endringen for små (og lite søkt etter) systemer vil være unaturlig store kontra større systemer med høy total søkemengde. Når alle blir slått sammen kan likevel en tydelig trend sees. Den totale poengsummen for denne runden ligger i vedlegg B.3.4.

3.2.3 Kravspesifikasjon

Etter de to innledende rundene ble det gått litt mer i dybden på hvert system ved å lese offisielle beskrivelser og dokumentasjon for å se se hvorvidt de oppfylte hovedkravene til et CM-system. Hovedkravene gruppa definert var følgende:

- automatisk konfigurering av maskiner

- idempotent operasjon
- skalerer til flere maskiner
- konfigurasjon i tekstfiler
- scriptbart interface
- tilsynsløs operasjon

Basert på disse kravene ble utvalget redusert til seks kandidater (se tabell i vedlegg B.4).

3.2.4 Resultat av initielt utvalg

Resultatet av det initielle utvalget ble basert på prosjektgruppens kapasitet i forhold til tid. Vurderingen som ble gjort var at hvert gruppemedlem kunne gå i dybden på en kandidat hver. Ansvaret for Salt, Chef og Ansible ble fordelt på medlemmene i gruppa, mens Puppet ble delt, siden dette var et system alle hadde litt kjennskap til. Derfor kunne det være fire kandidater som ble vurdert nærmere. Disse fire kandidatene ble da plukket fra topp fire på brukertrender-listen og prosjektgruppen satt igjen med følgende fire (i alfabetisk rekkefølge):

- Ansible
- Chef
- Puppet
- Salt

De følgende delene gir en overordnet beskrivelse av hver av de, basert på vurderingskriteriene i vedlegg A.2.

3.3 Ansible

Ansible ble opprinnelig utviklet av Michael DeHaan [104]. Før Ansible jobbet han bl.a for RedHat og Puppet. Ansible startet i februar-mars i 2012 og er dermed den yngste av de fire kandidatene [104] [2]. Ansible Inc. (tidligere AnsibleWorks) ble overtatt av RedHat i oktober 2015 [173]. Formålet med Ansible har alltid vært at det skal være enkelt å bruke og det kommer med "batterier inkludert" som de selv sier [36]. Ordet "ansible" er for øvrig i litteraturen en oppdiktet kommunikasjonsgjenstand som tilbyr øyeblikkelig, raskere enn lyset, kommunikasjon [202].

Ansible tilbyr en kommersiell løsning av Ansible i form av Ansible Engine og Ansible Tower [12] [27]. Tower er et GUI-styringsverktøy for Ansible som bl.a gjør det mulig å tidsplanlegge når Ansible skal kjøre. Det ble ganske nylig (september 2017) gjort til et offisielt åpen-kildekode-prosjekt kalt *AWX project* [174]. AWX er støttet av RedHat og er en "upstream" til Tower [33].

3.3.1 Brukersamfunn

På hjemmesiden til Ansible finnes det ulike typer informasjon rundt produktet [31]. Artikler og videoer om hvordan Ansible fungerer (og hvorfor man bør velge Ansible) gir en rask og lettforståelig innføring til produktet. Naturlig nok ønsker RedHat å fronte de kommersielle entreprise-løsningene, derfor omhandler mye info på hjemmesiden nettopp dette. Velger man fanen *Community* får man opp mer info om åpen kildekode-delen av Ansible, som f.eks link til Ansible og AWX på GitHub [17] [33]. Under *Resources* fanen

ligger det lenker til dokumentasjon og videoer [25] [10].

Brukerutviklet og gjenbrukbar kode, kalles i Ansible for roller (*role*). Disse er tilgjengelig gjennom Ansible Galaxy [14]. I forhold til Puppet Forge er det mindre aktivitet, dvs. færre nedlastinger, men Galaxy er også ferskere enn Forge [149]. Galaxy ble tilgjengelig i Beta i desember 2013 [103]. Det er heller ikke slik at det finnes offisielle roller fra Ansible selv, slik Puppet tilbyr offisielle moduler i Forge.

Ansibles blogg tilbyr interessante artikler, både om nyheter og kommende funksjoner, samt opplæringsartikler (eks. "Getting Started") [7].

3.3.2 Brukere av Ansible

Tallene samlet inn i den initielle vurderingsfasen viste at Ansible virker å være et mye brukt system. Ansible brukes av mange brukergrupper, både utviklere og systemadministratorer (DevOps). På Ansibles hjemmeside listes det opp flere ulike bruksområder, deriblant: provisjonering, konfigurasjonsstyring, utrulling av applikasjoner og orkestrering [53]. I forhold til de andre kandidatene er Ansible mye mer enn bare et rent CM-system. I følge RightScales *State of the Cloud*-rapport for 2018 er Ansible mye brukt til konfigurasjonsstyring, både blant store bedrifter og spesielt blant de små/mellomstore, der Ansible er det mest brukte systemet blant de som svarte i sistnevnte kategori [179].

3.3.3 Dokumentasjon og opplæring

Ansible har omfattende dokumentasjon på den offisielle nettsiden [10]. På nettsidene ligger det en enkel *quick start-guide* med tekst og video, en del korte introduksjonsvideoer samt "webinars" som er lengre videoer som går mer i detaljer [30]. I tillegg ligger det ute videoer fra foredragene på konferansen AnsibleFest.

Kursing og konsulenttjenester finnes også via Redhats entreprise-løsning. Kursene tilbys både som online, klasserom, on-site produkter etc., men disse koster selvfølgelig en del å benytte seg av [32]. Redpill Linpro holder kursene i Norge [100].

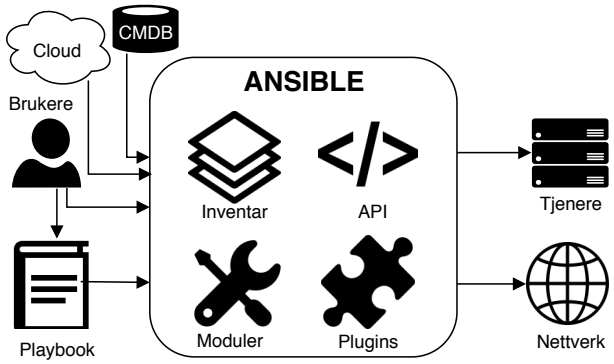
E-bøker (og videoer) er også tilgjengelig på nett, f.eks via PacktPub [112].

Ansibles runtime-dokumentasjon er også god. Kommandoen `ansible -h` lister tilgjengelige opsjoner. Man-sidene *man ansible* viser den samme hjelpemenyen, bare i en litt annen rekkefølge.

For de andre Ansible-kommandoene, som f.eks *ansible-playbook* og *ansible-galaxy* etc, får man opp tilsvarende dokumentasjon på samme måte. For dokumentasjon om en bestemt modul kan man enten finne det i Ansibles dokumentasjon på nettsiden, eller man kan bruke kommandoen `ansible-doc`, eksempelvis `ansible-doc ping` som viser dokumentasjonen for ping-modulen. `Ansible-doc -list` lister opp alle tilgjengelige moduler.

3.3.4 Utvikling

Da Ansible gikk fra versjon 1 til 2 skrev de at et av målene var at playbook-kjøringer skulle være bakoverkompatible [93]. Det skulle vistnok være oppfylt, men enkelte funksjoner kunne være inkompatible. Hvis en funksjon er "deprecated", altså foreldet, så vil Ansible alltid vente to major-utgivelser før funksjonen fjernes [3]. Da har man litt tid på seg for å gjøre eventuell refactoring av kode. En "deprecated" funksjon vil vise en advarsel når en playbook kjøres. Ansible har også egne *porting guides* som viser hvilke endringer som må gjøres for å være kompatibel med nyere versjoner av programvaren [11].



Figur 1: Ansibles arkitektur slik de selv presenterer det i sin Getting started-video på nettsiden [16] (figuren er reprodusert av rapportens forfattere)

Oversikt over utgivelser finnes både på GitHub og på Ansibles egen release-side [24]. Versjon 2.5.2 er på skrivende stund siste *stable version*. Det går ca. et halvt år mellom hver *major release*, men det heter seg at det er en "fleksibel" 4 måneders syklus [45].

- Ansible 2.5.0.0: 23.03.2018
- Ansible 2.4.0.0: 19.09.2017
- Ansible 2.3.0.0: 12.04.2017

3.3.5 Arkitektur

Ansible er agentløs [40]. Det betyr at alt initieres fra kontroll-maskinen (eng: *controller*) hvor Ansible kjører (altså en "push"-metode). Tre hovedfaktorer må være oppfylt for å bruke Ansible [40].

1. SSH-nøkkelpar
2. Målmaskinen definert i inventarfilen
3. Python på målmaskinen [13]

Maskinene som styres av Ansible trenger ingen ekstra programvare installert, med unntak av Python. Kommunikasjonen fra kontroll-maskinen til mål-maskinen foregår over SSH. Det må derfor eksistere et nøkkelpar mellom kontrollmaskinen og målmaskinen (selv om passord også er støttet) [40]. Ansible må også vite om maskinene den skal styre. Dette defineres i inventar-filen [40].

3.3.6 Viktige komponenter

Modul

Modulerer (eng: *module*) brukes til å oppnå den ønskede tilstanden på enhetene du automatiserer via Ansible og det finnes langt over 1000 inkludert i Ansible-installasjonen [40] [4]. En modul er små Python-progammer [20]. Eksempler på hva moduler kan styre er pakker, filer, brukere, brannmurregler, databaser osv. på en Linux-maskin, men også nettverksutstyr som ruter/switch. En del moduler er også ment for interaksjon med cloud-tjenester, eks. AWS, Azure med flere.

Inventarfilen

Definerer hvilke maskiner Ansible skal styre [40] [57]. Inventarfilen er i utgangspunket en statisk INI-fil (evt. YAML), men den kan også være dynamisk ved at et script henter den nødvendige informasjonen fra skytjenester eller en CMBD [56]. Inventarfilen gir også muligheten til å gruppere maskiner i (egendefinerte) kategorier, eks. etter tjenestetype.

Playbook

En *playbooks* hovedfunksjon er å knytte sammen *hva* som gjøres *hvor* [48]. En playbook består av et (eller flere) play(s). Et play angir hvilken gruppe maskiner som skal styres. Et *play* kan bestå av *tasks* (oppgaver) eller inkludere roller som grupperer tasks sammen. En *task* tilkaller en module som utfører oppgaven den er satt til. *Tasks* kjører sekvensielt i den rekkefølgen de er beskrevet.

YAML og Jinja2

Ansible bruker YAML-formatet til å beskrive playbooks og roller. For det som ikke kan beskrives ved hjelp av YAML, brukes *Jinja2 (templating system)*, eksempelvis til å referere til en variabel [54].

Rolle

Det å definere alle oppgaver direkte i hver eneste playbook blir tungvindt og lite skalerbart. En Ansible rolle er en måte å gruppere og gjenbruke oppgaver på [38]. Roller skaper et ekstra abstraksjonslag og gjør at playbooks blir "ryddigere"[99]. En rolle er i sin enkleste form bare en liste med *tasks*, men kan også inneholde variabler, templates, tester m.m.

3.3.7 Jobbe med Ansible

Det er hovedsaklig to måter å jobbe med Ansible på. Ad-hoc-måten og kjøring av playbooks. Den enkleste måten er ad-hoc-måten der man beskriver hva som skal gjøres på en kommandolinje (remote execution). Dette kan være praktisk under testing, men ikke noe man vil bruke til konfigurasjonsstyring og automatisering. Da skriver man heller *playbooks*. Eksempelet under viser en enkel playbook som tar i bruk en rolle som installerer pakken *cowsay*.

Listing 3.1: Enkel Ansible playbook

```
# cowsay-installasjon.yml
---
- hosts: cowsay
  become: yes
  roles:
  - cowsay
```

Hosts angir hvilke gruppe av maskiner som er målet, dvs at i *inventory*-filen finnes det en gruppe som heter *cowsay*. *Become* angir at det skal eskaleres til *sudo* (som er nødvendig for å installere pakker). I dette eksemplet inneholder *playbook*'en ingen oppgaver (*tasks*). Disse er blitt skilt ut i en egen *rolle*, derfor er det eneste som gjøres å inkludere rollen *cowsay*.

Listing 3.2: Enkel rolle

```
# roles/cowsay/tasks/main.yml
```

```
---
tasks:
 - name: Installasjon av Cowsay
   package:
     name: cowsay
     state: present
```

Rollen cowsay er så enkel som det går an å gjøre en rolle, dvs. bare en liste med oppgaver (*tasks*). Det en oppgave gjør er vanligvis å ta i bruk en modul til å utføre noe på målmaskinen. Oppgaver kjøres sekvensielt, altså i den rekkefølgen de er beskrevet. *Tasks* angir at her kommer det en liste med oppgaver. *Name* angir navnet til det første (og i dette tilfellet eneste) *task*. Strengt talt behøver ikke navnet være angitt, men det er "best practice" å gjøre det og det vil vises i terminal-outputen når playbooken kjøres [6]. Oppgaven kaller på *package*-modulen og ber om at pakken *cowsay* skal være installert *present*.

Playbooken kjøres med kommandoen `ansible-playbook cowsay-installasjon.yml`. Dette illustrerer det litt spesielle med Ansible kontra de andre systemene. Ansible *playboks* kjører ikke av seg selv. Skal det skje automatisk må man for eksempel lage *cronjobs* eller bruke *Ansible Tower/AWX* (eller andre lignende tredjepartsverktøy) til å tidsplanlegge og automatisere kjøringene.

3.3.8 Sikkerhet

Ansible bruker SSH som transportprotokoll, dvs at autentisering og kryptering mellom kontroll-maskin og mål-maskin er tilsvarende som for SSH [46]. Forenklet forklart brukes asymmetrisk kryptografi, privat/offentlig-nøkkelpar, til å til å utveksle en symmetrisk nøkkel som krypterer kommunikasjonen mellom klient/tjener [110].

3.4 Puppet

Puppet, som også er navnet på selskapet bak programvaren, har eksistert siden 2005 [138]. Ifølge deres egen beskrivelse av selskapet, har så mange som 40000 selskaper tatt i bruk åpen kildekode-versjonen og Puppet Enterprise. De beskriver Puppet som en industristandard. Det kan antas at mange CM-systemer lansert etter 2005 er inspirert av Puppet, og at mange derfor har lagd sine litt annerledes eller mer spesialiert, med samme prinsipper i bunn.

Puppet Enterprise (PE) er den kommersielle versjonen av Puppet [146]. Dette er nok Puppets viktigste produkt, men de tilbyr også en del andre mindre støttesystemer (*Puppet Discovery*, *Puppet Pipelines*, og noen andre åpen kildekode-produkter, eks. *Bolt*) [155]. De beskriver Enterprise som et altomfattende system for konfigurasjonsstyring ("enforce desired state") og automatisering av oppgaver. I tillegg med Enterprise får man et grafisk grensesnitt for overvåkning av Puppet, styring og visualisering av arbeidsflyt.

3.4.1 Brukersamfunn

Puppet tilbyr et økosystem for å jobbe med systemet. *Puppet Forge* er en tjeneste de tilbyr, der brukere av Puppet kan laste ned og opp ferdigskreven Puppet-kode (moduler) [149]. De har en egen spørsmålsside hvis man trenger hjelp [166]. På StackOverflow er Puppet blant de mest diskuterte av CM-systemene som denne rapporten omtaler, men slått av Chef og Ansible i antall tagginger. Deres egen spørsmålsside er nok uten tvil den mest brukte for Puppet-spørsmål, med over 6000 stilte spørsmål (per april 2018). Det fins også

andre kommunikasjonskanaler brukere kan kommunisere via, eksempelvis IRC [143].

3.4.2 Brukere av Puppet

Ifølge en artikkel, som siterer grunnleggeren av Puppet; Luke Kaines, så skal Puppet ha blitt grunnlagt og programvaren lansert på grunn av et ønske om å styre store infrastrukturer. Google, Amazon og andre store aktører gjorde dette allerede med sine infrastrukturer gjennom egenutviklede systemer [62]. Brukere av Puppet kan antas å være de som styrer større infrastrukturer, og som trenger en modellbasert måte å styre denne på.

3.4.3 Dokumentasjon og opplæring

På deres nettsider tilbyr Puppet en omfattende dokumentasjon av alle systemer de er ansvarlige for [145]. Dokumentasjonssidene inneholder både dokumentasjon for Puppet Enterprise og åpen kildekode-versjonen av Puppet. Sistnevnte er mest aktuelt å bruke, men det finnes også en del dokumentasjon for PE som også kan være nyttig og fungere for åpen kildekode-versjonen. Per april 2018 er Puppets versjonsnummer 5.5 og den gjeldene dokumentasjonen finnes i referansemanualen for akkurat dette versjonsnummeret [142]. Puppet oppdaterer altså dokumentasjonen for hver versjon, og man kan se tilbake på eldre/arkivert dokumentasjon for en gitt funksjonalitet.

Hvis man vil komme igang med å lære seg bruken av Puppet, så har de etablert et eget program ved bruk av en virtuell maskin man installerer på sin PC [153]. Bøker er fine å bruke for eksempler og beskrivelser av praksis, og for Puppet finnes mange bøker å velge mellom. Forlaget Packtpub[1] selger flere bøker for grunnleggende og avansert bruk av Puppet. For denne rapporten ble *Puppet 5 Essentials* brukt for å lære hvordan starte å bruke Puppet [1].

Ved bruk av Puppet på kommandolinjen kan argumentet `--help` beskrive funksjonaliteten for nesten alle valg mens man jobber med systemet.

Puppet har også en YouTube-kanal med mange nyttige forklaringer og diskusjon [139]. Mye der er filmet fra PuppetConf, en årlig offisiell konferanse for brukere av Puppet. I tillegg har de en egen side med deres egne ressurser for å være involvert i bruken av Puppet [169].

3.4.4 Utvikling

Utviklingen av Puppet skjer stadig vekk. Versjonsnotasjonen for åpen kildekode-versjonen er semantisk [137]

- **X:** Funksjonalitet endres radikalt eller fjernes
- **Y:** Funksjonalitet legges til
- **Z:** Bugfiks

Det er ingen spesiell sammenheng mellom major-versjoner og inkompatibilitet, annet enn det som blir nevnt i utgivelsesnotatene. Det kan altså være verdt å følge med på dette, samt utviklingen i GitHub-prosjektet de styrer [167]. Større inkompatibilitet blir likevel nevnt ettertrykkelig. Eksempelvis for versjon 3.x, som for lengst har nådd slutten av offisiell støtte fra Puppet, blant annet fordi det er store forskjeller mellom den og nåværende versjon 5 [140].

[1] https://www.packtpub.com/

3.4.5 Arkitektur

Puppets arkitektur er ganske enkel å forstå: Det er en klient-tjener-arkitektur. Eller som de selv kaller det: "The agent-master architecture" [136]. Puppet-serveren kjører et HTTP-API som agentene kobler seg til for å hente sin katalog (et begrep for den kompilerte lista over ting som skal gjøres). Puppet kan også kjøres i uavhengig modus, uten tilkobling mot en Puppet-server, hos klienten/agenten. Uavhengig kjøring av Puppet kan ha noen fordeler, men har likevel mange ulemper når det kommer til mer kompleks styring av systemer. For denne rapporten beskrives ikke annet enn klient-tjener-arkitekturen.

3.4.6 Viktige komponenter

Utover komponentene *master* (server) og *agent*, så har Puppet en hel del andre komponenter som fyller viktige oppgaver for å kunne gjøre konfigurasjonsstyring [117].

Puppet-språket

Språket i seg selv er en viktig del av hvordan Puppet fungerer. De har utviklet et eget modell-/klassebasert språk for deklarering av ressurser [124]. Språket har diverse logikk og semantikk som tillater ulik utføring. Språket skrives i filer kalt manifester med .pp-endingen og lagres i en spesiell katalogstruktur. Språket består i hovedsak av klasser, ressurser, variabler, logikk og funksjoner.

Moduler

Moduler er navnet på hvordan klasser er gruppert [144]. En modul er en samling av flere klasser/manifester, og som regel representerer som regel en tjeneste. Ferdigskrevne moduler for mye brukte tjenester finnes på Puppet Forge. Man kan skrive egne moduler for mer spesifikt oppsett av en tjeneste.

Facter

Såkalte *facts* er biter med statisk informasjon som kan beskrive ulike aspekter ved en maskin [121]. Eksempelvis vil Facter finne ut hvilket operativsystem agenten kjører og lagre dette. Disse bitene kan så brukes i Puppet-språket, eksempelvis for å skille mellom ulike operativsystem.

Hiera

Dette er navnet på Puppets undersystem for lagring av konfigurasjonsdata [115]. Ved å ha et separat system for å oppbevare konfigurasjonsdata, vil man kunne gjøre Puppet-koden mer gjenbrukbar. I Hiera oppbevares data som nøkkel-verdi-par i YAML. Navnet *Hiera* er en forkortelse av *hierarchy*, fordi Hiera er bygd opp som et hierarki, der høyere nivåer har høyere prioritet. Hvis en nøkkel er deklarert flere ganger, men med forskjellig verdi i ulike deler av hierarkiet, vil den lengst oppe bli valgt. Hovedfordelen med Hiera er altså separasjon av kode og data.

PuppetDB

Dette er et konfigurasjonsregister [156]. Det er ikke en CMDB, fordi Puppet server vil stadig oppdatere informasjonen i databasen. Den samler inn informasjon som er generert av Puppet for alle agenter. Blant annet samler den inn informasjon fra Facter, nyeste katalog for hver node, eller egendefinert informasjon. I Puppet-språket kan man hente ut eller legge inn disse dataene når en agent skal kjøre sin katalog. PuppetDB er strengt tatt

ikke nødvendig i et Puppet-oppsett, men den kan være nyttig for historikk, rapportering eller deling av statusinformasjon mellom noder.

3.4.7 Jobbe med Puppet

Deklarasjoner gjøres imanifester [124]. I disse manifestene deklarerer man først en overordnet klasse som andre ressurser deklareres under. En klasse kan simpelthen være navnet på en oppgave og dens enkelthandlinger som må utføres for å oppnå den ønskede tilstanden. Listing 3.3 viser et manifest som tar i bruk en variabel, en innebygd ressurs (*package*), og en annen modul (*NTP*).

Listing 3.3: Eksempel på manifest

```
class min_tjeneste {
  # Oppbevaring av NTP-servere i en array-variabel
  $ntp_servers = ['0.no.pool.ntp.org','1.no.pool.ntp.org']

  # Innebygd Puppet-ressurs for installasjon av pakker
  package { 'cowsay':
    ensure => present,
  }

  # Modul hentet fra Puppet Forge for styring av NTP
  class { 'ntp':
    servers => $ntp_servers,
  }
}
```

Hvis det er nødvendig med oppbevaring av konfigurasjonsdata i Hiera, kan `lookup()`-funksjonen brukes for å hente ut dette derfra.

3.4.8 Sikkerhet

Puppet benytter SSL-sertifikater for autentisering og kommunikasjon mellom Puppet server og Puppet agent skjer over TLS/HTTPS [164]. Når en agent kobler til serveren, vil den spørre om å motta et sertifikat. Serveren oppretter sertifikatet, men det blir i utgangspunktet ikke automatisk signert. Når noen/noe har signert det forespurte sertifikatet, vil det overleveres til agenten, og kommunikasjon og dataoverføring kan starte [1].

Puppet server fungerer altså som en sertifikatutsteder (CA) som oppbevarer privatnøkler, og bør derfor sikres svært godt.

3.5 Chef

Chef, utvikles av Chef inc og hadde første utgivelse i 2009 [80]. Hos Chef utvikles flere verktøy i tillegg til CM-systemet, deriblant: InSpec, Kitchen, Habitat. Både Chef-ansatte og frivillige utvikler disse for å forbedre brukeropplevelsen[83].

Chef Automate er den kommersielle versjonen av Chef. Selskapet reklamerer for at Chef Automate har raskere utrulling av programvare, og en mer effektiv arbeidsflyt og et webgrensesnitt med mulighet for overvåkning og visualisering[67].

3.5.1 Brukersamfunn

Ferdigskreven kode i Chef i form av *cookbooks*, kan som Puppet moduler i "Puppet Forge", lastes opp og ned fra *Chef Supermarket* hvor det ligger Cookbooks skrevet av både Chef og frivillige brukere.[83] Chef har et eget forum for brukerne sine, som er forholdsvis aktivt med nesten 8000 poster i Chef kategorien (per april 2018)[73]. Også på

StackOverflow er Chef mye diskutert med mange tagger. Andre brukersamfunn som blir tilbudt er blant annet: Meetups, ChefCon, Slack, Youtube, Facebook og Twitter. I tillegg finnes flere podcaster tilgjengelig som "Foodfight" og "Arrested devops". Selv om mange av dem ikke har publisert noe på lenge, er det fortsatt mye nyttig stoff som ligger ute. Foodfight alene har lagt ut over 100 podcaster [79].

3.5.2 Brukere av Chef

Chef er samarbeidspartner med mange selskaper. Eksempelvis partnerskapet med Azure, AWS og Google Cloud, som gjør det enklere å implementere Chef om man har en infrastruktur på disse platformene.[78] De har mange kunder som tar i bruk deres tjenester og lister opp noen suksesshistorier på sine sider. Noen av de større kundene er: Facebook, Gannett og Bank Hapoalim[70].

3.5.3 Dokumentasjon og opplæring

Chef har flere muligheter for opplæring og har enkelt tilgjengelig dokumentasjon. De har en egen læringsplatform[76] på hemmesidene sine. Der tilbyr de forskjellige opplæringsveier alt etter hvilken infrastruktur det ønskes å sette opp Chef på. Andre måter man kan lære Chef på er gjennom kurs som arrangeres jevnlig[69], eller bøker som er tilgjengelige, eksempelvis *Learning Chef* (S. Vargo, M. Taylor)[111].

Dokumentasjonssidene gir all nødvendig informasjon om Chef samt mulighet for å lete opp nødvendig referanse. "Getting started"-delen kan utføres for å få generell innsikt i hva Chef er og kan gjøre. Ellers finnes dokumentasjon for den åpne kildekode-versjonen, Chef Automate og de andre verktøyene som Chef utvikler.[72] Dokumentasjon fra tidligere Chef versjoner er tilgjengelige gjennom arkivet de har på dokumentasjonssidene sine, men den siste arkiverte versjonen er merkelig nok fra 2016[65].

Runtime-dokumentasjon er tilgjengelig for kommandoer som har flere underkommandoer. Manual-sider kan eksempelvis vises med: `knife help underkommando`. Mens for kommandoer som kjøres enkeltvis er det mulig å få informasjon fra `-h` eller `--help` parameterne. Dette gjelder kommandoene tilgjengelige i Chef Development kit[82].

3.5.4 Utvikling

Chef utvikler klient- og tjenerprogramvaren hver for seg. Begge er åpen kildekode. Utviklingen av Chef foregår på Github og går over flere milepæler:[74]

- **Master:** når kode legges til i hoved grenen. En slik endring øker det minst signifikante versjonstallet automatisk
- **Unstable:** når en pakke lages av versjonen og testes på de støttede OS'ene
- **Current:** om det tidligere leddet gikk igjennom alt av tester blir denne versjonen tilgjengelig for offentligheten
- **Stable:** Innimellom velger Chef en versjon som skal støttes over lengre tid. Stable versjoner er tilgjengelige i et eget repo.

For å se hvilke endringer som har skjedd og hvilke funksjoner som utgår må man følge med i utgivelsesnotatene [81]. Merk at disse er individuelle for "Chef server, client og Development kit".

3.5.5 Arkitektur

Chef har en klient-tjenerarkitektur, hvor konfigurasjonen ligger på tjeneren og klientene kjører periodisk, eller trigges av en hendelse for å hente sin konfigurasjon. Det er og en mulighet for at Chef kan konfigurere seg selv, men som regel brukes klient-tjener modellen[64]. Chef master bruker et HTTP-API som klientene kommuniserer med. Selve utviklingen av Cookbooks foregår i *Chef Development kit* som systemadministratorene enten har lokalt på sin maskin eller på en delt managermaskin for utvikling. Det er herifra Cookbooks sendes til serveren.

3.5.6 Viktige komponenter

Chef-språket

Chef er skrevet i Ruby, men har et eget domene spesifikt språk (DSL) for å skrive konfigurasjonsfiler. Tester og egenlagd funksjonalitet skrives i Ruby. Filer i Chef har .rb-endinger og lagres i en *Recipe*, som er en del av en Cookbook. Disse organiseres i en katalogstruktur som Chef genererer.

Cookbooks

Cookbooks er Chef sitt navn på organisering av konfigurasjonsfiler i ulike kataloger[68].

Ohai

Ohai er ekvivalent med Facter i Puppet og gjør det mulig å hente ut statisk info om de forskjellige maskinene, som nettverk, informasjon om operativsystem, CPU osv. Dermed kan forskjellige avgjørelser tas av Chef basert på denne informasjonen[77].

Databags

For å lagre konfigurasjonsdata data i Chef brukes noe kalles *Databags*.[71] Databags er globale variabler, og lagres som JSON-data. De er tilgjengelig fra en Chef-server, og er mulige å søke opp og puttes inn i en “Recipe”, basert på en indeks. Hvert enkelt data element kan krypteres med forskjellige nøkkler.

3.5.7 Jobbe med Chef

I Chef foregår utviklingen (som tidligere nevnt) i Chef Development kit. Her generer man Chef sin mappestruktur og skriver egne “recipes” som blir innholdet i Cookbooks. Alternativt kan man laste ned Cookbooks fra “Chef Supermarket” eller sine egne lokale repoer. [64]

Eksemplet i listing 3.4 viser hvordan installasjon av en pakke og styring av en tjeneste kan skrives. Samt periodisk kjøring av apt-update og opprettelse av en ny gruppe og bruker.

Listing 3.4: Eksempel på manifest

```
apt_update 'Update the apt cache daily' do
  frequency 86_400
  action :periodic
end

package 'apache2'

service 'apache2' do
  supports status: true
  action [:enable, :start]
```

```
end

group 'web_admin'

user 'web_admin' do
  group 'web_admin'
  system true
  shell '/bin/bash'
end
```

3.5.8 Sikkerhet

Autentisering

All kommunikasjon med Chef-masteren må være autentisert. Autentiseringsprosessen skal sørge for at Chef-serveren kun svarer på spørringer fra autentiserte brukere. Autentiseringen utføres med RSA public key kryptering, som gjør at klientene kjører `chef-validator` (eller `knife exec`) for å autentisere seg selv mot Chef master. Det autentiseres først med en default validator.pem. Deretter tildeles klientene en privat nøkkel av Chef master, mens serveren autentiserer dem med den offentlige nøkkelen.[66]

3.6 Salt

Salt er navnet på åpen kildekode-versjonen av programvaren som selskapet *SaltStack* tilbyr [180]. Den kommersielle versjonen av Salt heter *SaltStack Enterprise*, og skal kunne tilby kundestøtte og tilleggsfunksjonalitet. SaltStack beskriver Salt som et hendelsesorientert system bygd for skysystemer. Salt ble først lansert i 2011, den nest yngste etter Ansible fra utvalget utvalget i denne rapporten [192].

3.6.1 Brukersamfunn

I motsetning til alle de andre, har ikke SaltStack en egen ressursside for ferdigskreven kode. I stedet brukes en liste i GitHub over ferdigskrevne *Formulas* for ulike tjenester [191]. Kommunikasjon med andre Salt brukere kan foregå gjennom ulike kommunikasjonskanaler, blant annet IRC [192].

3.6.2 Brukere av Salt

I likhet med andre CM-systemer virker Salt som nok en solid kandidat. Salt er skrevet i Python [107]. Salt skal også være et svært raskt system, som kan være en fordel i miljøer med tusen og titalls tusen maskiner som skal styres. En fordel med Salt kan være at det inkluderer mye innebygd funksjonalitet.

3.6.3 Dokumentasjon og opplæring

SaltStack har dokumentasjonssider som forklarer ulike konsepter, men inkluderer også en omfattende dokumentasjon av all innebygd funksjonalitet [186] [189]. På samme hovedside for dokumentasjon finnes også en liste over bøker om Salt. Dokumentasjonen viser også én tidligere versjon for sammenligning mellom utgivelsene. I tillegg til dokumentasjonen, har SaltStack også en begynnerveiledning som i tilstrekkelig grad, og vel så det, forklarer ulike viktige aspekter ved Salt [185].

3.6.4 Utvikling

Salt benytter en datobasert versjonsnotasjon med år og måned, eksempelvis *2017.7.5*, der siste nummer er bugfix [193]. Med Salt er det ingen spesiell sammenheng mellom utgivelser og kompatibilitet mellom de. Er man bekymret for at funksjonalitet forsvinner eller endres mellom versjoner, bør man lese utgivelsesnotater og følge med på nye versjoner [187].

3.6.5 Arkitektur

Salts arkitektur er en salgs klient-tjener-arkitektur. Det har en *master* som styrer sine *minions*. I motsetning til eksempelvis Puppet, er det master-noden som bestemmer når en eller flere minions får sin konfigurasjon. Siden Puppet bruker et HTTP-API, vil de styrte nodene selv måtte spørre master-noden om sin konfigurasjon. I Salt sitt tilfelle vil masteren publisere en melding med konfigurasjon på en ZeroMQ-meldingsbus som minion henter og sørger for å iverksette sin konfigurasjon [181].

Arkitekturen er noe av det som skiller Salt fra andre CM-systemer. I utgangspunktet er Salt et *remote execution*-system [182]. Det vil si at Salt opprinnelig ble bygd for å kjøre kommandoer til mange maskiner fra en sentral plass. Ikke lenge etter ble *state*-systemet etablert rundt dette for å sørge for konfigurasjonsstyring [188]. Salt-arkitekturen og -systemet bygges i hovedsak rundt meldingsbusen på masteren. Undersystemer og plugins kommuniserer alle gjennom dette meldingssystemet. Dette meldingssystemet er sentralt i den "hendelsesorienterte" kjøringen av Salt.

3.6.6 Viktige komponenter

Salt har også mange av de samme støttesystemene som de andre kandidatene også har sine versjoner av. I tillegg til *master* og *minion* som egne komponenter, har Salt andre sentrale komponenter.

Remote execution-*systemet*

Salt ble som nevnt opprinnelig bygd for å sende kommandoer til andre maskiner på kommandolinjen [188]. Dette systemet er bygd opp av mange forskjellige funksjoner som kalles *moduler*. Et eksempel på en modul kan være en som oppretter en brukerkonto på en maskin.

State-*systemet*

Hvis man vil sørge for at resultatet fra en remote execution-modul skal vedvare permanent, bruker man State-systemet for å "wrappe" om en remote execution-modul [182]. Ved å legge på før- og ettertesting av tilstanden modulen vil produsere, vil man gjøre konfigurasjonsstyring og teste ønsket tilstand. *States* skrives i tekstfiler med YAML-format [184].

Formulas

En ferdigskreven tilstand (*state*) beskrives som en *forumula* og er nesten det samme som roles i Ansible, modules i Puppet eller cookbooks i Chef. En formula er altså en samling av flere state-filer, som igjen består av flere state-deklarasjoner [182].

Grains

Salt må også ha en måte å hente ut statisk informasjon om sine noder på. *Grains* brukes til dette og kjøres på en minion/styrt node og vil returnere ting som operativsystemnavn, IP-adresser, FQDN, også videre [183].

Pillar

Hvis states skal være gjenbrukbare, bør kode og data separeres. *Pillar* er Salts system for dette [183]. Det er strukturert på samme måte i kataloger som state-systemet i egne YAML-filer.

Salt Mine

Salt tilbyr også et konfigurasjonsregister med *Salt Mine* [183]. Har man informasjon om minions/styrte noder som stadig endrer seg, kan man bruke denne til å samle informasjonen hver minion melder inn om. Eksempelvis kan en dynamisk gruppe med servere publisere sin IP-adresse eller FQDN til Salt Mine, og tas i bruk av Salt til kjøring av states.

3.6.7 Jobbe med Salt

Når man bruker Salt er det to måter man kan interagere med minions/styrte noder: remote execution-systemet eller state-systemet. Førstnevnte sender kommandoer til en maskin eller en gruppe maskiner. State-systemet brukes gjennom tekstfiler skrevet i YAML, og plasseringen av disse tekstfilene i en katalogstruktur (som er viktig for oppslag og namespacing) [107].

Remote execution-systemet

```
salt '*' user.add alice
```

Listing 3.5: Eksempel på fjernstyring av maskin med Salt

```
salt '*' user.add alice
```

Listing 3.5 viser hvordan Salt kan sende en kommando til maskiner. *'*'* betyr at alle registrerte maskiner skal motta kommandoen. *user.add* betyr at en bruker *alice* skal legges til.

State-systemet

Listing 3.6: Eksempel på state for installasjon av pakke

```
install_cowsay:
  pkg.installed:
    - name: cowsay
```

Listing 3.6 viser hvordan tilstanden til en pakke deklareres i en state-fil. Ordet endres fra remote execution-systemet over til state-systemet. Eksempelvis vil en kommando for å installere en pakke i remote execution-systemet hete `package.install`.

State-systemet beskriver en ønsket tilstand. Remote execution-systemet beskriver en oppgave som skal utføres.

3.6.8 Sikkerhet

Autentiseringen og utveksling av symmetriske nøkler mellom master og minion er nøkkelbasert med offentlige og private nøkler [181]. Før kommunikasjonen har blitt etablert, vil minion starte en handshake-prosess og sende over sin public-nøkkel til masteren.

Denne nøkkelen oppbevares på master fram til den blir godkjent av administratoren ved hjelp av salt-key-kommandoen. For hver kjøring vil AES-nøkler roteres ved kryptering av data. Pillar-data krypteres med den enkelte minion sin offentlige nøkkel ved overføring på nettverket.

3.7 Valg av kandidat for eksempeloppsett

Den individuelle vurderingen av kandidatene viste at alle fire er gode produkter. Forbeholdet her er at det er begrenset hvor godt en kan lære seg fire komplekse systemer i løpet av den begrensede tiden som som var til rådighet. Derfor blir valget av kandidat for eksempeloppsett tatt ut i fra forutsetningen om at alle detaljer rundt systemene ikke er 100% kjent. Det er heller ikke sånn at et system er best på alle områder. Alle har sine styrker og svakheter. Valg av kandidat handler mye om å veie de ulike styrkene og svakhetene opp mot oppdragsgivers behov. Det blir også et spørsmål om hvor mye dagens arbeidsflyt skal kunne videreføres med det nye systemet, kontra hvilket system gir en hensiktsmessig arbeidsflyt i fremtiden.

Alle systemene oppfyller nesten alle krav fra kravspesifikasjonen og vil med stor sannsynlighet imøtekomme oppdragsgivers behov. Det eneste kravet som ikke ble møtt for alle var tilgangsstyring ved bruk av f.eks LDAP. Puppet støtter ikke dette uten å betale for kommersiell versjon. Ansible må bruke AWX/Tower. Tilgangsstyring kan gjøres på andre måter, og det ble besluttet at disse kandidatene ikke skal ekskluderes bare på grunn av denne mangelen. Det er også usikkert hvorvidt Chef og Salt sine implementasjoner av LDAP fungerer.

3.7.1 Resultater fra individuell vurdering

Tabellen i vedlegg C.2 viser resultatene fra den individuelle vurderingen. Det som er tydelig av tabellen er at kandidatene stort sett oppfyller alle krav. Det som skiller de er i hovedsak arkitekturmessige aspekter, som skalerbarhet, hvordan konfigurasjoner synkroniseres, sikkerhetsmekanismer, og forskjell i deklarasjonsspråk.

3.7.2 Karakteristikk ved de ulike kandidatene

Som nevnt over gir ikke noen ukers dypdykk ekspertkompetanse i hvert system, men kjennskapen til de fire kandidatene er god nok til at særtrekkene og styrker og svakheter kort kan oppsummeres. I vedlegg C.1 listes alle fordelene og ulempene prosjektgruppen mente var vesentlige for hver kandidat.

3.7.3 Vurdering opp mot oppdragsgivers behov

Oppdragsgiver har oppgitt noen punkter for vurdering i den opprinnelige oppgavebeskrivelsen, samt noen punkter i samtaler med prosjektgruppen.

Arbeidsflyt

Arbeidsflyt er i hovedsak lik mellom kandidatene, bortsett kanskje fra Chef som har et eget utviklingsverktøy. Alle tillater å bruke versjonskontroll i f.eks Git, noe også oppdragsgiver er kjent med. Alle har i større eller mindre grad mulighet for å teste og validere definisjonskode. Uansett hvilket system som velges, må en lære dets deklarasjonsspråk.

Skalerbarhet

På forsidene sine skryter de forskjellige CM-systemene av hvor mange noder de kan konfigurere fra en tjener, men disse tallene kan være basert på ulike faktorer, siden forskjellige maskiner kan ha blitt brukt i scenarioene. Istedet hentet vi ut antall noder som kan konfigureres under anbefalt mengde RAM og økte denne til 8GB for å få et bilde av hvor mange noder en slik maskin kan ha under seg med de forskjellige CM-systemene:

- **Chef:** Ingen klar info angående RAM (hevder at en Chef server skal kunne takle flere tusen noder)[75]
- **Ansible**: 200. Ansible har ingen klare tall selv, men med AWX/Tower er det 100 hosts på en 4GB master [28]
- **Puppet:** Viser ikke til noe konkret tall men er brukt i storskala miljøer. Standard størrelsen på Puppet server er 2GB RAM, for å ta ibruk mer må dette endres i JVM [154]
- **Salt:** 500-1000. 8GB RAM er også anbefalt minimum for en salt master [190]

Merk at disse bare er omtrentlige tall, ettersom flere andre faktorer som hvor mye som skal konfigureres, hvor lang tid man trenger mellom hver endring, antall CPU kjerner og at de forskjellige CM-systemene kan ha tatt i bruk forskjellige kriterier for å komme til disse tallene. Antakeligvis styrer ikke oppdragsgiver mange nok maskiner til at dette kan bli en stor problemstilling.

Forutsigbarhet for funksjonalitet i utgivelser

Alle de fire kandidatene har kommersielle selskaper i ryggen, så utvikling skjer ikke bare på frivillig basis. Å si noe om forutsigbarhet for funksjonalitet i utgivelser og bakoverkompatibilitet er svært vanskelig. Det er uansett viktig å lese notater for hver nye utgivelse og se hva som eventuelt krever fornying.

Oppgradering av systemet

Alle kandidatene har hyppige utgivelser. Ansible har her en klar fordel ved at ingen klientprogramvare er påkrevd og bare masteren krever vedlikehold (bortsett fra eventuelle Python-avhengigheter på styrt maskin). For de agent-baserte systemene er det en fordel om server og agent utgis i samme versjon. Dette kan være en ulempe med Chef, som har ulike versjoner mellom klient og tjener.

Synkroniseringsmodell (push vs. pull)

Ett av aspektene som skiller kandidatene er modellen for synkronisering av konfigurasjoner. Ansible og Salt bruker “push” (tjeneren bestemmer når klientene henter sin konfigurasjon), mens Chef og Puppet bruker “pull” (klientene initierer selv kommunikasjon med tjeneren for å hente sin konfigurasjon). Selv om Salt bruker push, trenger ikke klienten åpne en port, siden den uansett initierer tilkoblingen mot tjeneren og venter på en melding. I Ansibles tilfelle må en port for SSH åpnes hos klienten.

Sikkerhet

Alle kandidatene, bortsett fra Ansible, må kjøres som root-bruker. Det som kan kreve root-tilgang er oppgavene som utføres hos den styrte maskinen. Dette kan enkelt syres med et flagg i konfigurasjonen. Alle kandidatene sikrer og autentiserer overføring av informasjon på nettverket ved hjelp av kryptografisk teknologi.

Tilgangskontroll, det vil si begrensing av tilgang til CM-programvaren, støttes av Chef og Salt (slik prosjektgruppen kunne forstå dokumentasjonen). Puppet tillater dette gjennom Puppet Enterprise, og Ansible tillater gjennom AWX/Tower.

Sikring av sensitive konfigurasjonsdata gjøres litt forskjellig for alle kandidatene. Eksterne systemer, som HashiCorps Vault kan oppbevare og sikre slike data og benyttes av de ulike CM-systemene [91].

Organisatorisk sikring er like viktig som de tekniske implementasjonene for sikkerhet. Rutiner og retningslinjer for sikring av maskiner bør også gjelde maskinene involvert i CM-systemet.

Brukervennlighet

Det er stor variasjon i hvordan kandidatene oppleves å arbeide med. Ansible er ment å være simpel og har mange muligheter for bruk. Siden Ansible ikke trenger agenter på styrte maskiner, kan det enkelt kjøres rett fra brukerens personlige maskin. De andre kandidatene krever større oppsett med dedikerte tjener- og klientprogramvare.

Moment i markedet

Ifølge våre tall, spesielt søketrenden fra Google, viser at Ansible er klart mest omtalt. Rightscale har publisert rapporter som presenterer tall fra bedrifter i "skyindustrien", blant annet med tall om bruk av CM-systemer. I 2014 var Chef og Puppet de to mest brukte systemene [175]. Fra 2015 og inn i 2016 ble flere CM-systemer tatt med i vurderingen, deriblant Salt og Ansible, der sistnevnte allerede var større enn Salt [176][177]. Chef og Puppet var på det tidspunktet ennå de to største. I 2017 begynner Ansible å nå Chef og Puppet i antall brukere, men er altså ennå på tredjeplass [178]. De nyeste tallene, fra 2018, viser at Ansible har tatt igjen Chef og Puppet i antall brukere [179].

Rightscales undersøkelser har hatt rundt 1000 respondenter (bedrifter) alle år.

Sidespor: Containere

Containere er et alternativ til styring og oppsett av tjenester. Siden Docker lanserte i 2013 har det ikke kommet mange nye CM-systemer[84]. Framveksten av container-teknologi er ikke nødvendigvis årsaken til dette. Ifølge State of the cloud-rapporten av Rightscale har Dockers containerteknologi hatt sterk vekst. Denne undersøkelsen fokuserte på statistikk i cloud-systemer. I deres rapport fra 2017, under "Respondents Using DevOps Tools", var det 35 prosent av respondentene som svarte at de brukte Docker og 32 prosent ønsket å ta det ibruk. Men på samme tid var det og voksende interesse for CM-systemer, da spesielt Ansible. Chef og Puppet var like etter, der omtrent 16 prosent ønsket å ta ibruk hver av de og 21-28 prosent allerede bruker én av disse [178].

3.7.4 Møte i Trondheim

For oppdragsgiver ble det holdt en kort presentasjon av de 4 CM-systemene vi mente egnet seg best til deres implementasjon. Dette møtet fant sted på NTNU campus Gløshaugen i Trondheim. Det ble presentert forskjeller i syntaks, arbeidsflyt, arkitektur og andre funksjonaliteter i henhold til kravene.

Etter presentasjonen av CM-systemene uttrykte oppdragsgiver interesse for Ansible og Puppet. Dette var også kandidatene som kom best ut i første runde av utvalget (se B.3).

3.7.5 Valgte kandidater

Alle kandidatene vil grunnleggende kunne løse oppdragsgivers arbeid med konfigurasjon av Linux-maskiner. Det største argumentet for å velge Ansible og Puppet var oppdragsgivers ønske om å fokusere på disse. Likevel kan ytterligere argumentasjon underbygge dette valget.

Som nevnt er Ansible enkel å starte med og bruke. For infrastrukturer med noen hundre maskiner, vil synkronisering av konfigurasjoner gå fint. Ansible er nå også, som nevnt, sterk vekst i IT-bransjen ellers i verden [179].

Puppet er den eldste kandidaten, og har tydeligere mønstre for bruk. Kvaliteten på moduler i Puppet virker også av høyere kvalitet, spesielt siden Puppet selv har skrevet mange av de mest brukte. Puppet er allerede i bruk ved NTNU, blant annet for å drifte OpenStack-løsningen SkyHiGh i Gjøvik. Ved at flere tar i bruk Puppet i IT-miljøer ved universitetet, kan kunnskap og kode deles mellom de.

Chef og Puppet har veldig like tilnærminger til konfigurasjonsstyring. Det som har vippet Puppet over Chef har vært de bedre resultatene i første runde, ønske fra oppdragsgiver og at de i Trondheim og Gjøvik har flere med kompetanse på Puppet enn Chef. Puppet og Ansible er kanskje også mer interessant å sammenligne, siden de har noen svært vesentlige forskjeller (arkitektur, “brukervennlighet”, begrensninger). Salt fikk mindre oppmerksomhet, blant prosjektgruppa og oppdragsgiver, samt at den er minst brukt og omtalt ellers i verden.

Ansible og Puppet blir altså tatt med videre til eksempeloppsettet. Neste kapittel Utvidet beskrivelse av Ansible og Puppet, vil gå i dybden for de tekniske aspektene ved disse to.

4 Utvidet beskrivelse av Ansible og Puppet

En kort beskrivelse av Ansible og Puppet ble gjort i utvalgsprosessen. I dette kapitlet vil det bli gitt en mer detaljert beskrivelse av te to kandidatene som ble tatt med videre til et eksempeloppsett.

4.1 Ansible

4.1.1 Roles

En Ansible-rolle er som tidligere nevnt en samling av gjenbrukbar, portabel kode og i sin aller enkleste form bare en liste av *tasks*, men en god rolle består gjerne av mer enn bare det. Standard rollestruktur vises i listing 4.1. Den ekleste måten å få en ferdig oppsatt rollestruktur er å bruke kommandoen `ansible-galaxy init rollenavn` [15]. (Da blir det også opprettet en mappe kalt *tests* og readme-fil i tillegg). Mappestrukturen er ganske selvforklarende og Ansibles dokumentasjon forklarer det også nærmere [47].

Tasks er allerede nevnt i del 3.3.7, *vars* separerer konstanter fra tasks og *default* definerer standardverdier for variabler. *Files* inneholde filer som kopieres som de er, mens *templates* er filer som blir tolket av *Jinja2* [49]. *Meta* inneholder metadata, eksempvis versjonsnummer eller avhengigheter av andre roller.

Handlers

En *handler* er i grunn det samme som et *task*, men kjøres bare hvis et *task* ber den om å gjøre det [48]. En typisk oppgave for en handler er å starte en tjeneste på nytt igjen for at eventuelle endringer skal tre i kraft. Hvis en f.eks har handleren "Restart Apache" kan et *task* varsle handleren med å si `notify: restart apache`. Handlers kjøres på slutten av et play og bare en gang, selv om flere tasks har varslet handleren.

Listing 4.1: Rolle-struktur

```
roles/
    my-role/
        defaults/
        files/
        handlers/
        meta/
        tasks/
        templates/
        vars/
```

4.1.2 Ansible-språket

Rapporten vil ikke forklare hvordan YAML fungerer, men enkelt sagt dreier det i Ansible seg om å lage lister, hvor hvert listeelement er en liste av *key/value-par* [60]. Det er imidlertid begrenset hvor mye en kan uttryke ved hjelp av YAML, derfor bruker Ansible også templating-systemet *Jinja2*[203] (som også Salt benytter) for å uttrykke mer komplekse ting [44]. En behøver ikke kunne alt om Jinja2, men bruk av variabler og logikk i Ansible benytter en del enkel Jinja2-syntaks.

Variabler og kontrollmekanismer

En variabel kan defineres på følgende måte:

```
brukernavn: foobar
```

Det vil altså si at nøkkelen *brukernavn* får verdien *foobar*:

For å referere til variabelen brukes Jinja2-syntaks sammen med YAML. Ved å bruke Ansibles *user*-modul (som oppretter brukere) som eksempel kan brukeren som opprettes få navnet som ble definert i variabelen *brukernavn*, altså *foobar* ved å si:

```
- name: Opprette bruker
  user:
    name: "{{ brukernavn }}"
```

Facts

Ansible henter automatisk informasjon om nodene, *facts*, når en playbook kjøres ved ved hjelp av *setup*-modulen [26]. Denne informasjonen kan også refereres til på samme måte som ved variabler, eks. ‘‘`{{ ansible_fqdn }}`’’. Det er også mulig å hente *facts* fra en bestemt maskin eller en gruppe av maskiner [13].

When

Ansible sitt *when*-uttrykk tilsvarer et *if*-uttrykk, altså gjør dette hvis en betingelse er oppfylt. Et vanlig bruksområde for dette er å gjøre noe basert på en nodes *fact*, eks. OS-familie eller -distribusjon vist i listing 4.2. Ved bruk av *when* behøver ikke det enkelte *facts* omsluttes av snutter og klammer.

Listing 4.2: Bruk av when for å installere pakke for en bestemt OS-familie

```
- name: Installer Httpd
  package:
    name: http
    state: present
  when: ansible_os_family == "RedHat"
```

Loop

Noen ganger er det ønskelig å gjøre flere ting på en gang i samme *task*. Eksempelvis installere flere pakker eller opprette flere brukere på en gang, uten å ha et eget *task* for hvert enkelt element. Da er det mulig å iterere over ulike datastrukturer ved å bruke *loop*-nøkkelordet, som i listing 4.3. Da brukes ‘‘`{{ item }}`’’ for å utrykke enkeltelementet. Tidligere har Ansible hatt mange forskjellige uttryk for å iterere avhengig av hvilken datastruktur, eks. *with_items*, *with_dict*, *with_nested* med flere, men fra versjon 2.5 er alle disse blitt erstattet av *loop* [19].

Det er også mulig å bruke en *query*-funksjon + *lookup*-plugin for å tilpasse hva som itereres over [42]. Dette kan f.eks brukes for å iterere over flere lister på en gang.

Listing 4.3: Iterering av liste

```
- name: Opprette brukere
  user:
    name: "{{ item }}"
  loop: "{{ liste_med_brukernavn }}"

- name: Installere pakker for Debian
```

```
        apt:
          name: "{{ item.pakkenavn }}"
          state: "{{ item.tilstand | default('present')}}"
        loop: "{{ liste_med_pakker }}"
```

Det må også nevnes at enkelte moduler, som f.eks *apt*- og *yum*-modulen også kan ta en liste som argument (for pakkenavn), uten å måtte bruke *loop*-konstruksjonen.

Plugins

Ansible bruker plugins for å utvide funksjonaliteten. Mange er allerede inkludert, men det er også mulig å skrive sine egne. Listing 4.3 viser i tillegg til iterering også et eksempel på hvordan en *filter*-plugin kan brukes til å angi en standardverdi hvis verdien ikke er definert. I dette tilfellet til å angi at brukerens tilstand vil være tilstede (present) hvis dette ikke er definert.

Ansible-språket oppsummert

Denne rapporten er ikke omfattende nok til å forklare alle aspekter rundt "Ansible-språket". Vil en vite mer står det meget godt forklart i den offisielle dokumentasjonen. Eksemplene som er nevnt over er eksempler på ting som er nyttig å vite (og som også er benyttet i eksempeloppsettet). Bruk av "Jinja2-magi" gjør gjerne koden mer kompakt, men det samme resultatet kan som regel oppnås uten, ved bruk av noen flere linjer. Ansible sin debug-modul er nyttig å bruke under testing for å enkelt sjekke at en henter og behandler variabler slik en faktisk ønsker [39].

Det som er tydelig ved å lese tidligere versjoner av Ansible-dokumentasjonen er at "Ansible-språket" er i konstant utvikling. For hver nye versjon er det gjerne lagt til små endringer og forbedringer.

4.1.3 Struktur på roller og playbooks

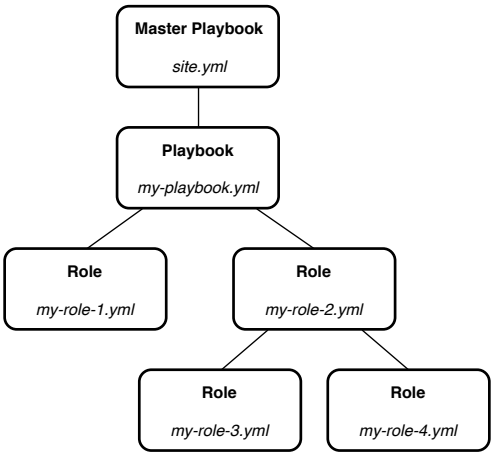
I Ansible er det mye opp til en selv hvordan en velger å organisere den hierarkiske strukturen. Et enkelt eksempel vises i figur 2. Eksempeloppsettet er hovedsaklig basert på best *practice*-prinsippene [6]. En *master playbook*, kalt *site.yml* inkluderer alle andre *playbooks*, men en kan også velge å kjøre de individuelle playbook, eksempelvis *webserver.yml* eller *databaserver.yml*, hver for seg. En annen mulighet er å bruke *limit*-opsjonen som gjør det mulig å begrense målet for en playbook-kjøring til en enkelt gruppe av maskiner eller enkeltmaskin. Som nevnt kan playbooks inkludere roller, men roller kan også inkludere andre roller, enten som avhengigheter (hvor disse rolle kjøres aller først), eller som en del av oppgavene (*task*-listen). I tillegg kan playbook også inkludere andre playbooks. [38]

Innad i en *playbook* er det også mulig å bruke mønstre og logikk for treffe et spesifikt servergruppe-mål ut i fra hvordan de er gruppert i *inventory*-filen [58].

4.1.4 Nodegruppering og -klassifisering

Som nevnt tidligere bruker Ansible *inventory*-filen (eller et dynamisk inventory-script) til å gruppere de ulikene serverne. Et utdrag fra inventory-filen i eksempeloppsettet vises i listing 4.4. En server behøver ikke tilhøre noen bestemt gruppe. Alle servere definert i inventory-fila er implisit "barn" av gruppa *all*. En server kan også være medlem av flere grupper samtidig. En gruppe kan også ha undergrupper (*children*).

Gruppering av serverne gjør hovedsaklig to ting. Det er det som lar en *playbook* be-



Figur 2: Hierarki på Ansible-oppsett

stemme hvilke servere som er målet for en "run"/kjøring av en playbook og det lar deg angi variabler for en bestemt gruppe servere (noe som forklares nærmere etter hvert).

Hvordan en velger å gruppere de ulike serverne vil selvfølgelig variere ut i fra hvilke type tjenester en har og hvor like/ulike de enkelte serverne er. En annen faktor er hvordan en velger å organisere variablene sine. Færre grupper gjør det nødvendig med mer logikk innad i *playbooks/roles*, mens mange spesifikke grupper minsker logikkbruken, men gjør at plasseringen av variablene blir mer fragmentert (spredt over flere filer). Et generelt råd for konfigurasjonsstyring av servere er å ha så like servere som mulig, med tanke på OS og basisoppsett. Det gjør alt veldig mye enklere da en slipper å ta så mange individuelle hensyn.

En maskin kan gjerne tilhøre både en gruppe basert på tjenestetype, eks. web eller database, og en gruppe som knytter maskinen til hvilket OS den har, eks. Debian eller CentOS.

Det å kunne definere overordnede variabler på OS-nivå er praktisk. Varibelen *ansible_user* som angir hvilken SSH-bruker Ansible logger på målmaskinene med kan brukes som eksempel på dette. For henholdsvis Debian- og CentOS-maskiner er brukeren enten *debian* eller *centos*. Enten må dette defineres for hvert *play* i en *playbook*, hvorpå det må skrives et *play* for hver OS-type, eller så defineres bare brukeren som en variabel tilhørende gruppene *Debian* og *RedHat* og så slipper man å tenke mer på det (fordi Ansible automatisk fikser det).

Et annet eksempel på hva som er praktisk å kunne definere på OS-distribusjonsnivå er *ansible_python_interpreter*. Ansible bruker Python2.7 som standard. For distroer som bare er levert med Python3 må python-interpreter spesifiseres til å være denne versjon [23].

Listing 4.4: Utdrag fra inventory-filen i Ansible eksempeloppsett 5.4

```
# Alle Debian-maskiner
[Debian:children]
webdeb
dbdeb

# Alle webservere
[web:children]
```

```
webdeb
webcent
webhotell

# Debian webservere
[webdeb]
webdeb01.foo.bar
webdeb02.foo.bar
webdeb03.foo.bar

# Webhotell
[webhotell]
webdeb01.foo.bar
```

4.1.5 Datahåndtering

Ansible er veldig fleksibelt med tanke på hvor en plasserer variabler. Det er nesten slik at det er i overkant mange valgmuligheter. Derfor er det lurt å gjøre det så enkelt og konsistent som mulig. Ansible sin dokumentasjon har en komplett liste over variablenes prioritet [55]. Denne kan umiddelbart virke uoversiktlig, men er egentlig ganske intuitiv. Noen vanlige (men ikke alle) eksempler rundt temaet plassering av variabler diskuteres nedenunder.

Gruppevariabler (Group_vars)

Gruppevariabler gjør det mulig å definere variabler for ulike grupper av servere. Dette gjøres ved å opprette filer i mappen *group_vars/* der filnavnet korresponderer med gruppenavnet i *inventory*-filen. Som allerede nevnt er alle servere implisitt medlem av gruppen *all*. Variabler som angis i filen *group_vars/all.yml* vil da være globalt gjeldende for absolutt alle servere definert i inventaret. En god regel er da å plassere så mye som mulig av variablene her, både de som gjelder absolutt alle, men også de fleste (majoriteten) av serverne. Eksempler på slike data kan være generelle brannmurregler, som at at port 22 må være åpen på alle serverne og NTP-spesifikke data og tidssone. Her må det også nevnes at en rolle gjerne har angitt fornuftige standardverdier (*role default*, som er aller lavest på variablenes rangstige). I en rolle hentet fra Galaxy kan det tenkes at de aller fleste av disse kan brukes som de er, men at enkelte behøver å overskrives av gruppevariabler.

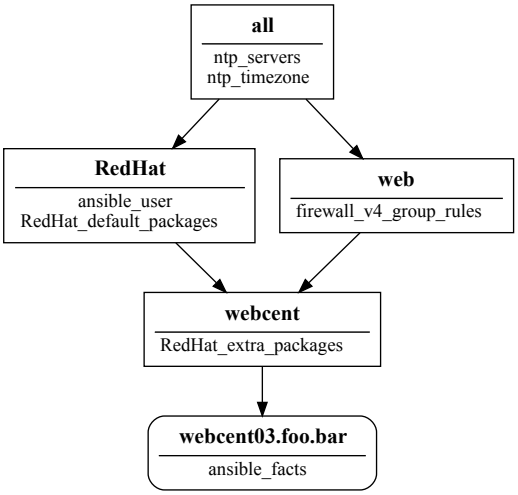
For gruppen *web*, som definerer hvilke maskiner som er webservere, trengs det gjerne gruppespesifikke varibler bare for disse. Et eksempel på dette er gjerne brannmurregler, der port 80/443 må være åpen. Dette defineres da i filen *group_vars/web.yml*. Gruppevariabler for en spesifikk gruppe av maskiner har høyere rang enn gruppen *all*. Er samme variabel definert to steder vil stedet med høyest rang “vinne”, og dens verdi vil bli gjeldende.

Tjenervariabler (Host_vars)

Varibler kan også defineres helt ned på enkeltserver-nivå. Dette gjerne noe en helst vil unngå, men muligheten er i hvert fall der. *hostvars* har høyere rang enn *group_vars*.

Playbook-variabler

Et alternativ til gruppevariabler er å plassere variablene i selve playbooken. Fordelen med dette er at det blir umiddelbart tydelig hvilke variabler som er gjeldene ved å åpne en playbook, istedenfor å begynne å sjekke de forskjellige gruppevariabel-filene.



Figur 3: Eksempel på gruppevariabler for en enkelt webserver

Eksempel på variabel-hierarki

Grafene i figur 3 og vedlegg D.1 visualiserer ved hjelp av *Ansible-Inventory-Grapher* hvordan variabelhierarkiet fungerer med gruppen *all* som som rot i treet, etterfulgt av *group_vars* og til slutt *host_vars* (bladnodene) [204]. En maskin vil få variabler fra alle gruppene den tilhører (altså via alle veier som fører til den gitte noden).

I figur 3 får noden *webcent03.foo.bar* NTP-variabler fra *all*-gruppen, Ansibles SSH-bruker og programvarepakker fra *RedHat*-gruppen, spesifikke brannmurregler for webservere fra *web*-gruppen og *webcent*-gruppen angir ekstra programvarepakker i tillegg til standardpakkene. De enkelte nodenes *facts* vil ha høyere rang enn *group_vars* og *host_vars*. Eksempelet i vedlegg D.1 viser at det fort blir mer komplekst når antall servere og grupper øker. En bør spesielt passe på å unngå å definere samme variabel flere ganger på samme nivå i gruppehierarkiet, eksempelvis både i gruppen *RedHat* og *web*. Den siste lastede variablen vinner alltid og tidligere refererte Ansible-dokumentasjon forklarer denne rekefølgen nærmere.

Kombinering av variabler (Hash-merging)

Ansible vil som standard ikke gjøre *hash-merging*. Det er mulig å slå dette på i konfigurasjonsfilen, men det er ikke den anbefalte måten å jobbe på og derfor tok heller ikke eksempeloppsettet i punkt 5.4 i bruk denne opsjonen [8]. Det er imidlertid mulig å organisere roller slik at variabler, f.eks to lister med brannmurregler, slås sammen i rollen.

4.1.6 Skalering og ytelse

Å skalere opp en playbook fra å omfatte en maskin til mange maskiner er svært enkelt, fordi det bare innebærer å definere de nye maskinene i *inventory*-filen og sikre at de kan nås via SSH.

En undersøkelse av tidsbruk for utrulling av programvare-pakker viste at Ansible var tregere enn både Chef og Salt, men det nevnes imidlertid at koden ikke var optimalisert og siden alle verktøyene har mulighet for parallelliserte tilkoblinger vil bare båndbredde i nettverket være det som begrenser ytelsen[61]. I tillegg til båndbredde vil også CPU/-

minne på kontroll-maskinen være en begrensende faktor, men Ansible oppgir kun tall for Tower [28].

Ansible har flere muligheter i konfigurasjonsfilen til å optimalisere tidsforbruk [9]. *Forks* bestemmer hvor mange maskiner som blir konfigurert parallelt. Standardverdi er kun 5, noe som er veldig konservativt. Det er vanskelig å finne konkrete tall for hva som er en god verdi, det kommer helt an på tilgjengelig ytelse. På Ansibles blogg, fra helt tilbake til 2014, anslår Ansibles utvikler Micheal DeHaan 25-100 forks [102]. Eksempeloppsettet 5.4 er for lite til at en får testet dette i praksis. SSH-pipelining, som er deaktivert som standard reduserer antall nettverks-operasjoner, og vil gi ytelsesforbedring [9]. *Strategy*-valget for Playbook-kjøringer vil som standard fullføre et tasks på alle maskiner før neste task starter [43]. Endres dette til at alle maskiner kjører alle sine tasks så fort som det er mulig kan også tid spares. Enkle ytelsestester, vedlegg D.2, gjort på eksempeloppsettet 5.4 viste at en kjøring av *Master playbook, site.yml*, på allerede ferdig konfigurert oppsett, viste at pipelining reduserte tidsbruken med 16% og pipelining sammen med endret *strategy* reduserte tidsbruken med 38% sammenlignet med standardverdier.

En muligheter for å bruke Ansible i tilnærmet uendelig stor skala er *Ansible Pull* som lar hver enkelt maskin lytte til et Git-repo og kjøre playbooks lokalt [22]. Bakdelen med dette er at en mister sentralisert logging/kontroll [102]. I tillegg finnes det mer eksperimentelle prosjekter som *Mitogen for Ansible* som lover stor ytelsesforbedring [85].

4.2 Puppet

Utgivelsen av Puppet som ble benyttet i denne rapporten var versjon 5. Puppet-plattformen består av i hovedsak tre pakker: *puppet-agent*, *puppetserver* og *puppetdb* [142].

puppet-agent inneholder selve kjernen (Puppet), Ruby, Facter, Hiera og annen omkringliggende kode som støtter opp om Puppet.

puppetserver inneholder server-programvaren som agentene kobler seg til for å motta konfigurasjoner og data.

puppetdb installerer en database skreddersydd for Puppet-installasjoner, som oppbevarer ulike data og resultater om styrte maskiner (agenter).

Disse tre komponentene er sentrale i en Puppet-arkitektur. PuppetDB er ikke strengt nødvendig, men kan være svært nyttig for å samle statistikk og annen informasjon om de styrte maskine i infrastrukturen.

4.2.1 Arkitektur

Puppet server

Dette er navnet på tjeneren i et Puppet-oppsett [164]. På grunn av Puppets klient-tjener-modell vil mye av arbeidet og prosessene i et Puppet-oppsett være sentrert rundt serveren. Det skilles mellom to mye brukte navn når det refereres til denne delen av arkitekturen:

- *Puppet master:* Består av HTTP-APIet
- *Puppet server:* Håndterer HTTP-tilkoblinger og en del andre tjenester

Puppet *master* er i dag en del av Puppet *server*, men dens opprinnelige kode skrevet i Ruby vedvarer fortsatt, selv om Puppet server er skrevet i Java. Ved hjelp av oversetting med JRuby kan master og server fungere som én server-tjeneste for Puppet agent. Før

Puppet server kom, ble HTTP-forbindelsene håndtert sammen med Apache. Etter versjon 3.7 av Puppet Enterprise var server og master konfigurert som én og samme ting.

HTTP-APIet er sentralt i arkitekturen, og det agentene benytter for å hente konfigurasjoner og data fra serveren. All kommunikasjon som agentene initierer gjøres over HTTPS/TLS, noe som krever at sertifikater må håndteres. Dette gjør også serveren med sin innebygde tjeneste for sertifikatutsteding og signering (Certificate authority, CA). Før kommunikasjonen mellom agent og server starter, må agentens sertifikat verifiseres og godkjennes med signatur.

Puppet agent

Styrte maskiner må ha Puppet agent installert for å kunne motta konfigurasjonskataloger fra serveren [163]. Agenten henvender seg til serveren på regelmessige intervaller for å hente nyeste konfigurasjon. Standard intervall er hvert 30. minutt. Første gang agenten henvender seg til serveren, må den spørre om å motta et sertifikat fra serverens innebygde CA-tjeneste. For at agenten skal kunne laste ned sertifikatet, må det bli signert av serveren først [1].

Alle maskinene bør ha et fullstendig kvalifisert domenenavn (FQDN) som er søkbart ellers i infrastrukturen. Dette vil ha betydning også for sertifikater, siden navnet på sertifikatet automatisk blir maskinens FQDN.

En alternativ "arkitektur" er å kjøre agenten selvstendig, uten en Puppet servermaskin å koble seg opp mot [136]. Da kreves alternative måter å distribuere konfigurasjonsfilene til disse maskinene. Det finnes ulike argumenter for og mot bruken av selvstendige agenter, men generelt mister en den sentraliserte kontrollen, oversikt og rapportering/monitorering av maskinene.

4.2.2 Puppet server: synkronisering av kode og data

For at et Puppet-oppsett skal være mest mulig automatisert, bør konfigurasjonskode og data hentes automatisk når endringer blir gjort eksternt i et versjonskontrollsystem. Det er i hovedsak tre elementer i Puppet som kan sørge for dette.

Kontrollrepo

Puppet kaller repoet som brukes for oppbevaring av konfigurasjoner for *control repository* eller kontrollrepo [133]. I tillegg til konfigurasjoner, kan kontrollrepoet inneholde en fil kalt "Puppetfile" som lister opp nødvendige moduler og deres versjon. Sånn sett vil ingen modulkode nødvendigvis ligge i kontrollrepoet, men installeres på serveren når Puppetfile blir lest av r10k.

Dette repoet bør befinne seg i et versjonskontrollsystem, som Git. Hovedgren i kontrollrepoet må hete "production". Hver nye gren, f.eks testing, vil kunne bli et nytt miljø i Puppet.

Miljøer

Miljøer er separate kataloger for ulike grener i kontrollrepoet [114]. I environment-katalogen under kodekatalogen på serveren (/etc/puppetlabs/code/environments), oppbevares alle grener som ble opprettet i kontrollrepoet i hver sin katalog. Grener/miljøer kan benyttes til testing og separering av maskiner i infrastrukturen. Ulike versjoner av de samme modulene/klassene skal altså kunne testes isolert fra produksjonssystemene. For

at den nyeste koden og grenene/miljøene skal befinne seg på serveren, må det brukes et program, f.eks r10k, for å kopiere og organisere det som er nødvendig.

r10k

R10k kan koble seg opp mot flere Git-repoer og hente ned nødvendig konfigurasjonskode som skal fra ett eller flere kontrollrepoer til serveren [132]. r10k sørger også for å kopiere riktig gren til riktig miljø. En ny gren i kontrollrepoet, blir altså automatisk ordnet med r10k. Når r10k kjøres vil den også kunne lese Puppetfile-fila i kontrollrepoet for å installere alle nødvendige moduler som inngår i miljøet.

4.2.3 Puppet-språket

Puppets domenespesifikke språk (DSL) er kjernen i den modelldrevne bruken av Puppet for konfigurering av infrastrukturen. Den byr på mange ulike måter å håndtere deklarerte elementer. Boken *Puppet 5 Essentials* (M. Alfke, F. Frank) gir en kort, men bra gjennomgang av det viktigste som kan gjøres i Puppet-språket [1].

Manifester

Hvert manifest er en klasse eller underklasse av et annet manifest/klasse [124]. Det er lett å tenke på disse filene som script, men de deklarerte elementene blir kompilert til en *katalog*, som ofte består av deklarasjoner fra flere manifester. Det er heller ingen garantert rekkefølge for eksekvering av de deklarerte elementene. Hvert deklarert element kalles en *ressurs*.

Ressurser

Byggeblokkene i manifester/klasser er ressurser [169]. En ressurs skal deklarere en modell for en ønsket tilstand for et enkelt element på en maskin eller system. Kanskje de mest kjente og brukte ressursene i Puppet-språket er *package*, *file*, og *service*, som styrer sine respektive ting i operativsystemet på en maskin. Dokumentasjonen beskriver syntaksen som vist i listing 4.5. Et eksempel på en `service`-ressurs vises i listing 4.6

Listing 4.5: Syntaks på ressurser i Puppet-språket

```
<TYPE> { '<TITLE>':
  <ATTRIBUTE> => <VALUE>,
}
```

Listing 4.6: Eksempel på en `service`-ressurs

```
service { 'apache2':
  ensure   => running,
  provider => upstart,
}
```

Variabler

I manifester/klasser kan variabler deklareres. Variabler er egentlig konstanter [130]. Det finnes tre typer variabler: Enkeltvariabler, array og hash. Enkeltvariabler kan oppbevare én dataverdi. Array kan oppbevare flere enkeltverdier i en, og elementene den oppbevarer refereres til med indekser. Hash kan tenkes på som en assosiativ array, der indekser er navnord i stedet for tall. Alle tre har svært nyttige bruksområder, spesielt i forhold til å hente konfigurasjonsdata fra Hiera. Variablene kan deklareres som vist i listing 4.7.

Listing 4.7: Eksempel på deklarering av ulike variabler

```
### Enkeltvariabel
$variabel1 = 'verdi'

### Array
$variabel2 = ['verdi1','verdi2','verdi3']

### Hash
$variabel3 = {
  'Navn1' => 'verdi 1',
  'Navn2' => 'verdi 2',
  'Navn3' => 'verdi 3',
}
```

Datatyper

Variabler kan oppbevare flere typer data eller strukturer i form av tekstverdier, tallverdier, boolske verdier, arrayer, hasher og *undef* [123].

I Puppet-språket kan en variabel spesifiseres til å oppbevare en gitt type data for å verifisere at den blir angitt riktig datatype. Dette gjøres ved eksempelvis å skrive `Boolean $variabelnavn` for å forsikre at den får en boolsk verdi (true eller false) [1].

Kontrollmekanismer

Puppet-koden kan styres avhengig av ulike betingelser eller verdier i variabler [125]. For dette kan *if* eller *case* omslutte ressurser, og en *selector* inne i ressurser. I listing 4.8 vises hvordan disse kan brukes.

Listing 4.8: Eksempler på kontrollmekanismer i Puppet-språket

```
### If
if $os_family == 'Debian' {
  package { 'apache2': ensure => present }
} elsif $os_family == 'RedHat' {
  package { 'httpd': ensure => present }
}

### Case
case $os_family {
  'Debian': package { 'apache2': ensure => present }
  'RedHat': package { 'httpd': ensure => present }
}

### Selector
package { 'apache-httpd':
  ensure => present
  name   => $os_family ? {
    'Debian' => 'apache2',
    'RedHat' => 'httpd',
  },
}
```

Rekkefølge og avhengigheter mellom ressurser

Ofte trenger noen ressurser å bli styrt i en viss rekkefølge [129]. Eksempelvis må en programpakke installeres før tjenesten startes, og før tjenesten startes må konfigurasjonen til programmet være til stede. Ved hjelp av metaparametrene *require* og *before* kan dette styres. *require* kan tenkes på som spesifisering av avhengigheter. *before* kan tenkes på som spesifisering av rekkefølge. Eksempel vises i listing 4.9

Listing 4.9: *before* kjører file-ressurs før ressurs i Service-referansen

```
file {'/etc/apache2/apache2.conf'
  ensure => present,
  before => Service['apache2'],
}
```

Hvis konfigurasjonsfilen har endret seg siden sist, kan ressursen som styrer den varsle ressursen som styrer tjenesten for å restarte den slik at den nye konfigurasjonen blir lastet inn. Dette gjøres ved bruk av *notify* hos ressursen som ble endret til å sende melding til en annen interessert ressurs, eller *subscribe* hos den ressursen som er interessert i informasjonen om at endringen skjedde. Det gjøres omtrent som *before* i 4.9.

I manifestet kan det også, mellom ressursdeklarasjonene, settes inn piler for å styre rekkefølge og notifikasjoner. -> betyr "denne før den", og ~> (tilde og >) betyr "denne varsler den". Førstnevnte omtales som *ordering arrow*, sistnevnte som *notifying arrow*. Eksempel på bruk av piler for rekkefølge og varsler vises i listing 4.10.

Listing 4.10: Eksempel på bruk av piler for rekkefølge

```
### Rekkefolge
file { '/etc/apache2/apache2.conf':
  ensure => present,
} ->

service { 'apache2':
  ensure => running,
}

### Varsel ved endring i foregaende ressurs. Vil restarte service
file { '/etc/apache2/apache2.conf':
  ensure => present,
} ~>

service { 'apache2':
  ensure => running,
}
```

Referanser til ressurser

Som vist i eksempler i listing 4.9, blant annet med *require* og *before*, refereres andre ressurser til ved bruk av en spesiell notasjon [126]. Stor bokstav for ressurstypen, etterfultgt av klammeparenteser, med ressurstittelen/navnet til den deklarerte ressursen. En referanse til en navngitt filressurs vil kunne se slik ut:

`File['/etc/hosts']`

En referanse til en klasse vil kunne se slik ut:

`Class['::profile::base::ssh']`

Facter

Facter er en samling av mange ulike karakteristikker ved en maskin. En mye brukt *fact* er operativsystemfamilie. Siden det eksisterer forskjeller mellom, eksempelvis, Debian- og RedHat-baserte operativsystemer, er det naturlig å basere, eksempelvis, en if-setning på denne karakteristikken for å diskriminere på konfigurasjonsdetaljer. Det fins to måter å hente ut informasjon fra facter hos en maskin: `$fact_name` eller `$facts[fact_name]`. Eksempel i listing 4.11 viser hvordan Facter kan brukes i Puppet-koden for å diskriminere mellom ulike operativsystemer.

Listing 4.11: Facter brukt i Puppet-kode

```
if $facts['os']['family'] == 'RedHat' {
  package { 'httpd': ensure => present }
} elsif $facts['os']['family'] == 'Debian' {
  package { 'apache2': ensure => present }
}
```

4.2.4 Klasser og moduler

Klasser er en måte å dele opp ressurser i passende enheter. Hovedklasser omtales generelt som *moduler*, og siden Puppet er modellorientert, er modulene de overordnede modellene en jobber med for organisering av konfigurasjoner til ulike tjenester [144].

Figur 4.12 viser hvordan en klasse defineres i et manifest. Hovedklassen i en modul blir lagret i filen init.pp, som oppbevares i modulens *manifests*-katalog. Underklasser kan også defineres ved bruk av *scoping* med doble kolon, som vist i listing 4.13

Listing 4.12: Eksempel på definering av en klasse

```
class minklasse {
}
```

Listing 4.13: Eksempel på definering av en underklasse

```
class minklasse::underklasse1 {
}
```

Denne underklassen oppbevares i samme katalog som init.pp, men med navnet på underklassen som filnavn (underklasse.pp). Skal underklassen ha flere underklasser, må en ny katalog opprettes med navnet på underklassen. Den første underklassen etter hovedklassen skal fortsatt ligge i manifests-katalogen. Denne logikken kan følges rekursivt for hver underklasse. Puppet gjenkjenner altså navngiving/namespacing med scoping (dobbel kolon,'::', mellom klasse- og underklassenavn) [127].

4.2.5 Økosystem og brukersamfunn

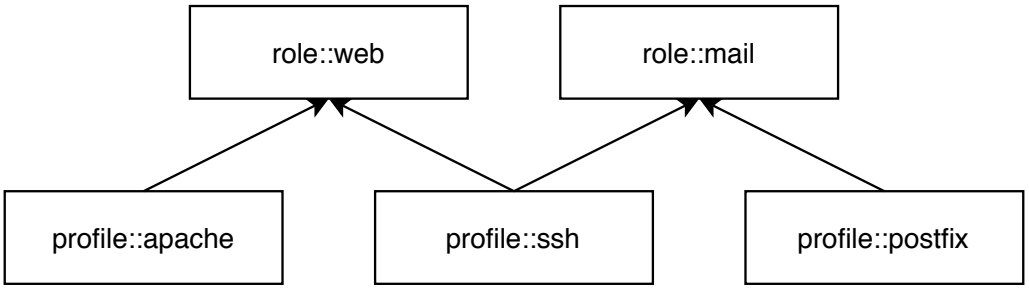
På Puppet Forge samles alle publiserte moduler som brukere har skrevet. Puppet selv har utviklet mange mye brukte moduler [150] [151]. Det fins også en del kjente navn (enkeltpersoner og organisasjoner) som har utviklet moduler av bra kvalitet. saz, ghoneycutt, example42 er noen navn på produktive utviklere av Puppet-moduler [152] [148] [147].

4.2.6 Gruppering av konfigurasjoner

Manifester inneholder klasser, som i prinsippet er moduler i Puppet. Moduler er den modelldrevne måten å organisere konfigurasjoner på i Puppet. Her beskrives to mønstre som kan brukes, litt avhengig av hvordan det klasser velges å bli strukturert og organisert.

Hostgroup-mønsteret

Det første mønstret er å simpelthen lage nye klasser og underklasser for hver interne tjeneste som skal konfigureres. Siden Hiera-data kan grupperes i moduler (se 4.2.7), kan dette være en hensiktsmessig løsning for å skille mellom ulike maskiners data. CERN kaller dette mønstret for *hostgroups* [86]. Tanken er at et sett med noder som tilbyr en spesiell tjeneste internt i organisasjonen kalles *hostgroup*. For å skille host-grupper fra "normale" moduler, er å bare kalle sistnevnte for "moduler" [87]. Moduler er generelle og gjenbrukbare mens host-grupper er ikke.



Figur 4: Illustrasjon av rolle-profilmønstret

Rolle-profilmønsteret

Det andre mønstret kalles for *Roles and Profiles*. I rolle-profilmønstret samles konfigurasjoner i én større modul kalt *profile*, og bruker scoping for å skille mellom ulike interne tjenester [170]. Eksempelvis kan klassen for en for en intern tjeneste kalles **profile::web**, eller **profile::database**. De kan ytterligere deles/scopes: **profile::web::forum**, og **profile::database::forum**. Dette kan gjøres på forskjellige måter, alt etter hvordan konfigurasjoner og konfigurasjonsdata organiseres. Når forskjellige profiler for en tjeneste er skrevet, kan disse inkluderes i en role-klasse, som også eksisterer som sin egen modul. Eksemplet i listing 4.14 viser hvordan profil-klasser kan inkluderes i en rolle-klasse. Figur 4 illustrerer dette.

Listing 4.14: Eksempel på en rolle

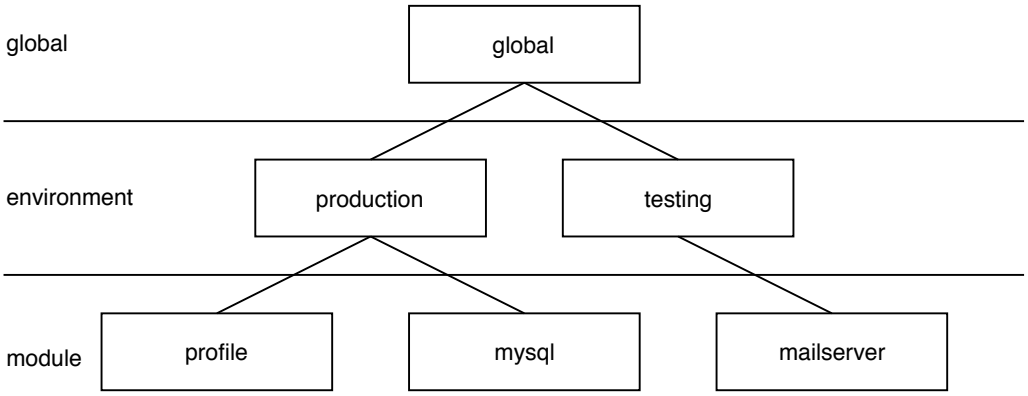
```
class role::web::forum {
  include ::profile::web
  include ::profile::web::forum

  Class['::profile::web'] -> Class['::profile::web::forum']
}
```

En mulig ulempe med dette mønsteret er at all Hiera-data må oppbevares for alle profiler på samme sted, i hovedsak på miljønivået i Hiera (se 4.2.7). Host-gruppe-mønstret tillater altså å separere i flere enheter per host-gruppe. Det skal fint gå an å kombinere både rolle-profil-mønstret og host-gruppe-mønstret. Da er det likevel lurt å være konsistent og klar på hvilke deler som plasseres hvor. Kanskje kan mer generelle konfigurasjoner, som SSH og NTP, skrives som profiler. Mer komplekse tjenester, som konfigurasjonen av en web-tjeneste eller en databaseklynge, kan plasseres i egne host-gruppemoduler omtrent slik CERN gjør.

4.2.7 Separering av kode og data med Hiera

I Hiera oppbevares data i nøkkel-verdipar i form av enkeltvariabel, array eller hash, strukturert YAML [118]. Som vist i figur 5, starter hierarkiet på globalt nivå. Dette nivået overstyrer alt som er definert under. Neste nivå er per miljø, og vil overstyre alle moduler i dette miljøet. Det siste og laveste nivået er moduler. Hierarkiet styrer oppslag etter data, og data funnet høyere i hierarkiet har høyere prioritet. Med dette menes at variabler med samme navn kan defineres flere steder i hierarkiet, men med ulike data.



Figur 5: Hierarkiet i Hiera

Konfigurering av Hiera

Hvert nivå av hierarkiet har ytterligere et lagdelt hierarki [115]. Dette lagdelte hierarkiet for hvert nivå defineres i en `hiera.yaml`-fil. Stien til disse filene er som regel noe av følgende fra katalogen `/etc/puppetlabs`:

- **Global-nivå:** `./puppet/hiera.yaml`
- **Miljø-nivå:** `./code/environments/<miljønavn>/hiera.yaml`
- **Modul-nivå:**
  `./code/environments/<miljønavn>/<modulkatalog>/<modulnavn>/hiera.yaml`

En `hiera.yaml`-fil kan se ut som vist i listing 4.15. Dette eksemplet viser hvordan data kan struktureres på miljønivået, som antakeligvis er mest typisk plassering for generelle konfigurasjonsdata.

Listing 4.15: Eksempel på `hiera.yaml` for miljønivået

```
### ./<miljonavn>/hiera.yaml
### f.eks: /etc/puppetlabs/code/environments/production/hiera.yaml
version: 5
defaults:
  data_hash: yaml_data
  datadir: data
hierarchy:
  - name: "Verdier for enkeltnoder"
    path: "nodes/\%{trusted.certname}.yaml"
  - name: "Verdier avhengig av operativsystem"
    path: "os/\%{facts.os.name}.yaml"
  - name: "Verdier avhengig av tjeneste"
    glob: "services/*.yaml"
  - name: "Felles verdier for miljo"
    path: "common.yaml"
```

Det som listes etter `hierarchy:`-linja bestemmer alle kataloger og filer Hiera skal søke gjennom for å finne variabelnøkler med verdier. Hierarkiet, også her, følger en prioritert rekkefølge. Det vil si at en variabel kan deklareres flere ganger med ulike verdier, men plassering høyere i lista vil føre til at den blir valgt over andre.

I det øverste laget i listing 4.15 for enkeltmaskiner vises bruken av interpolering av filnavnet med en variabel (%{..}-syntaksen). På neste nivå under kommer data som kan variere mellom ulike operativsystemer. Kanskje det mest interessante laget som kommer under, det for tjenester. Etter dataene for spesifikke tjenester og helt til slutt (for eksemp-

let) er data som er felles for alle maskiner i miljøet.

Informasjonen i `defaults:` bestemmer diverse innstillinger, blant annet hvilke katalog Hiera skal søke i. I miljøet `./production` vil eksemplet i listing 4.15 peke på `data`-katalogen under den.

Deklarering av Hiera-data

Lagring og strukturering av data i YAML-filer i Hiera er en enkel sak, men det gjelder å finne et godt system for variabelnavn [115]. Eksemplet i listing 4.16 viser hvordan NTP-tjenere kan navngis som variabel og listes som en array i `common.yaml`. Under NTP-tjenerene vises hvordan SSH-tjenerinnstillinger kan deklareres som en hash.

Listing 4.16: Eksempel på strukturering av Hiera-data i YAML

```
### common.yaml
# NTP-tjenere
ntp::servers:
  - '0.no.pool.ntp.org'
  - '1.no.pool.ntp.org'

ssh::server_options:
  PrintMotd: 'no'
  Port: '22'
  PermitRootLogin: 'no'
```

Oppslag i Hiera

I Puppet-koden gjøres oppslag i Hiera for å hente de aktuelle dataene som skal brukes i manifester [131]. Eksemplet i listing 4.17 viser hvordan listen over NTP-servere fra eksemplet i listing 4.16 kan hentes med `lookup()`-funksjonen og settes inn i `ntp`-klassen som deklareres i et manifest [160]. Funksjonen kan også brukes med variabler, som vist i samme figur for SSH.

Listing 4.17: Eksempel på oppslag i Hiera med `lookup()`-funksjonen

```
class { 'ntp':
  servers => lookup('ntp::servers'),
}

$ssh_server_options = lookup('ssh::server_options')
class { 'ssh::server':
  options => $ssh_server_options
}
```

Hiera-data på modulnivå

Hvis det er ønskelig å benytte mønsteret for host-grupper, slik CERN gjør i sitt system, framfor rolle-profil-mønsteret, kan Hiera-data oppbevares på modulnivå [115].Disse dataene kan likevel overstyres på miljø- og global-nivå. Hvis en ikke ønsker å samle all konfigurasjonsdata på global- eller miljønivå i Hiera, kan de altså separeres i mindre håndterbare enheter per host-gruppe.

4.2.8 Nodeklassifisering

Det å angi en maskin sin konfigurasjon blir ofte kalt *node defninition* eller *node classification* [128]. Dette gjøres i det såkalte hovedmanifestet [119]. Hovedmanifestet er enten enkeltfila `site.pp` som ligger i `.manifests`-katalogen i miljøet. Alternativt kan flere `.pp`-filer utgjøre hovedmanifestet i samme katalogen.

Hovedmanifestet er der Puppet starter å lese deklarasjoner. Her deklareres hver enkelt maskin eller grupper av maskiner sine klasser. Følges rolle-profilmønstret eller lignende vil hver enkelt maskin ha kun én deklarert hovedklasse. Eksemplet i listing 4.18 viser hvordan en enkeltmaskin kan klassifiseres med en rolle. Alternativt kan mange maskiner angis en enkelt klasse ved bruk av lister regulære uttrykk i som vist i listing 4.19.

Listing 4.18: Eksempel på nodeklassifisering

```
node 'web01.foo.bar' {
  include ::role::web::forum
}
```

Listing 4.19: Eksempel på nodeklassifisering med regulært uttrykk

```
node /^web\d+\.foo\.bar$/ {
  include ::role::web::homepage
}
```

Ekstern nodeklassifisering

Å vedlikeholde filer for nodeklassifisering på denne måten kan ha sine begrensninger. Puppet tillater en annen måte å klassifisere noder på, uten bruk av `.pp`-filer. Ekstern nodeklassifiserer kan benyttes (*External Node Classifier*, ENC) [172]. En slik ENC vil hente klassifikasjonen fra en annen datakilde/database og gi den til Puppet Server i YAML-format.

En ENC kan skrives som et simpelt script. Dette scriptet kan hente nodeklassifiseringen fra en database som er strukturert på det viset som er ønskelig, så Puppet kan mate scriptet med FQDN (eller det som er sertifikatnavnet) og få ut en liste over miljø noden hører til, hvilke klasser og parametre den skal få.

4.2.9 Skalering av Puppet server

Det er mulig å skalere Puppet server til et distribuert oppsett ved bruk av såkalte "compile masters" [171]. Siden Puppet server kompilerer en katalog over faktiske endringer en maskin skal ha, vil den bruke mye tid og ressurser på å kompilere for mange hundre og kanskje tusen maskiner. Kompileringen i seg selv kan altså legges til flere lastbalanserte *compile masters*.

5 Eksempeloppsett

Forrige kapittel, Utvidet beskrivelse av Ansible og Puppet, er et grunnlag for beskrivelser i denne delen av rapporten, som handler om eksempeloppsettene for både Ansible og Puppet. Når de to kandidatene ble valgt, ga oppdragsgiver gruppen en liten liste over tjenester de ville se eksempel på konfigurasjoner av. Disse tjenestene virket greie, men viste seg å by på ulike problemstillinger for begge kandidatene.

5.1 Forutsetninger

Dette eksempeloppsettet krevde først og fremst kunnskapen om Ansible og Puppet. Mye av denne kunnskapen ble tilegnet under selve arbeidet. Kildene for denne kunnskapen har i hovedsak vært offisiell dokumentasjon og bøker.

De andre forutsetningene var i hovedsak tekniske. Det var behov for maskinressurser, og dette var mulig å benytte gjennom skyløsningen SkyHiGh. Mer om de spesifikke maskinressursene i 5.3. I tillegg ble all eksempelkode produsert i oppsettet lagret i flere repoer i et prosjekt i BitBucket[1].

5.2 De ønskede tjenestene

Oppdragsgiver ønsket å se eksempeloppsett som inkluderte følgende tjenester:

- NTP-klientkonfigurasjon
- SSH-serverinnstillinger (sshd_conf)
- Web-tjenester
  - Styring av Apache-moduler
  - Konfigurere "webhotell"-maskin med mange *Virtualhosts* (definisjoner for nettsider på en web-server)
- Sympa
  - Sympa og web-server (Apache)
  - Mail Transfer Agent (Postfix)
  - Database (MySQL)

I forberedelsene til eksempeloppsettene ble det oppdaget at Sympa, en tjeneste for epostlister, ikke hadde en fungerende rolle i Ansible Galaxy, og modulen i Puppet Forge hadde ikke blitt vedlikeholdt siden 2015 (samt at det tilhørende GitHub-prosjektet og brukeren var forsvunnet) [101]. Det ble likevel funnet en GitHub-bruker kalt *The Sympa Community* som hadde en rolle for Ansible [200] og et manifest for Puppet [201].

For web-servere generelt vil ulike nettsider kunne defineres ved bruk av såkalte *virtualhosts*, som definerer nettsidens navn og innstillinger [197].

En annen problemstilling var å finne ut hvordan konfigurasjoner kan gis til maskiner på ulikt vis. SSH-serverkonfigurasjonen som oppdragsgiver viste til, viste at ulike ma-

[1] Koden fra BitBucket-repoene er vedlagt som arkivfil i leveransen av denne rapporten.

skiner måtte ha ulike innstillinger for SSHD. Ulik konfigurasjon måtte angis basert på gruppemedlemskap, operativsystem, enkelttjenester, alternativer for autentisering, også videre. Filen oppdragsgiver viste til var en mal (sshd_config.cfdef) for sshd_config-fila hos maskinene, der CFEngine la til forskjellige linjer basert på maskinens gruppemedlemskap.

5.3 Maskinressurser i SkyHiGh

I OpenStack-løsningen SkyHiGh ble det opprettet to "stacks", eller samlinger av maskinressurser. Én stack for hver kandidat. I tillegg ble en "konfigklient" installert med Git og koblet opp mot repoene i BitBucket.

Maskinressursene var som følger:

- Konfigklient
- Stacks (én hver for Ansible og Puppet)
    - Nettverket 192.168.180.0/24
    - 1 Debian/Ubuntu-maskin (Ansible/Puppet) som "master" i oppsettet
    - 3 Debian/Ubuntu-maskiner (Ansible/Puppet) som web-servere
    - 3 CentOS-maskiner som web-servere
    - 1 Debian/Ubuntu-maskin som database-server
    - 1 CentOS-maskin som database-server
    - 1 DNS-tjener (ikke styrt av CM-system)

Alle maskinene hadde såkalte "floating IP-address", som tillater å nå maskinene utenfor det private nettverket i stacken. Dette var nødvendig blant annet for å nå de over SSH fra skolenettet til det private nettverket i stacken, og teste ulike tjenester som ble konfigurert.

Maskinene ble opprettet gjennom OpenStack Heat-APIet i SkyHiGh. Det betyr at alle maskinene og nettverk ble definert i strukturerte YAML-filer og matet til APIet for å opprette ressursene. Disse filene ble oppbevart i sitt eget repo i BitBucket. I tillegg ble oppstartsskript kjørt på hver maskin for å gi de et grunnleggende oppsett.

5.3.1 DNS

Det ble som nevnt installert en DNS-tjener i hver stack. Formålet med en DNS-tjener var å gjøre nettverkstilkoblinger mellom maskinene enklere. IP-adresse til navne-bindinger foregår på det private nettverket. Dette er automatisk for både master-maskinen og DNS-maskinen, siden disse hadde statiske IP-adresser. Men for de styrte maskinene som blir tildelt adresser av OpenStack må bindinger skrives inn manuelt hver gang en stack blir opprettet, slik oppsettet i dette tilfellet ble. En bedre løsning kunne vært å bruke "nsupdate" hvor serverne melder seg inn i domenet, men dette ble nedprioritert på grunn av tid.

DNS viste seg å ikke fungere helt som tenkt i Puppet-stacken. Problemene gikk ut på at OpenStack-systemet tildeler maskinene noe DNS-informasjon. Problematikken ble ikke grundig undersøkt, og det endte opp med å bare redigere /etc/hosts-filen på alle maskiner for å nå Puppet Server-maskinen med navnet *puppet* knyttet til IP-adressen 192.168.180.107.

Oppdragsgiver nevnte at deres virtualiseringsløsning leverer en virtuell maskin som

har fått IP-adresse og at CM-systemet tar seg av den videre konfigureringen. Sånn sett vil oppsettet i OpenStack simulere det samme utgangspunktet som oppdragsgiver har på en god måte.

5.4 Ansible

5.4.1 Beskrivelse av oppsett

Under opprettelse av stacken ble det kjørt et boot-script som distribuerte SSH-nøkler og installerte Ansible og Git på masteren. I tillegg ble det manuelt satt opp nøkkel med lesetilgang til Bitbucket-repoet, slik at Ansible-repoet kunne klones inn og nye endringer hentes etter hvert. Hovedmålet med eksempeloppsett var først og fremst å teste CM-systemene, derfor ble det ikke satt opp fullverdig CI/CD-pipeline. Dette bør selvfølgelig være med i et et produksjonsmiljø, men ble utelatt på grunn av begrenset tid til rådighet. Det bør også nevnes at ikke all Ansible-kode er skrevet fullstendig på "best practice"-måte eller fullstendig konsistent, men er ment å vise fram forskjellige måter å løse ting på.

Repository

All Ansible-kode ble oppbevart i repoet i Bitbucket. Mappestrukturen ble satt basert på Ansibles best practice-eksempel [6]. Ansible sier selv at dette oppsettet er ment som et utgangspunkt og kan tilpasses etter egne behov hvis nødvendig.

> Your usage of Ansible should fit your needs, however, not ours, so feel free to modify this approach and organize as you see fit[6].

Ansible har ikke noe tilsvarende Puppets R10k (se 4.2.2) som henter konfigurasjonen fra versjonskontrollsystemet og installerer avhengigheter automatisk. Dvs. at en er avhengig av å bruke *git pull* for å hente oppdatert kode, hvis en ikke velger å skrive en playbook (kjørt lokalt) som automatiserer prosessen. For å installere roller fra Galaxy, GitHub, m.fl. defineres disse i filen *requirements.yml* som ligger i *role*-mappa og kjøres med kommandoen *ansible-galaxy install -r requirements.yml* [15].

Ansible-Controller

Kontrollmaskinen fikk nyeste versjon av Ansible-programvaren for Debian basert på den offisielle installasjonsguiden [41]. Nyeste Ansible-versjon var 2.4.3 da arbeidet ble startet, men denne ble oppdatert til Ansible versjon 2.5 som ble sluppet underveis i arbeidet med eksempeloppsettet.

5.4.2 Organisering av playbooks og roller

Eksempeloppsettet følger i stor grad allerede nevnte best-practice-prinsipper. Det ble opprettet en master playbook *site.yml* som ikke gjorde noe annet enn å inkludere to playbooks, *webservers.yml* og *dbservers.yml*.

Et mål med eksempeloppsettet var å bruke ferdige roller fra Ansible Galaxy. Som allerede nevnt holder disse kanskje ikke samme kvalitet som de i Puppet Forge, men etter litt prøving og feiling ble det funnet noen som fungerte tilfredstillende. Hvis en ikke er fornøyd med hvordan rollene fungerer er det ganske lett å endre disse til å passe egne behov, men en mister da støtte og oppdateringer. Bruker en Ansible må en rett og slett regne med å måtte skrive en del egne roller.

5.4.3 Baseoppsett

En egen rolle kalt *Base* samlet alt basisoppsett som pakkeinstallasjon, NTP, brannmur og SSH. Det eneste Base-rollen gjør er å importere de andre rollene som tasks. Alle parametere blir angitt i *group_vars*-filene, men det er selvfølgelig også mulig å angi disse direkte i selve rollen. En verdi som sannsynligvis aldri behøver endres på, eks tidssonen, kan like gjene defineres i rollen, men på den andre siden kan det være mer oversiktlig og synlig å samle alt på en plass i *group_vars*. Istedenfor å ha roller i rolle er det selvfølgelig også mulig å importere alle de individuelle rollene hver for seg i de ulike playbookene.

5.4.4 NTP

For NTP-konfigurasjon ble rollen *geerlingguy.ntp* benyttet [96]. Variabler for *tidssone*, *NTP-tjenerne* og *begrensninger* ble satt i *group_vars/all.yml* slik at de skal være globale for absolutt alle tjenerne.

5.4.5 SSH

Rollen *willshersystems.sshd* ble benyttet til å styre SSHD-konfigurasjonen [206]. I eksempeloppsett ble alle variabler som er felles for Debian/RedHat satt i *group_vars/all.yml*, mens OS-spesifikke variabler for Debian og RedHat ble satt i *group_vars/Debian.yml* og *group_vars/RedHat.yml*. Dette var strengt talt ikke nødvendig, fordi rollen vil som standard lage SSHD-konfigurasjonen tilsvarende det som er standardverdier for de ulike OS-familiene. Hvis noen grupper av maskiner, eller enkeltmaskiner, avviker fra standard kan disse defineres i tilhørende variabelfiler. Disse vil da ha høyere rang enn *group_vars/all.yml* og vil overskrive disse parametrene.

5.4.6 Brannmur

Ansible kan styre IPtables med sin egen *iptables*-modul [18]. Det finnes ulike roller i Galaxy som tar seg av brannmur-oppsett, men ingen syntes å være skikkelig bra med tanke på organisering av regler. Valget sto mellom *geerlingguy.firewall* og *mikegleasonjr.firewall* [95] [105]. Førstnevnte lot deg enkelt spesifisere port-nummer, men hadde ikke muligheten for å enkelt definere standard regler for alle maskiner som kunne slås sammen med tjenestespesifikke regler. Den andre var bedre på organisering av standard/gruppespesifikke relger, men der måtte en bruke IPtables-syntaks for å skrive reglene. (I tillegg funger de litt forskjellig i at den første bruker iptables, mens den andre bruker iptables-persistent). Eksempeloppsettet endte opp med å bruke *mikegleasonjr.firewall*.

En fallgruve med Ansible og brannmur er å stenge port 22 for SSH. Gjør en det blir en låst ute fra maskinene. Ekstra aktsomhet bør derfor utvises når en gjør brannmurendringer.

5.4.7 Web-server

For oppsett av webtjenere ble det opprettet to roller, *web* og *webhotell* og en playbook kalt *webservers.yml* som inneholdt to *play*. Det første *play* brukte web-rollen til å sette opp fire like webtjenere (to stk Debian/to stk CentOS), mens det andre satte opp en Debian og en CentOS-maskin med to forskjellige webhotell, dvs. samme konfigurasjon, men forskjellige vhosts.

Web-rollen og webhotell-rollen benyttet *geerlingguy.apache*-rollen til å sette opp Apache [94]. Web-rollen gjør i eksempeloppsettet ikke spesielt mye. Den benytter bare standard

vhost og kopierer en index.html-fil til /var/www/html-mappa. Dette for å illustrere et oppsett med mange replikerte webservere bak en lastbalanserer. For mer detaljert beskrivelse av konfigurasjon av Apache-rollen, se punktet under om webhotell.

Styring av Apache-moduler

Mulighet til å styre Apache-moduler var også et ønske fra oppdragsgiver. Ansible kommer med en egen modul, *apache2_module* som kan gjøre dette (men kun for Debian-familien) [5]. Det som er litt negativt med denne er at den ikke selv installerer moduler utover de som kommer standard med Apcache, så det må gjøres via en egen *task*, eller som en del av pakkeinstallasjonen for webtjenerne. Apache-rollen i eksempeloppsettet kan selv gjøre jobben med å styre moduler hvis brukeren angir variablene *apache_mods_enabled* og *apache_mods_disabled* (se figur 5.1). I tillegg er det også mulig å definere alle Apache-pakkene en trenger i *apache_packages*-variabelen.

Listing 5.1: Eksempel på styring av Apache-moduler i rollen *geerlingguy.apache*

```
# Styring av Apache-moduler i geerlingguy.apache-modul
    apache_mods_enabled:
      - rewrite.load
      - ssl.load
```

Webhotell med mange Virtualhosts

Webhotellet-oppsettet ble gjort litt mer avansert enn det enkle webtjener-oppsettet. Her ble det benyttet vhosts både med og uten SSL. Hvert webhotell fikk totalt fire ulike vhosts. Det webhotell-rollen tok seg av var å opprette mapper for SSL-sertifikater og kopiere disse på plass, samt opprette rot-kataloger for hver virtualhost og kopiere en index.html-fil til disse. Til slutt ble Apache-rollen importert. Det hadde vært ønskelig om Apache-rollen selv kunne sørget for å opprette alle nødvendige mapper og kopiere filer, men dette måtte altså gjøres manuelt. I Apache-rollens varibler *apache_vhosts* og *apache_vhosts_ssl* ble de ulike vhosts angitt. Eksempelvis *servername*, *documentroot*, *certificate_file* osv. Apache-rollen hadde ikke parameterisert alle opsjonene i vhost-fila, derfor måtte det benyttes en extra_parameters-variabel der alt som ble angitt etter denne ble tolket bokstavelig, altså slik det faktisk vil vært skrevet i vhost-konfigurasjonen. Dette ble i eksempelet benyttet til *rewrite*-regler (regulære uttrykk) og fungerte som det skulle, selv om det virket litt vel "hackete". Se figur 5.2 for eksempel.

Listing 5.2: Utdrag fra *apache_vhosts_ssl-variablen*

```
---
# vhosts.yml
apache_vhosts_ssl:
- servername: "site1.foo.bar"
  serveralias: "www.site1.foo.bar"
  documentroot: "/var/www/html/site1"
  certificate_file: "{{ apache_path }}/ssl.crt/site1.foo.bar.crt"
  certificate_key_file: "{{ apache_path }}/ssl.key/site1.foo.bar.key"
  extra_parameters: |
    RewriteEngine On
    RewriteCond %{HTTP_HOST} !^www\. [NC]
    RewriteRule ^(.*)$ https://www.%{HTTP_HOST}$1 [R=301,L]
```

De samme vhosts-variablene ble også benyttet til å opprette mapper og kopiere filer ved å angi nøkkelen til det elementet en ville hente ut verdien for og deretter iterere over disse ved hjelp av Ansibles *loop*-funksjon. Dette sparer mange linjer kode hvis en

har flere hundre vhosts.

Filer (SSL-sertifikater og HTML-filer) ble skilt ut fra rollen og lagret et eget sted. Det er mulig å lagre disse direkte i webhotell-rollen, men da går en litt bort fra gjenbruksprinsippet. Vhost-variablene kunne vært definert direkte i group_vars hvis en bare har en webhotelltjener, eller i host_vars, for å ha flere ulike webhotell, men ble skilt ut i en egen variabel-fil og samlet med resten av webhotellets filer (slik det vises i figur 5.3.

Listing 5.3: Eksempel på inkludering av variabel-fil og parameterisering

```
# Under webhotell-play i webservers.yml-playbook
vars_files:
  - "{{ webhotell_root_path }}/vhosts/vhosts.yml"

# I host_vars/webXXXX.foo.bar.yml
# Lokasjonen hvor webhotellets vhosts, filer, certs/keys kopieres fra:

webhotell_root_path: eksterne_data/tjenester/web/webhotell_X
```

Variabelen *“ {{ webhotell_root_path }}”* blir angitt i host_var-filen til *webdeb01* og *webcent01*. Dette knytter altså rett webhotell-konfigurasjon til rett tjener. En annen måte å gjøre dette på ville vært gjennom å gjøre en form for logikk innad i rollen som knytter rett webhotell-konfig til rett tjener. En tredje måte, og kanskje enkleste måte, er å bruke flere *plays*, altså å ha ett play for webhotell-1 og et annet for webhotell-2, ved å angi *webhotell[0]* og *webhotell[1]* som *hosts*-pattern [59]. Da slipper en å parameterisere og angir rett webhotell-konfigurasjon direkte. Det er altså mange måter å oppnå det samme resultatet på. Om en måte er bedre enn en annen kommer helt an på hvilke forutsetninger/behov en har, samt størrelse på oppsett.

5.4.8 Sympa

I Ansible Galaxy finnes det en rolle som heter *Sympa*. Den er imidlertid tom, dvs. kun en rolle-template uten faktisk innhold. Sympa-Community på GitHub har lagd et ferdig generelt Sympa-oppsett som setter opp Sympa på en enkelt-maskin med bruk av Vagrant og Ansible [199]. Rollene som fulgte med dette ble testet og fungerte (med litt triksing av path-variabler), men i et reelt oppsett vil en helst kjøre databasen og webtjenesten på forskjellige maskiner. Sympa-rollene fra Sympa-Community kan brukes som utgangpunkt for å lage sitt eget Sympa-oppsett, der de forskjellige tjenestene Sympa består av blir dekoblet fra det monolittiske oppsettet til Sympa-Community. Eksempelvis ved å lage en Sympa-playbook som bruker en rolle for Sympa, i tillegg til Apache, MySQL og Postfix, til å sette opp en komplett Sympa-tjeneste spredt utover flere noder.

Database

Eksempeloppsettet for Ansible setter også opp to databaseservere. Uten oppsett av Sympa er har ikke disse noen annen funksjon enn å bare være tilstede, samt gi oppsettet en litt større skala og nevnes derfor ikke nærmere.

5.4.9 Anbefalte driftsrutiner og sikkerhet

Drift og overvåkning av Ansible

AWX/Tower gir mange muligheter utover bare tidplanlegging. Her kan det nevnes organisering av team/brukere, autentisering mot katalogtjenester som LDAP/RADIUS m.fl., automatisk henting av konfig fra Git, kryptering av SSH-nøkler, API med mer [27].

Ansible kan også integreres med tredjepartsverktøy som *The Foreman*. Foreman kan f.eks motta facts om noder, tillegne roller til noder og kjøre roller/playbooks [198].

Ansibles funksjonalitet kan utvides med *plugins* [21]. Mange plugins er allerede inkludert som standard. *Callback plugins* kan brukes til å reagere på hendelser fra Ansible-kjøringer og videreformidle outputen [34]. Eksempler på inkluderte plugins er Epost, Slack, Logstash. I tillegg finnes det også tredjepartsplugins, eller en kan skrive det selv.

Oppgradering av Ansible

Siden Ansible ikke er et klient/tjener-oppsett er oppgradering enkelt fordi det er kun kontroll-maskinen som kjører Ansible som må oppgraderes. Dette gjøres enkelt via vanlig pakkehåndtering. For CentOS/RedHat installeres siste versjon fra YUM, mens for Debian/Ubuntu kreves det at en legger til Ansibles pakkebrønn. For å installere en spesifikk versjon anbefaler Ansible å bruke *Python pip* [41]. Det anbefales å følge med på Ansibles utviklingsdokumentasjon for å følge med på nye endringer, spesielt "porting guides" [11].

Oppgradering av styrte tjenester

Oppgradering av en styrt tjeneste kan utarte seg på flere måter. Det kan eksempelvis innbære å oppdatere pakkeversjoner, installere en ny versjon av en rolle eller lage en egen playbook som automatiserer prosessen.

Ansibles APT-/YUM-moduler er litt forskjellige, men begge kan angi ulike tilstander (*state*)til en pakke. "Present" vil si at pakken er installert, mens "latest" betyr at pakken alltid blir oppdatert til siste versjon. En spesifikk versjon av en pakke angis sammen med pakkenavnet, eks. *name: pakkenavn=1.0* (for APT-modulen).

For tjenester som settes opp ved hjelp av roller vil en spesifikk pakkeversjon være avhengig av hvordan rollen fungerer. For *geerlingguy.apache*-rollen som eksempeloppsettet benytter er *state*-variablen parameterisert (*apache_packages_state*) og *present* er angitt som standardverdi. Da er det enkelt å endre denne *latest*.

For å installere en bestemt versjon av en rolle kan en angi versjonsnummer i *requirements.yml*. Roller blir ikke automatisk oppdatert til nyeste versjon hvis en kjører kjører installasjonskommandoen med mindre en bruker *force*-opsjonen [15].

Egne playbooks kan også brukes til å beskrive prosessen med å oppdatere kjørende tjenester. Et interessant poeng i den forbindelse er at det finnes eksempeler på brukere som bruker Puppet til selve konfigurasjonsstyringen, men Ansible til automatiserte vedlikeholdsprosesser [98].

Sikkerhet

Sikkerhet er utvilsomt viktig uansett hvilket CM-system en velger. Som nevnt i den innledende delen bruker Ansible SSH-protokollen til å autentisere tilkoblinger og kryptere kommunikasjonen [110]. SSH er en "industristandard" og må anses å være generelt sikker (så lenge en bruker nøkler på 2048 bits eller mer) [109]. Kontroll-maskinen gir potensielt kontroll over hele infrastrukturen, derfor er det svært viktig å sikre tilgangen til denne.

Ansible kobler til målmaskinene som en vanlig bruker og bruker sudo for å eskalere til root. Denne eskaleringen kan gjøre for et helt play i en playbook, evt. individuelt for hvert enkelt play. Det er også mulig å bruke horisontal privilegie-eskalering til utføre

oppgaver som andre brukere enn root. Hvis målmaskinen krever passord for å bli sudo er det mulig å lagre dette passordet på kontrollmaskinen, gjerne kryptert med Vault.

Bare AWX/Tower har mulighet for tilgangsstyring og autentisering mot katalogtjenester for å kunne styre f.eks hvem som har tilgang til å kjøre hvilke playbooks [37]. Hvis en ikke bryr seg om GUI er det mulig å bruke det sammen med Tower-CLI for å få tilgang til disse funksjonene fra kommandolinjen [52].

Hemmeligheter

Ansible kommer med *Ansible Vault* som en del av installasjonen [29]. Ansible Vault gjør det mulig å kryptere sensitive data, enten på filnivå, eller på variabelnivå (dvs kryptere enkeltverdier i en variabelfil).

Testing

Testing har ikke vært i fokus i denne rapporten, men det finnes mange muligheter. Syntaks kan sjekkes med ansible-playbook-kommandoen med syntax-check-opsjonen for å sjekke om en playbook er kjørbar. *Check*-opsjonen gir mulighet for å kjøre playbook uten å gjøre endringer [35]. Ansible har også moduler som kan brukes til testing slik at testing blir innebygd i playbooks [50].

Det finnes også en del eksterne verktøy som *Ansible-Lint* som sjekker koden for feil [205]. I tillegg er mange andre kjente verktøy, som eks. Test Kitchen (m/Kitchen-Ansible, Serverspec, Travis-CI + Docker-containere, Jenkins, med flere, også mulig å bruke sammen med Ansible [63] [108] [196] [51] [97]. I eksempeloppsett ble det for det meste gjort syntaks-sjekker, men det ble også eksperimentert med Test Kichen (m/Docker-provisjonering).

5.5 Puppet

5.5.1 Maskinene

Alle maskinene i Puppet-oppsettet benyttet Puppets offisielle pakkebrønn for versjon 5-utgivelsen [141].

puppetmaster

Pupppet-serveren ble installert med Ubuntu 16.04 Xenial. Puppet Server ble installert med pakken puppetserver fra pakkebrønnen[2]. Den ble konfigurert med r10k for å hente konfigurasjoner fra kontrollrepoet, og maskinens SSH-nøkkel ble lagt til i BitBucket for at den kunne lese fra repoet. Puppet Server ble konfigurert til å automatisk signerte alle sertifikater fra de styrte maskinene. Dette er ikke anbefalt i et produksjonsmiljø, og ble brukt for enkelhets skyld i dette oppsettet.

De styrte maskinene

De styrte maskinene ble halvt inndelt i 4 maskiner med Ubuntu 16.04 Xenial og 4 maskiner med CentOS 7. Puppet agent ble installert med pakken puppet-agent fra samme pakkebrønn som puppetmaster, men med Yum-pakkebrønnen[3] for CentOS-maskinene. Puppet agent prøver i utgangspunktet å nå puppetmaster med navnet *puppet*, og en binding mellom det navnet og IP-adressen til puppetmaster ble lagt til i `/etc/hosts`-fila.

[2] `https://apt.puppetlabs.com/puppet5-release-xenial.deb`
[3] `https://yum.puppetlabs.com/puppet5/puppet5-release-el-7.noarch.rpm`

5.5.2 Organisering av konfigurasjoner

Tidlig i arbeidet med oppsettet for Puppet ble det oppdaget at SSHd-konfigurasjonen som oppdragsgiver viste til, var vanskelig å oppnå. I starten kunne problemstillingen virke umulig å løse, og det er ikke sikkert om Puppet i det hele tatt vil kunne direkte oversette denne konfigurasjonen slik den ble gjort i CFEngine. En av hovedoppgavene ble å vise hvordan en slik konfigurasjon kan være forskjellig ut fra en enkelt maskin eller en gruppe av maskiners forutsetninger (hvilket operativsystem, hvilken tjeneste de kjører, osv).

Organiseringen av konfigurasjon og konfigurasjonsdata har betydning for hvordan denne problemstillingen håndteres. Det ble bestemt å ta ibruk host-gruppemønstret, i tillegg til rolle-profilmønstret for generelle konfigurasjoner. Begge er beskrevet i 4.2.6. I bunn og grunn er begge basert på bruken av moduler, men de organiseres på litt forskjellig vis. Rolle-profilmønstret er enkelt og intuitivt, men fører til at konfigurasjoner og data i Hiera samles på kun ett sted (alle profil-klasser i profile-modulen, og som regel all data på miljønivået). Host-gruppemønstret grupperer konfigurasjoner ut fra gruppe maskiner eller tjenester i egne moduler, og legger tilhørende data der, i stedet for på miljønivået.

5.5.3 Organisering av data

Hiera er systemet Puppet bruker for å oppbevare og organisere konfigurasjonsdata. Hiera må konfigureres for miljønivået, for hver host-gruppe i respektive `hiera.yaml`-filer, og eventuelt på globalt nivå. I dette oppsettet er miljønivået og modulnivået for hostgrupper konfigurert.

Hierarkiet for miljønivået (`./production/hiera.yaml`):

- Data for enkeltmaskiner
- Data for ulike operativsystemer
- Data for ulike tjenester
- Data for basistjenester
- Data som er felles

Miljønivået inneholder ikke data for “ulike tjenester”, siden dette vil ligge i hver host-gruppemodul, men ble tatt med for å vise hvordan hiera på miljønivået kan organiseres.

Hierarkiet på modulnivå for hostgruppen web (`./production/site/web/hiera.yaml`):

- Data for enkeltmaskiner
- Data for ulike operativsystemer
- Data for webhotell
- Data for annen web-tjeneste

Nest siste linje i listen over viser bare at flere lag kan legges til i hierarkiet ettersom flere web-tjenester blir lagt til. Hvis en ender opp med å ha mange slike undertjenester, kan disse gjerne samles i en egen katalog (eksempelvis `.data/webtjenester/`), men “path” i `hiera.yaml` må byttes ut med “glob” slik at filnavnet kan globbes og Hiera søker gjennom alle filene i denne katalogen.

Listing 5.4: Utdrag fra `./production/hiera.yaml`

```
hierarchy:
  - name: 'Individuelle noder'
    path: "nodes/%{::trusted.certname}.yaml"
```

```
  - name: 'Operativsystem'
    path: "operativsystem/%{facts.operatingsystem}.yaml"
```

Utdraget fra hiera.yaml vist i listing 5.4 viser de to øverste linjene i `hiera.yaml` for miljønivået i oppsettet. Øverst er hver enkelt nodes data plassert (om den har noe). Under er spesifikke data for ulike operativsystemer plassert. Legg merke til at filnavnet blir *interpolert* ut fra sertifikatnavnet maskinen har og hvilket operativsystem den kjører (ved bruk av Facter). I dette hierarkiet er det altså slik at de øverste linjene vil ha høyeste prioritet hvis en variabel med samme navn er definert flere steder. Det er nok også heller uvanlig å angi mange enkeltmaskiner sine egne spesielle data, men det er altså mulig å overstyre oppslag på denne måten.

5.5.4 NTP

NTP-klientkonfigurasjonen i Puppet var den enkleste. Det ble også valgt å vise hvordan tidssone kan stilles. For disse konfigurasjonene ble det valgt å ikke ha konfigurasjonsdata separert i Hiera, siden det består av såpass lite.

NTP-tjenerene som ble brukt var `ntp.ntnu.no`, `ntp.uninett.no`, `0.no.pool.ntp.org` og `1.no.pool.ntp.org`. Noen ekstra `restrict`-parametre ble også lagt til. Tidssonen for operativsystemet ble satt til "Europe/Oslo" og maskinklokken justert til UTC. Klassene for NTP-klientkonfigurasjonen og tidssone ble skrevet i hver sin profilklasse, **profile::basis:ntp** og **profile::basis::timezone**. Disse klassene ble ganske små, og kunne antakeligvis blitt kombinert i én og samme profilklasse kalt **profile::basis::tid** eller lignende.

5.5.5 SSH

"Saz" sin Puppet-modul ble brukt for SSH-konfigurasjon i dette oppsettet [207]. SSHD-innstillinger blir deklarert i klassen **ssh::server** i `options`-parametret. I Hiera ble det bestemt at hash-variablen **ssh::server_options** skulle oppbevare de ulike innstillingene (ref. 5.2 angående diskusjon om ulike innstillinger for ulike grupper av maskiner).

Det ble først prøvd å lagre denne variablen flere plasser i hierarkiet for miljønivået og bruke sammenslåing (*merging*) [118]. Sammenslåing skulle eksempelvis kunne variere på OS-familie og den styrte tjenesten for en maskin. Det viste seg å være vanskelig å bruke samme navn på denne variablen for ulike tjenester. Eksempelvis hvis en gruppe web-tjenere skal ha noen særegne innstillinger, kunne ikke **ssh::server_options** plasseres i f.eks `./data/tjenester/web.yaml`. Da vil den slås sammen med samme variabel deklarert i datafilen for eksempelvis e-posttjenere i `./data/tjenester/mail.yaml`.

En egen variabel for hver tjeneste, eksempelvis **web::ssh::server_options**, kan i stedet deklareres. Da kreves også egne SSH-manifester, siden oppslaget må tilpasses dette navnet. To oppslag kan gjøres, eksempelvis for generelle og tjenestespesifikke innstillinger hver for seg. Disse oppslagene kan så slås sammen til bruk i samme **ssh::server**-klasse.

Det som uansett ble valgt var å bruke hostgrupper der hver hostgruppe har sitt eget manifest for SSH-konfigurasjon. Spesielle data om SSH for denne gruppen maskiner vil finnes i dens modul, mens generelle data finnes på miljønivået. To separate oppslag må gjøres, men de slås sammen i manifestet ved bruk av plusstegn-operatoren mellom variabler. Eksempel på dette kan ses i listing E.1 for hostgruppen *web*.

5.5.6 Brannmur

Ved bruk av Puppets egen *firewall*-modul kan brannmuren, det vil si Iptables, styres [159]. Puppet anbefaler, ved bruk av denne modulen, å legge til før- og etterregler for å tillate ulike ting (slikt som *established*), og til slutt blokkere all annen frafikk som ikke matcher noen andre regler.

Klassen `::profile::firewall` sørger for at brannmuren blir installert og legger til før- og etterreglene for alle andre regler. Før- og etterreglene er deklarert i hver sin klasse: `::profile::firewall::pre` og `::profile::firewall::post`. Øvrige regler deklareres i sine egne klasser med navn på tjenesten som åpnes for, eksempelvis

`::profile::firewall::smtp`.

For å angi reglene, listes disse klassene i rollen for en tjeneste, som vist i listing 5.5.

Listing 5.5: Eksempel på deklarasjon av brannmur i rollen `role::webtest1`

```
# Brannmuroppsett
# For- og etterregler inkludert i ::profile::firewall
class { '::profile::firewall': } ->
class { '::profile::firewall::ssh': } ->
class { '::profile::firewall::http': }
```

5.5.7 Web-server

Styring av Apache-moduler

Puppet-modulen Apache inkluderer en klasse kalt **apache::mod** og underklasser med navn på ulike Apache-moduler for å kunne styre disse. Eksempel på en Apache-modul med en egen klasse er **apache::mod::ssl**, som også brukes i dette oppsettet. Hvis en klasse ikke eksisterer for en ønsket Apache-modul, kan den *definerte typen* **apache::mod** benyttes, som har ulike parametre for installere pakken til en Apache-modul. Eksemplet i listing 5.6 viser hvordan Apache-moduler kan deklareres (installeres og slås på), samt angi konfigurasjon. Hvis en modul ikke er deklarert, vil den ikke være installert eller påslått.

Listing 5.6: Eksempel på bruk av **apache::mod**

```
include apache::mod::cgi

class { "apache::mod::ssl":
  ssl_compression => true,
}
```

Webhotell med mange Virtualhosts

Listen over virtualhosts som ble sendt over til prosjektgruppen var meget lang. Det tydet på at å skrive ett Puppet-manifest som deklarerte hver enkelt virtualhost ville antakeligvis vært svært lang og vanskelig å vedlikeholde. Her kommer Hiera inn igjen. Puppet-modulen for Apache har en definert type **apache::vhost** [157]. Det ble først vurdert å bruke iterering for å generere flere av denne typen [122].

Ikke lenge etter ble klassen apache::vhosts funnet (merk flertall). Denne har et parameter kalt *vhosts* som aksepterer en hash-verdi. Denne hash-verdien kan bli oppbevart i hiera, og konfigurasjonen for webhotellet blir kort og grei. Listing 5.7 viser hvordan denne klassen ble benyttet i dette oppsettet. Listing E.2 viser to virtualhosts plassert i Hiera.

Listing 5.7: Utdrag fra `./production/site/web/manifests/webhotell.pp`

```
# Oppretter vhoster vha. apache::vhosts-klassen
class { "apache::vhosts":
  vhosts => lookup('web::webhotell::vhosts'),
}
```

Hash-variablen **web::webhotell:vhosts** inneholder altså alle virtualhosts maskinen for webhotellet trenger. Nye virtualhosts kan enkelt legges til under.

5.5.8 Sympa

En Sympa-modul finnes, men som nevnt var den gammel og utvikleren har fjernet koden fra GitHub og seg selv [101]. På grunn av dens alder ble det besluttet å ikke bruke modulen. Derimot ble det bestemt å gjøre et forsøk på manuelt å installere tjenestene som i tillegg utgjør et oppsett av Sympa (Apache, MySQL, Postfix). Målet med oppsettet av Sympa vil i hovedsak være å vise hvordan ulike deler av konfigurasjonen kan deles opp i forskjellige manifester.

Først ble en modul for hostgruppen *sympa* opprettet ved bruk av kommandoen `puppet module generate ntnu-sympa` på konfigklient-maskinen som hadde Puppet installert. Å gjøre dette er ikke strengt tatt nødvendig; katalogene kan bare opprettes og Puppet vil automatisk søke i disse uansett.

Manifestene og klassene som ble opprettet var som følger: `init.pp` for **sympa**, `service.pp` for **sympa::service**, `package.pp` for **sympa::package** og `config.pp` for **sympa::config**. Hvert manifest gjør hver sin ting for oppsetett, og bør inkluderes i hovedklassen **sympa** (`init.pp` for å kjøres.

Mot slutten av prosjektarbeidet ble det tydelig at arbeidet med Sympa ville ta lengre tid enn anntatt. Dette arbeidet ble lagt på is, og annet arbeid ble prioritert. Likevel håpes det at strukturen på manifestene som er skrevet, vil være til eksempel på hvordan et reelt oppsett kan gjøres ved å installere nødvendige pakker, kopiere konfigurasjoner og styre systemd-tjenester. Hvis eventuelle kommandoer må kjøres kan `exec`-ressursen i Puppet benyttes, men det anbefales å være sparsommelig i bruken av denne.

5.5.9 Anbefalte driftsrutiner og sikkerhet

Drift og overvåking av Puppet

Overvåking

Graphite kan settes opp til å kommunisere med Puppet server sitt "Metrics API" [134]. Foreman kan benyttes for styring overvåking av Puppet server, blant annet ved å være ENC for den [135]. Foreman har også mulighet for varslinger.

Oppgradering av Puppet

Puppet server og Puppet agent på maskinene kan i seg selv oppgraderes ved bruk av manifester. Puppet har skrevet en egen modul for styring av Puppet agent [161]. Når Puppet server installeres, gjøres dette fra en pakkebrønn av en gitt utgivelse. I dette oppsettet ble APT- og Yum-pakkebrønnene for versjon 5-utgivelsen brukt. Ved å bruke Puppets offisielle modul for håndtering av APT- eller Yum-pakkebrønner kan den ønskede utgivelsen av Puppet angis [158] [165]. Eksempler på manifest for Puppet server er tatt med (men hverken brukt eller testet) i eksempeloppsettet.

Oppgradering av styrte tjenester

De styrte tjenestene kan via Puppet hovedsaklig oppgraderes på to måter. Den første måten er å oppgradere Puppet-modulen, ved å spesifisere det nye versjonsnummeret i Puppetfile.

Den andre måten å oppgradere tjenester på, eller å sikre at en bestemt versjon skal være installert, er å angi ønsket versjon for *ensure*-parametret i pakkeressursen for en bestemt pakke som styres [168]. Bruk eksempelvis kommandoen `apt show` *pakkenavn* brukes for å vise det korrekte versjonsnummeret for APT-baserte pakker. Ved bruk av kommandoen `puppet resource package` *pakkenavn* kan den eksakte tilstanden til eksempelvis en installert pakke uttrykkes i Puppet-kode.

Sikkerhet

Det er viktig å sikre Puppet-systemet, og da spesielt Puppet serveren, siden den er sertifikatutsteder for store deler av infrastrukturen [1]. Åpen kildekode-versjonen av Puppet ser ikke ut til å ha mulighet for tilgangsstyring til maskinen gjennom eksempelvis LDAP. Hvis eksempelvis Foreman benyttes, vil disse kunne gjøre tilgangsstyring.

Git-repoet eller repoene bør også sikres. Som nevnt kan host-grupper skrives som egne moduler. Disse modulene kan bli utviklet i hvert sitt repo som bare utvalgte folk har tilgang på. Dette er samme mønster som CERN benytter [86]. For å samle alle modulene fra hvert sitt repo, kan Puppet serveren ved hjelp av modulen *vcsrepo* for å hente ned alle repoer til en katalog på maskinen [162]. For at Puppet server skal kunne se disse modulene fra en annen katalog enn de den er vant til i utgangspunktet, må *modulepath*-variablen for miljøet endres i `environment.conf` [120].

Hemmeligheter

Passord for ulike tjenester bør ikke skrives eksplisitt i hverken kode eller i Hiera. For dette har Puppet en egenutviklet metode for å kryptere data [116]. Den omtales som en *backend* for datakilde. Nøkler må opprettes for å kunne kryptere enkeltdata. HashiCorps Vault kan også brukes for å lagre hemmeligheter i en kryptert database [91]. Hemmelighetene må da i manifester hentes ut ved bruk av oppslag mot Vault-databasen.

5.6 Sammenligning og diskusjon

Eksempeloppsettet for både Ansible og Puppet viser det er fult mulig å etablere og styre tjenester gjennom begge. Det eksisterer likevel noen vesentlige forskjeller i både arbeidsflyt, anvendelse, roller og moduler og arkitektur.

5.6.1 Forskjell i arbeidsflyt og anvendelse

Slik Ansible fungerer, kan det kort beskrives som et system for utføring av sekvensielle oppgaver (derav *tasks* i playbooks). Som nevnt flere steder tidligere i denne rapporten, er Ansible skapt for å være simpelt. Tidlig anvendelse er enklere med Ansible. Puppet krever mer etablering i infrastrukturen, og har kanskje en høyere terskel for forståelse og anvendelse. Puppet beskriver i større grad ønsket tilstand med litt mer abstraksjon enn Ansible. Puppet har færre innebygde funksjoner, og støtter seg mer på bruk av moduler.

5.6.2 Arkitektur

Skalering kan være en utfordring i Ansibles arkitektur. Det finnes som nevnt løsninger for distribuering av Ansible-maskiner, men er ikke offisielt støttet. Det er usikkert om eksempelvis 500 maskiner kan være en utfordring med Ansible. Problemet ikke nødvendigvis kapasitet på Ansible-maskinen, men tid brukt hvis en kjøring skal inkludere alle maskiner i infrastrukturen. Når kjøringer gjøres, er bestemt av administratoren.

Puppet skalerer bedre i og med at alle styrte maskiner automatisk bestemmer når de skal hente konfigurasjon. Puppet servere kan som nevnt også lastbalanseres om det er nødvendig.

5.6.3 Galaxy-roller vs Puppet Forge-moduler

Ansible Galaxy har færre roller enn Puppet Forge. Galaxy har heller ingen offisielle roller, og inntrykket er at kvaliteten på en del av de er varierende. Det er likevel enkelt å skrive roller selv. Puppet Forge har mange moduler, en del av de skrevet av Puppet selv, og kvalitet på disse modulene er som regel bra.

5.6.4 Fornyelse av kode (*refactoring*)

Både Ansible og Puppet blir stadig utviklet med ny funksjonalitet. En del funksjonalitet og aspekter ved deklarasjonsspråkene endres også. Det er dermed viktig å følge med på alle nye utgivelser og stadig fornye sin egen kode.

6 Avslutning

Funnene i denne rapporten viser at alle de vurderte kandidatene vil kunne gjøre konfigurasjonsstyring av Linux-maskiner. Eksempeloppsettet viste mer detaljert hvordan Ansible og Puppet kan brukes. Det som skiller kandidatene er måten de fungerer på og anvendes, og krever litt ulik tankegang hver for seg. Uansett hvilket system man bruker, vil man komme i mål med samme resultater.

Prosjektgruppen har formulert en kort beskrivelse for Ansible og Puppet: Ansible er enklere å anvende, og tankegangen baserer seg på sekvensiell utføring av oppgaver. Puppet er en mer komplett CM-løsning, har et høyere nivå av abstraksjon og krever tydeligere etablering i infrastrukturen.

Basert på funnene i denne rapporten, anbefaler prosjektgruppen Puppet som erstatter for CFEngine. Første argument er at CFEngine og Puppet har ganske lik arkitektur. Andre argument er at kunnskap om Puppet andre IT-miljøer ved NTNU. Tredje argument er at Puppets fartstid og bruk i bransjen også viser det har modnet til et mer komplett system, og løser det meste en skulle ønske.

6.1 Videre arbeid

Uansett hvilken kandidat Unix-gruppen velger ønsker prosjektgruppen å foreslå hvordan de kan gå fram etter at et valg er tatt.

Finn egne behov

Få full oversikt over det som styres og hvilke begrensninger som finnes.

Les litteratur og test systemet

Prosjektgruppen anbefaler å lese bøker og begynnerveiledninger for nyeste versjon av systemet. Offisiell dokumentasjon og referansemanualer dekker også mye av det som er nødvendig for å komme i gang. Videoer (YouTube, e.l.) kan også være av stor nytte.

Legge plan for gradvis overgang

Det kan være enklere å se for seg en gradvis overgang til det nye systemet. Fokuser på én tjeneste av gangen, og ta med erfaring i påfølgende oppsett av andre tjenester. Bestem også etter hvert nødvendige retningslinjer for bruk og organisering.

Start med blanke ark og tenk nytt!

Det kan være fort gjort å gå i den fellen å bare oversette de gamle CFEngine-filene til det nye språket. Det anbefales å følge det valgte systemets tenkemåter/best-practice-prinsipper. Spesielt anbefales det å starte å benytte moduler som gjør konfigurasjonen for deg, i stedet for å bare kopiere konfigurasjonsfiler på plass. Det anbefales også at basistjenester (og andre generelle aspekter), som SSH, standardiseres for hele infrastrukturen.

Andre implementasjoner

Gruppen anbefaler å utføre automatisert testing av kode, gjerne gjennom en pipeline. Dette kan bidra til å kvalitetssikre kode før den implementeres. Det burde og tas i bruk en CMDB for å holde på informasjon om infrastrukturen. En slik database vil eksempelvis kunne brukes med ekstern nodeklassifisering.

Refactoring

Sett av litt tid og resursser til å stadig forbedre og fornye kode. I løpet av prosessen vil en også oppdage/lære nye muligheter med systemene, noe som åpner for forbedringsmuligheter for gammel kode. Hvis en hele tiden klarer å gjøre litt refactoring vil en unngå stort etterslep.

6.2 Evaluering av arbeidet

Organisering

Alle i prosjektgruppen er fornøyde med organiseringen av prosjektet. Oppmøte har vært konsistent, og ingen uventede avvik. Noen nevner at det kunne vært større muligheter for eget arbeid. I perioder kunne gruppen arbeidet mer effektivt, men likevel har prosjektet hatt god kontinuitet.

Prosessen

Alle i prosjektgruppen er enige om at utvalgsprosessen kunne tatt kortere tid. Utvalget av de fire kandidatene ble uansett det vi antok før arbeidet startet. Kanskje kunne arbeidet ha startet rett på vurdering, slik at det ble mer tid til arbeid med eksempeloppsett.

Arbeidsfordeling

I arbeidet med eksempeloppsett ble ansvar og arbeid i hovedsak fordelt i to. Det endte med at kunnskap om de andres arbeid i mindre grad enn ønsket ble tilegnet. Gruppen burde sånn sett jobbet mer sammen om Ansible, Puppet og maskinressursene i SkyHiGh.

Måloppnåelse

Oppsett av Sympa kom prosjektgruppen dessverre ikke i mål med. Det håpes likevel at noe av arbeidet kan være til inspirasjon. Ellers er det meste av planen gjennomført og omtrent slik det ble tidsplanlagt.

Læringsutbytte

Alle i prosjektgruppen er fornøyde med læringsutbytte. Langt større erfaring og kunnskap er ervervet, både om CM-systemer, men ikke minst om prosjektarbeid. Som nevnt ble ikke alle kjente med de ulike systemene. Likevel uttrykker alle at de er inspirerte til å lære om hverandres arbeid på egenhånd.

6.3 Konklusjon

Hva er det beste konfigurasjonssystemet for Unix-gruppen ved NTNUs IT-avdeling? Prosjektgruppen mener Puppet er den beste kandidaten. Det finnes likevel andre gode kandidater, men som nevnt har de ulikt bruksområde og anvendelse. Kanskje kan både Ansible og Puppet anvendes, men for forskjellige oppgaver. Gruppen er fornøyd med resultatene, og håper beskrivelsene og eksemplene er tilstrekkelig som en faglig vurdering.

Bibliografi

[1] ALFKE, M. F. F. *Puppet 5 Essentials*, 3 ed. Packt Publishing Ltd., 2017.

[2] ANSIBLE. Ansible 0.01 [Internett]. San Francisco, USA: GitHub Inc.; 08.03.2012 [16.05.2018] Tilgjengelig fra https://github.com/ansible/ansible/releases/tag/0.01/.

[3] ANSIBLE. Ansible 2.0 Porting Guide [Internett]. Durham, NC, USA: Red Hat Inc.; [25.04.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/porting_guides/porting_guide_2.0.html#deprecated.

[4] ANSIBLE. Ansible All Modules [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/list_of_all_modules.html.

[5] ANSIBLE. Ansible apache2_module module [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/apache2_module_module.html.

[6] ANSIBLE. Ansible best practices [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/user_guide/playbooks_best_practices.html.

[7] ANSIBLE. Ansible Blog [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/blog.

[8] ANSIBLE. Ansible Configuration hash behaviour [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/config.html#default-hash-behaviour.

[9] ANSIBLE. Ansible configuration settings [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/config.html#ansible-configuration-settings.

[10] ANSIBLE. Ansible Docs [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/.

[11] ANSIBLE. Ansible Docs, Porting guides [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/devel/porting_guides/porting_guides.html.

[12] ANSIBLE. Ansible Engine [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/products/engine.

[13] ANSIBLE. Ansible FAQ [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/faq.html.

[14] ANSIBLE. Ansible Galaxy [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/.

[15] ANSIBLE. Ansible Galaxy [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/galaxy.html.

[16] ANSIBLE. Ansible Get Gtarted [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/resources/get-started.

[17] ANSIBLE. Ansible GitHub [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/ansible/ansible.

[18] ANSIBLE. Ansible IPtables module [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/iptables_module.html.

[19] ANSIBLE. Ansible loop [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/user_guide/playbooks_loops.html.

[20] ANSIBLE. Ansible modules [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/ansible/ansible/tree/devel/lib/ansible/modules.

[21] ANSIBLE. Ansible Plugins [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/plugins/plugins.html.

[22] ANSIBLE. Ansible pull [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/cli/ansible-pull.html.

[23] ANSIBLE. Ansible Python3 support [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/devel/reference_appendices/python_3_support.html.

[24] ANSIBLE. Ansible Releases [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://releases.ansible.com/ansible/.

[25] ANSIBLE. Ansible Resources [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/resources.

[26] ANSIBLE. Ansible setup module [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/setup_module.html.

[27] ANSIBLE. Ansible Tower [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/products/tower.

[28] ANSIBLE. Ansible Tower Requirements [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible-tower/3.2.4/html/installandreference/requirements_refguide.html.

[29] ANSIBLE. Ansible Vault [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/vault.html.

[30] ANSIBLE. Ansible Webinars and training [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/resources/webinars-training.

[31] ANSIBLE. Ansible.com [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/.

[32] ANSIBLE. Automation with Ansible 1 [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.redhat.com/en/services/training/do407-automation-ansible-i.

[33] ANSIBLE. AWX GitHub [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/ansible/awx.

[34] ANSIBLE. Callback plugins [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/plugins/callback.html.

[35] ANSIBLE. Check Mode [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_checkmode.html.

[36] ANSIBLE. Configuration management [Internett]. Durham, NC, USA: Red Hat Inc.; [25.04.2018]. Tilgjengelig fra https://www.ansible.com/use-cases/configuration-management.

[37] ANSIBLE. Configure Authentication [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible-tower/3.2.4/html/quickstart/configure_authentication.html.

[38] ANSIBLE. Creating reusable playbooks [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_reuse.html#creating-reusable-playbooks.

[39] ANSIBLE. Debug module [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/debug_module.html.

[40] ANSIBLE. How Ansible works [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/overview/how-ansible-works.

[41] ANSIBLE. Installation guide [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/installation_guide/intro_installation.html.

[42] ANSIBLE. Lookup plugins [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://docs.ansible.com/ansible/2.5/plugins/lookup.html.

[43] ANSIBLE. Playbook strategies [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_strategies.html.

[44] ANSIBLE. Playbooks templating [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_templating.html.

[45] ANSIBLE. Release and maintenance [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/release_and_maintenance.html.

[46] ANSIBLE. Remote Connection Information [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/intro_getting_started.html#remote-connection-information.

[47] ANSIBLE. Roles [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_reuse_roles.html.

[48] ANSIBLE. Task lists [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_intro.html.

[49] ANSIBLE. Template module [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/modules/template_module.html.

[50] ANSIBLE. Test strategies [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/test_strategies.html.

[51] ANSIBLE. Testing Ansible roles with Docker [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/blog/testing-ansible-roles-with-docker.

[52] ANSIBLE. Tower CLI [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible-tower/3.2.4/html/towerapi/towercli.html.

[53] ANSIBLE. Use Cases [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/use-cases.

[54] ANSIBLE. Using variables about Jinja2 [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_variables.html#using-variables-about-jinja2.

[55] ANSIBLE. Variable precedence [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable.

[56] ANSIBLE. Working with dynamic inventory [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/intro_dynamic_inventory.html#intro-dynamic-inventory.

[57] ANSIBLE. Working with inventory [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/intro_inventory.html.

[58] ANSIBLE. Working with Patterns [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/intro_patterns.html.

[59] ANSIBLE. Working with Patterns [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/user_guide/intro_patterns.html.

[60] ANSIBLE. YAML syntax [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra http://docs.ansible.com/ansible/2.5/reference_appendices/YAMLSyntax.html.

[61] BENSON, J. O., PREVOST, J. J., AND RAD, P. Survey of automated software deployment for computational and engineering research. In *2016 Annual IEEE Systems Conference (SysCon)* (April 2016), pp. 1–6. doi:10.1109/SYSCON.2016.7490666.

[62] CADE METZ. The Chef, the Puppet, and the Sexy IT Admin [Internett]. San Francisco, CA: Wired; 10.26.11 [20.04.2018] Tilgjengelig fra https://www.wired.com/2011/10/chef-and-puppet/.

[63] CHEF. Test Kitchen [Internett]. Seattle, USA: Chef Software Inc.; [16.05.2018] Tilgjengelig fra https://kitchen.ci/.

[64] CHEF. Chef architecture [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/chef_overview.html.

[65] CHEF. Chef archive [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs-archive.chef.io.

[66] CHEF. Chef authentication [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/auth.html.

[67] CHEF. Chef automate [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://www.chef.io/automate/.

[68] CHEF. Chef Cookbook [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/cookbooks.html.

[69] CHEF. Chef courses [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://training.chef.io/.

[70] CHEF. Chef customers [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://www.chef.io/customers/.

[71] CHEF. Chef Databags [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/data_bags.html.

[72] CHEF. Chef documentation [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/.

[73] CHEF. Chef forum [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://discourse.chef.io/.

[74] CHEF. Chef github[Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://github.com/chef/chef.

[75] CHEF. Chef hevder å kontrollere[Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/server_components.html.

[76] CHEF. Chef learning [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://learn.chef.io.

[77] CHEF. Chef Ohai [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/ohai.html.

[78] CHEF. Chef partners [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://www.chef.io/partners/directory/.

[79] CHEF. Chef podcast [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra http://foodfightshow.org/blog/archives/.

[80] CHEF. Chef release date [Internett]. Seattle, WA: 15.01.2009 Chef inc.; [25.04.2018]. Tilgjengelig fra https://blog.chef.io/2009/01/15/announcing-chef/.

[81] CHEF. Chef release notes [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/release_notes_server.html.

[82] CHEF. Chef runtimedocs[Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://docs.chef.io/about_chefdk.html.

[83] CHEF. Chef supermarket [Internett]. Seattle, WA: Chef inc.; [25.04.2018]. Tilgjengelig fra https://supermarket.chef.io/.

[84] CONTRIBUTORS, W. Comparison of open-source configuration management software. Besøkt: 26.04.2018.

[85] DAVID WILSON. Mitogen for Ansible [Internett]. London, UK: David Wilson; [14.05.2018] Tilgjengelig fra http://mitogen.readthedocs.io/en/latest/ansible.html.

[86] EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH. Hostgroups and git repos [Internett]. Genève: CERN; [19.04.2018] Tilgjengelig fra https://configdocs.web.cern.ch/configdocs/details/hostgroups.html.

[87] EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH. Modules and git repositories [Internett]. Genève: CERN; [19.04.2018] Tilgjengelig fra https://configdocs.web.cern.ch/configdocs/details/modules.html.

[88] GARTNER. Business Process Automation (BPA) [Internett]. Stamford, CT: Gartner; [25.04.2018] Tilgjengelig fra https://www.gartner.com/it-glossary/bpa-business-process-automation/.

[89] GROOVER, MIKELL P. *Fundamentals of modern manufacturing : materials, processes, and systems*, 4th ed. ed. Wiley, 2010.

[90] GUTTAG, J. V. *Introduction to Computation and Programming Using Python*. The MIT Press, 2013.

[91] HASHICORP. Vault by HashiCorp [Internett]. San Francisco, CA: HashiCorp; [21.04.2018] Tilgjengelig fra https://www.vaultproject.io/.

[92] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO Standard nr. 10007. Quality management - Guidelines for configuration management [Internett]. Genève: ISO; mars 2017. [24.04.2018]. Tilgjengelig fra https://www.iso.org/obp/ui/#iso:std:iso:10007:ed-3:v1:en.

[93] JAMES CAMMARATA. Ansible 2.0 has arrived [Internett]. Durham, NC, USA: Red Hat Inc.; [25.04.2018] Tilgjengelig fra https://www.ansible.com/blog/ansible-2.0-launch.

[94] JEFF GEERLING. geerlingguy.apache [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/geerlingguy/apache/.

[95] JEFF GEERLING. geerlingguy.firewall [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/geerlingguy/firewall/.

[96] JEFF GEERLING. geerlingguy.ntp [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/geerlingguy/ntp/.

[97] JENKINS. Jenkins [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/jenkinsci/jenkins.

[98] JIMMY OLSEN. Using Ansible for system updates [Internett]. Oslo, NO: Redpill Linpro; 24.12.2017 [14.05.2018] Tilgjengelig fra https://www.redpill-linpro.com/sysadvent/2017/12/24/ansible-system-updates.html.

[99] JUSTIN ELLNGWOOD. How to Use Ansible Roles to Abstract your Infrastructure Environment [Internett]. New York, NY, USA: Red Hat Inc.; 11.02.2014 [14.05.2018] Tilgjengelig fra https://www.digitalocean.com/community/tutorials/how-to-use-ansible-roles-to-abstract-your-infrastructure-environment.

[100] KURSGUIDEN. Kurs: DO407 - Automation with Ansible [Internett]. Fornebu, NO: Kursguiden; [12.05.2018] Tilgjengelig fra https://web.archive.org/web/20180512143817/https%3A%2F%2Fwww.kursguiden.no%2Fkurs%2FRed-Hat-Linux%2FDO407-Automation-with-Ansible%2F.

[101] MCNICOL, D. dmcnicks/sympa [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/dmcnicks/sympa.

[102] MICHAEL DEHAAN. Ansible performance tuning [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://www.ansible.com/blog/ansible-performance-tuning.

[103] MICHAEL DEHAAN. Ansibleworks Galaxy is now available [Internett]. Durham, NC, USA: Red Hat Inc.; 19.12.2013 [25.04.2018] Tilgjengelig fra https://www.ansible.com/blog/2013/12/19/ansibleworks-galaxy-is-now-available.

[104] MICHAEL DEHAAN. The origins of Ansible [Internett]. Durham, NC, USA: Red Hat Inc.; 08.12.2013 [25.04.2018] Tilgjengelig fra https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible.

[105] MIKE GLEASON JR. mikegleasonjr.firewall [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/mikegleasonjr/firewall/.

[106] MORRIS, K. *Infrastructure As Code: Managing Servers in the Cloud*, 1st ed. O'Reilly Media, Inc., 2016.

[107] MYERS, C. *Learning SaltStack*, 2 ed. Packt Publishing Ltd., 2016.

[108] NEILL TURNER. Kitchen Ansible [Internett]. San Francisco, USA: GitHub Inc.; [14.05.2018] Tilgjengelig fra https://github.com/neillturner/kitchen-ansible.

[109] OPENSSH. SSH security. Waltham, MA, USA: SSH Communications Security, Inc.; [14.05.2018] Tilgjengelig fra https://man.openbsd.org/ssh-keygen.

[110] OPENSSH. What is SSH. Waltham, MA, USA: SSH Communications Security, Inc.; [14.05.2018] Tilgjengelig fra https://www.openssh.com/.

[111] OŔEILLY. Oŕeilly Learning Chef [Internet]. Sebastopol, CA: Oŕeilly; [25.04.2018]. Tilgjengelig fra http://shop.oreilly.com/product/0636920032397.do.

[112] PACKT PUBLISHING. Ansible search [Internett]. Birmingham, UK: Packt Publishing Ltd.; [14.05.2018] Tilgjengelig fra https://search.packtpub.com/?query=ansible.

[113] PRESSMAN, R. S. *Software engineering : a practitioner's approach*, 8 ed. McGraw-Hill, 2015.

[114] PUPPET. About environments [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/environments_about.html.

[115] PUPPET. About Hiera [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/hiera_intro.html.

[116] PUPPET. Configuring Hiera [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/hiera_config_yaml_5.html.

[117] PUPPET. Core components [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/products/platform/core-components.

[118] PUPPET. Creating and editing data [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/hiera_merging.html.

[119] PUPPET. Directories: The main manifest [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/dirs_manifest.html.

[120] PUPPET. Directories: The modulepath (default config) [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/dirs_modulepath.html.

[121] PUPPET. Facter: Core Facts [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/facter/3.10/core_facts.html.

[122] PUPPET. Iteration and loops [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_iteration.html.

[123] PUPPET. Language: About values and data types [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_data.html.

[124] PUPPET. Language: Basics [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_summary.html.

[125] PUPPET. Language: Conditional statements and expressions [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_conditional.html.

[126] PUPPET. Language: Data types: Resource and class references [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_data_resource_reference.html.

[127] PUPPET. Language: Namespaces and autoloading [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_namespaces.html.

[128] PUPPET. Language: Node definitions [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_node_definitions.html.

[129] PUPPET. Language: Relationships and ordering [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_relationships.html.

[130] PUPPET. Language: Variables [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/lang_variables.html.

[131] PUPPET. Looking up data with Hiera [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/hiera_automatic.html.

[132] PUPPET. Managing code with r10k [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/pe/2017.3/code_management/r10k.html.

[133] PUPPET. Managing environments with a control repository [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/pe/2017.3/code_management/control_repo.html.

[134] PUPPET. Monitoring Puppet Server metrics [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppetserver/5.3/puppet_server_metrics.html.

[135] PUPPET. Monitoring Puppet Server metrics [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://theforeman.org/.

[136] PUPPET. Overview of Puppet's architecture [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/architecture.html.

[137] PUPPET. PE and open source version numbers [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/pe/2017.3/overview/pe_and_open_source_version_numbers.html.

[138] PUPPET. Puppet - Our Company [Internett]. Portland, OR: Puppet; [20.04.2018] Tilgjengelig fra https://puppet.com/company.

[139] PUPPET. Puppet - YouTube [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://www.youtube.com/user/PuppetLabsInc.

[140] PUPPET. Puppet 3.8.x to 5.x: Get upgrade-ready [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.0/upgrade_major_pre.html.

[141] PUPPET. Puppet 5.0 Release Notes [Internett]. Portland, OR: Puppet; [20.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.0/release_notes.html.

[142] PUPPET. Puppet 5.5 reference manual [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/index.html.

[143] PUPPET. Puppet Community - Find help and get involved [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/community.

[144] PUPPET. Puppet Documentation (5.5): Module fundamentals. Besøkt: 19.04.2018.

[145] PUPPET. Puppet Documentation [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs.

[146] PUPPET. Puppet Enterprise [Internett]. Portland, OR: Puppet; [20.04.2018] Tilgjengelig fra https://puppet.com/products/puppet-enterprise.

[147] PUPPET. Puppet Forge: example42 [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/example42.

[148] PUPPET. Puppet Forge: ghoneycutt [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/ghoneycutt.

[149] PUPPET. Puppet Forge [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/.

[150] PUPPET. Puppet Forge: puppet [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppet.

[151] PUPPET. Puppet Forge: puppetlabs [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs.

[152] PUPPET. Puppet Forge: saz [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/saz.

[153] PUPPET. Puppet Learning VM - Try Puppet in a downloadable virtual machine with quests [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/download-learning-vm.

[154] PUPPET. Puppet master default RAM [Internett]. Portland, OR: Puppet; [24.04.2018] Tilgjengelig fra https://puppet.com/docs/puppetserver/5.3/install_from_packages.html.

[155] PUPPET. Puppet Products [Internett]. Portland, OR: Puppet; [20.04.2018] Tilgjengelig fra https://puppet.com/products.

[156] PUPPET. PuppetDB 5.2 overview [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppetdb/5.2/index.html.

[157] PUPPET. puppetlabs/apache [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/apache.

[158] PUPPET. puppetlabs/apt [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/apt.

[159] PUPPET. puppetlabs/firewall [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/firewall.

[160] PUPPET. puppetlabs/ntp [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/ntp.

[161] PUPPET. puppetlabs/puppet_agent [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/puppet_agent.

[162] PUPPET. puppetlabs/vcsrepo [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppetlabs/vcsrepo.

[163] PUPPET. Puppet's services: Puppet agent on *nix systems [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/services_agent_unix.html.

[164] PUPPET. Puppet's Services: Puppet Server [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppetserver/5.3/services_master_puppetserver.html.

[165] PUPPET. puppet/yum [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/puppet/yum.

[166] PUPPET. Questions - Ask Puppet: Puppet DevOps QandA Community [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://ask.puppet.com/.

[167] PUPPET. Releases - puppetlabs/puppet [Internett]. Portland, OR: Puppet; [20.04.2018] Tilgjengelig fra https://github.com/puppetlabs/puppet/releases.

[168] PUPPET. Resource Type: package [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/types/package.html.

[169] PUPPET. Resources [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/resources.

[170] PUPPET. Roles and profiles: Introduction [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/pe/2017.2/r_n_p_intro.html.

[171] PUPPET. Scaling Puppet Server with compile masters [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppetserver/5.3/scaling_puppet_server.html.

[172] PUPPET. Writing external node classifiers [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://puppet.com/docs/puppet/5.5/nodes_external.html.

[173] RED HAT. Red Hat press release [Internett]. Durham, NC, USA: Red Hat Inc.; 16.10.2015 [25.04.2018] Tilgjengelig fra https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible.

[174] RED HAT. Red Hat press release [Internett]. Durham, NC, USA: Red Hat Inc.; [25.04.2018] Tilgjengelig fra https://www.redhat.com/en/about/press-releases/red-hat-advances-enterprise-and-network-automation-new-ansible-offerings.

[175] RIGHTSCALE. State of the cloud Report [Internett], 2014. Santa Barbara, CA: Rightscale; 02.04.2014 [09.05.2018] Tilgjengelig fra https://assets.rightscale.com/uploads/pdfs/RightScale-2014-State-of-the-Cloud-Report.pdf.

[176] RIGHTSCALE. State of the cloud Report [Internett], 2015. Santa Barbara, CA: Rightscale; 18.02.2015 [09.05.2018] Tilgjengelig fra https://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf.

[177] RIGHTSCALE. State of the cloud Report [Internett], 2016. Santa Barbara, CA: Rightscale; 09.02.2016 [09.05.2018] Tilgjengelig fra https://assets.rightscale.com/uploads/pdfs/RightScale-2016-State-of-the-Cloud-Report.pdf.

[178] RIGHTSCALE. State of the cloud Report [Internett], 2017. Santa Barbara, CA: Rightscale; 15.02.2017 [09.05.2018] Tilgjengelig fra https://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf.

[179] RIGHTSCALE. State of the cloud Report [Internett], 2018. Santa Barbara, CA: Rightscale; 13.02.2018 [09.05.2018] Tilgjengelig fra https://assets.rightscale.com/uploads/pdfs/RightScale-2018-State-of-the-Cloud-Report.pdf.

[180] SALTSTACK. About SaltStack [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://saltstack.com/about/.

[181] SALTSTACK. Communication and Security [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/system/communication.html.

[182] SALTSTACK. Create a Salt State [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/fundamentals/states.html.

[183] SALTSTACK. Data [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/system/data.html.

[184] SALTSTACK. Functions [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/config/functions.html.

[185] SALTSTACK. Introduction [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/.

[186] SALTSTACK. Introduction to Salt [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/2017.7/topics/.

[187] SALTSTACK. Release Notes [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/latest/topics/releases/.

[188] SALTSTACK. Remote Execution [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/getstarted/system/execution.html.

[189] SALTSTACK. Salt Module Index [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/2017.7/salt-modindex.html.

[190] SALTSTACK. Salt Nodes per master [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://saltstack.com/saltstack-enterprise-system-requirements/.

[191] SALTSTACK. Salt Stack Formulas [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://github.com/saltstack-formulas.

[192] SALTSTACK. The Salt Open Source Software Project [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://saltstack.com/community/.

[193] SALTSTACK. Version Numbers [Internett]. Salt Lake City, UT: SaltStack; [24.04.2018] Tilgjengelig fra https://docs.saltstack.com/en/2017.7/topics/releases/version_numbers.html.

[194] SCHMIDT, D. Guest Editor's Introduction: Model-Driven Engineering. *Computer 39*, 2 (February 2006), 25–31.

[195] SCOTT CHACON, B. S. *Pro Git*, 2 ed. Apress Media, 2014.

[196] SERVERSPEC. Serverspec [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/mizzy/serverspec.

[197] THE APACHE SOFTWARE FOUNDATION. Apache Virtual Host documentation [Internett]. The Apache Software Foundation; [21.04.2018] Tilgjengelig fra https://httpd.apache.org/docs/2.4/vhosts/.

[198] THE FOREMAN. Foreman Ansible plugin [Internett]. The Foreman; [14.05.2018] Tilgjengelig fra https://www.theforeman.org/plugins/foreman_ansible/2.x/index.html.

[199] THE SYMPA COMMUNITY. Sympa Ansible [Internett]. San Francisco, USA: GitHub Inc.; [14.05.2018] Tilgjengelig fra https://github.com/sympa-community/sympa-ansible.

[200] THE SYMPA COMMUNITY. sympa-community/sympa-ansible: Generic install of Sympa with Ansible [Internett]. The Sympa Community; [21.04.2018] Tilgjengelig fra https://github.com/sympa-community/sympa-ansible.

[201] THE SYMPA COMMUNITY. sympa-community/sympa-puppet-manifest [Internett]. The Sympa Community; [21.04.2018] Tilgjengelig fra https://github.com/sympa-community/sympa-puppet-manifest.

[202] WIKIPEDIA CONTRIBUTORS. Ansible [Internett]. Wikipedia, The Free Encyclopedia; [25.04.2018] Tilgjengelig fra https://en.wikipedia.org/w/index.php?title=Ansible&oldid=837117293.

[203] WIKIPEDIA CONTRIBUTORS. Jinja (template engine) [Internett]. Wikipedia, The Free Encyclopedia; 25.03.2018 [24.05.2018] Tilgjengelig fra https://en.wikipedia.org/w/index.php?title=Jinja_(template_engine)&oldid=832414686.

[204] WILL THAMES. Ansible Inventory Grapher [Internett]. San Francisco, USA: GitHub Inc.; [16.05.2018] Tilgjengelig fra https://github.com/willthames/ansible-inventory-grapher.

[205] WILL THAMES. Ansible Lint [Internett]. San Francisco, USA: GitHub Inc.; [14.05.2018] Tilgjengelig fra https://github.com/willthames/ansible-lint.

[206] WILLSHERSYSTEMS. willshersystems.sshd [Internett]. Durham, NC, USA: Red Hat Inc.; [14.05.2018] Tilgjengelig fra https://galaxy.ansible.com/willshersystems/sshd/.

[207] ZIEGER, S. Puppet Forge: saz [Internett]. Portland, OR: Puppet; [21.04.2018] Tilgjengelig fra https://forge.puppet.com/saz/ssh.

A Kravspesifikasjon og vurderingsgrunnlag

A.1 Kravspesifikasjon

For at vi skal kunne gjøre et utvalg av kandidater, må vi basere oss på de konkrete krav som oppdragsgiver har gitt, i tillegg til de krav som beskriver de helt sentrale elementene som kreves av et slikt system. Vi har delt kravene inn i krav for grunnleggende funksjonalitet, krav til utvikling, krav til sikkerhet, krav til dokumentasjon og krav til opplæring.

A.1.1 Grunnleggende krav

De grunnleggende kravene er helt sentrale for valg av system. De inkluderer krav som beskriver grunnleggende funksjonalitet som kreves av et konfigurasjonsstyringssystem.

Automatisk konfigurering av maskiner

Systemet skal kunne utføre repeterbare prosesser for konfigurasjon av maskiner, uten at det kreves innsyn fra systemadministrator.

Idempotent operasjon

CM-systemet skal kunne kjøre den samme konfigurasjonen på maskiner uten at den gjør endringer hvis det ikke behøves. Altså hvis en maskin allerede er i ønsket konfigurasjonstilstand, skal ikke oppgavene utføres på nytt.

Kan skaleres til mange maskiner

At systemet kan skaleres til mange maskiner innebærer at den sentrale komponenten (tjeneren, "masteren") skal kunne koble til og konfigurere mange maskiner samtidig eller kontinuerlig.

Skriptbart grensesnitt

Systemets grensesnitt på kommandolinje skal kunne brukes i skript for å hente ut informasjon og gjøre andre automatiserte oppgaver.

Tilsynsløs operasjon

Når systemet er i normal drift skal det kunne kjøre automatisk uten at personer trenger være involvert i denne prosessen.

A.1.2 Krav til utvikling

Krav til utvikling omhandler kodehåndtering av konfigurasjonssystemet fra brukeren og produsenten sitt perspektiv.

Bruker av CM-systemet

Konfigurasjon i tekstfiler

Konfigurasjon i tekstfiler innebærer at de definerte konfigurasjonene og data skal kunne skrives i et strukturert tekstformat i vanlige tekstfiler med et enkelt tegnsett. Disse tekstfilene skal være input til CM-systemet når det skal konfigurere maskiner.

Versjonskontroll

Konfigurasjonen i form av tekstfiler skal kunne versjonskontrolleres i et ekstern system for å spore endringer mellom versjoner i konfigurasjonsdefinisjonen.

Testing

Automatisert testing av definisjonskoden for når endringer i konfigurasjonen blir gjort. Det skal være mulighet for å kjøre den gjennom ulike tester, eksempelvis testing for korrekt syntaks, enhetstesting, akseptansetesting, og eventuelt andre spesifikke tester.

Produsent / utvikler av CM-systemet

Oppgraderbarhet og bakoverkompatibilitet

CM-systemet i seg selv skal kunne oppgraderes jevnlig uten forandringer som skaper inkompatibilitet mellom minst to versjoner.

Åpen kildekode

Åpen kildekode er en forutsetning for at brukere av systemet skal kunne ha innsikt i funksjonalitet, og å kunne få hjelp og støtte fra andre enn produsenten/utvikleren selv. Med åpen kildekode er det også enklere å følge nye versjoner, og at brukere skal kunne bidra med ønsket funksjonalitet.

Stabil versjon i løpet av det siste året

Det er viktig at systemet fremdeles utvikles og jobbes med jevnlig. Dette kravet kan indikere den nåværende utviklingen, populariteten og fremtidig støtte.

A.1.3 Krav til sikkerhet

Tilgangsstyring

Det skal være mulig å styre hvem av de ansatte som har tilgang til hvilken del av konfigurasjonen, hvem som har lov til å gjør endringer, osv.

Sikker lagring

Hemmeligheter, slikt som passord og private krypteringsnøkler, bør kunne legges utenfor konfigurasjonen i tekstfiler i et sikret eksternt system.

Sikker distribuering av konfigurasjoner

Konfigurasjoner skal være sikret med kryptografiske løsninger når de kommuniseres til styrte maskiner over nettverket.

Autentisering

Tjener og styrt maskin i CM-systemet skal autentiseres overfor hverandre ved bruk av kryptografiske løsninger, eksempelvis sertifikater eller nøkler.

Styre brannmur på maskin

Brannmuren på en maskin skal kunne styres gjennom systemet.

A.1.4 Krav til dokumentasjon

Det er krav til at produsenten/utvikleren av CM-systemet tilbyr omfattende dokumentasjon av funksjonalitet og bruk av systemet. Denne dokumentasjonen skal tilbys gratis.

Offisielt på produsentens nettside

Produsenten/utvikleren skal tilby dokumentasjon på deres nettsider, som skal omfatte beskrivelser av funksjonalitet og bruk av CM-systemet.

Arkivert dokumentasjon

Den nettbaserte dokumentasjonen skal være arkivert, som betyr at dokumentasjon fra tidligere versjoner er tilgjengelig for innsyn til sammenligning for nyere versjoner av systemet.

Runtime

Dokumentasjonen skal være tilgjengelig på kommandolinjen når man arbeider med CM-systemet, eller runtime. Dette innebærer at funksjoner og moduler skal ha en kort beskrivelse med eksempler. Dette inkluderer også man-sider i Linux.

A.1.5 Krav til opplæring

For at brukerne av et CM-system skal lære seg bruken av det, kreves det at flere ressurser for opplæring er lett tilgjengelig.

Bøker

Bøker for systemet skal eksistere, som kan kjøpes eller erverves. Bøkene vil kunne gi brukeren innsikt fra de erfarne brukerne som har skrevet boken.

Kurs

Kurs skal være tilgjengelig i form av nettkurs eller kurs med fysisk oppmøte. Det er ikke krav om at dette skal tilbys innenfor Norges landegrenser.

Offisielle ressurser

Produsenten/utvikleren av CM-systemet skal tilby egne ressurser for grunnleggende opplæring og annen praksis. Disse ressursene skal være åpent tilgjengelig gjennom deres nettsider.

A.2 Vurderingskriterier

Vurderingskriteriene skal hjelpe oss i å gjøre en strukturert og grundig vurdering av de utvalgte kandidatene. Noen av disse vurderingskriteriene vil i første omgang inngå i utvalgsprosessen sammen med kravspesifikasjonen for å velge ut en håndfull kandidater (se Kvalifisering, utvalg og vurdering).

A.2.1 Organisasjon

Vi må gjøre en vurdering av organisasjonen og deres offentlige profil. Dette er en viktig del av inntrykket, både i forhold til systemets antatte utvikling i fremtiden, og kvaliteten på systemet.

Hvem står bak produktet/systemet

Hvem som utvikler produktet/systemet kan ha betydning for kvalitet og framtidig utvikling og støtte.

Tilbyr kommersiell løsning

Hvis organisasjonen bak produktet/systemet tilbyr en kommersiell løsning, kan det bety at profesjonell opplæring og konsulenttjenester er tilgjengelig.

Økosystem

Det kan være en stor fordel at organisasjonen bak produktet/systemet også tilbyr tjenester som tillater å dele kode og kompetanse mellom brukere.

Hjemmesider

Hvorvidt organisasjonens hjemmesider er oversiktlige og tilbyr ulike ressurser, som dokumentasjon, opplæring og lenker til kommunikasjonskanaler.

A.2.2 Arkitektur

Arkitekturen er viktig å skaffe et godt inntrykk av. Det finnes flere ulike modeller, som for noen kan være avgjørende for valget av et slikt system.

Systemets arkitektur kan ha vesentlig betydning for anvendelse av og funksjonalitet. Følgende momenter skal vurderes angående arkitektur.

Synkronisering-/distribuering av konfigurasjon

Hvordan konfigurasjon blir distribuert fra en tjener til styrte maskiner. Det finnes i hovedsak to modeller: “push” og “pull”. Førstnevnte bestemmes av tjeneren, og sistnevnte bestemmes av klienten. Hvilken modell som brukes vil ha betydning for maskinressurser og skalering av tjeneren.

Skalering

Hvor mange klienter en tjener kan håndtere. Dette punktet kan avhenge mye av tjenerens maskinressurser og nettverkstilkobling. Skalering betyr også hvorvidt tjeneren i seg selv kan skaleres, eksempelvis med lastbalansering over flere tjenermaskiner.

A.2.3 Datahåndtering

Vi vurderer i hovedsak tre måter å håndtere informasjon på. Disse måtene finner man brukt i flere systemer, men det er ikke sikkert alle har det, eller at oppdragsgiver har bruk for alle.

Konfigurasjonsdata

Hvordan konfigurasjonsdata organiseres. Hvis en konfigurasjon har mye konfigurasjonsdata, kan det lønne seg at disse er plassert et annet sted og gjort oppslag mot.

Statisk informasjon om maskiner

Hvordan systemet benytter statisk informasjon om maskiner. Statisk informasjon er som regel IP-adresse, innstillinger, versjon av operativsystem, også videre.

Konfigurasjonsregister

Om systemet benytter konfigurasjonsregister. Dette kan benyttes for å oppbevare historiske data om systemet og de styrte maskinene. Statisk informasjon fra alle maskiner kan også oppbevares og deles via denne databasen.

A.2.4 Sikkerhet

CM-systemer styrer i noen tilfeller svært store distribuerte infrastrukturer, og da er det viktig at ulike sikkerhetsmekanismer er på plass, spesielt for behandling av data og hemmeligheter, som passord, nøkler, osv.

Tilgangskontroll for administrering av systemet

Om systemet støtter autentisering og tilgangskontroll basert på hvem som er logget inn.

Konfidensialitet i kommunikasjon

Om systemet støtter kryptert kommunikasjon mellom klient og tjener, og hvordan dette er implementert.

Konfidensialitet for konfigurasjonsdata

Om systemet støtter kryptering av sensitive konfigurasjonsdata, og hvordan dette er implementert og anvendt.

Eksponering på nettverket

Om noen av maskinene, tjener eller klienter, må åpne TCP- eller UDP-porter for kommunikasjon, og hvilken betydning dette eventuelt kan ha for sikkerhet i systemet og maskinen.

A.2.5 Arbeidsflyt

Arbeidsflyten for de fleste CM-systemer er ganske lik, men systemene skiller seg fra hverandre, spesielt syntaks.

Type språk for deklarasjon

Om systemet bruker DSL (Domain Specific Language), strukturert dataspårk eller annet. DSL-er kan kreve mer forståelse for programmering og kan fungere på et høyere nivå for abstraksjon.

Muligheter for testing

Om konfigurasjonsspråket lar seg teste. Både om syntaks og enhetstesting er mulig.

A.2.6 Dokumentasjon

Offisiellt tilgjengelig dokumentasjon

Om organisasjonen tilbyr offisiell dokumentasjon på deres nettsider, og om denne er omfattende nok til å dekke alle behov for læring og anvendelse av systemet.

Arkivert dokumentasjon

Om organisasjonens dokumentasjon finnes tilgjengelig i flere eldre versjoner, og hvor langt tilbake de viser det som er arkivert eller av eldre versjon.

Runtime-dokumentasjon på kommandolinje

Om systemet, når det er installert på maskiner, tilbyr enkelt dokumentasjon for funksjoner og ressurser.

A.2.7 Brukerbase

Brukerbasen sier mye om hvem som bruker systemet og hvor aktivt det brukes. Dette vil inngå i det initielle utvalget av kandidater, men vil også vurderes nærmere på individuelt nivå.

Hvem som bruker systemet

For hvilke formål systemet er tatt i bruk. Dette kan gi en indikator på hva systemet er spesialisert mot.

Andel brukere

Hvor mange som har tatt ibruk systemet. Dette kan beskrives av relative tall basert på statistikk fra søkemotorer eller rapporter.

Offentlig diskusjon og omtale

Hvor mye systemet er diskutert, eksempelvis målt i antall diskusjonsinnlegg på nettforum og spørsmålssider.

A.2.8 Systemets utvikling

Utviklingen av systemet kan si noe om dets tilgjengelighet og støtte i fremtiden. Hvis det utvikles lite i løpet av senere tid, sammenlignet med andre systemer, vil man kunne vurdere om det er sannsynlig at systemet vil bli mindre brukt og utviklet i overskuelig fremtid.

Bakoverkompatibilitet mellom versjoner

Hvordan støtte av funksjonalitet mellom versjoner er, og hvordan slutt på støtte for funksjonalitet eller endringer historisk sett har vært.

Utviklingsfrekvens og aktivitet i kodebase

Hvor ofte systemet utvikles. Tall fra offentlige plattformer for publisering av kode og programvare kan gi en indikator på systemets interesse og aktivitet blant brukere.

A.2.9 Opplæring

Ressurser for opplæring til et system bør være lett tilgjengelig. Det finnes mange ulike ressurser, men vi har valgt å begrense vurderingen til bøker, kurs og offisielle(fritt tilgjengelige) ressurser.

Offisielle, fritt tilgjengelige ressurser

Om organisasjonen bak systemet tilbyr gratis ressurser (eksempelvis begynnerveiledninger) for å komme igang med bruken av det.

Bøker

Om det finnes bøker som beskriver systemets aspekter og anvendelse.

Kurs

Om det tilbys kurs for å lære seg bruken av systemet.

B Initielt utvalg

B.1 Alle kandidater i alfabetisk rekkefølge

1	Ansible
2	Bcfg2
3	Capistrano
4	cdist
5	CFEngine
6	Chef
7	Desired State Configuration for Linux
8	ISconf
9	Juju
10	Local ConFiGuration system (LCFG)
11	NOC
12	OCS Inventory NG with GLPI
13	Open pc server integration (Opsi)
14	PIKT
15	Puppet
16	Quattor
17	Radmind
18	Rex
19	Rudder
20	Salt
21	SmartFrog
22	Spacewalk
23	STAF
24	Synctool

Hovedkilden til denne listen var Wikipedia-siden om Åpen-kildekode CM-systemer https://en.wikipedia.org/w/index.php?title=Comparison_of_open-source_configuration_management_software&oldid=823008810. *DSC for Linux* ble lagt til som en kadidat av gruppen selv.

I tillegg foreslo oppdragsgiver *xCat - (Extreme Cloud Administration Toolkit)*, https://xcat.org/, som en kandidat etter at den initielle utvelgelsen var fullført. Denne ble av gruppa vurdert til å ikke tilfredstille kravene i kravspesifikasjon og ble derfor ikke nærmere vurdert.

B.2 Runde 0: Kandidater sortert etter siste release

Tall hentet 2. februar 2018. Kandidater som ikke har hatt release siste år er markert med rødt.

	Kandidat	Versjon	Dato	Kilde
1	CFEngine	3.10.3 LTS	06.02.2018	https://cfengine.com/product/supported-versions/
2	Salt	2017.7.3	05.02.2018	https://github.com/saltstack/salt/releases/tag/v2017.7.3
3	Puppet	5.3.4	05.02.2018	https://puppet.com/docs/puppet/5.3/release_notes.html#puppet-534
4	Ansible	2.4.3.0-1	01.02.2018	https://github.com/ansible/ansible/releases/tag/v2.4.3.0-1
5	Local ConFiGuration system (LCFG)	2018012901	29.01.2018	http://www.lcfg.org/download/
6	Chef	12.17.15 (tjener)	25.01.2018	https://github.com/chef/chef-web-docs/blob/master/chef_master/source/release_notes_server.rst
7	Juju	2.3	08.12.2017	https://launchpad.net/juju/+milestone/2.3.1
8	Capistrano	3.10.1	08.12.2017	https://github.com/capistrano/capistrano/releases/tag/v3.10.1
9	Rudder	4.2.3	08.12.2017	https://github.com/Normation/rudder/releases/tag/4.2.3
10	OCS Inventory NG with GLPI	2.4	06.12.2017	https://github.com/OCSInventory-NG/OCSInventory-ocsreports/releases
11	Rex	1.6.0	04.12.2017	https://github.com/RexOps/Rex/releases/tag/1.6.0
12	cdist	4.7.3	10.11.2017	https://github.com/ungleich/cdist/releases/tag/4.7.3
13	Quattor	17.8.0	31.10.2017	http://www.quattor.org/documentation/
14	Spacewalk	2.7	26.09.2017	https://github.com/spacewalkproject/spacewalk/wiki/Spacewalk-releases
15	SmartFrog	3.18.016	16.05.2017	https://sourceforge.net/projects/smartfrog/
16	STAF	3.4.26	31.12.2016	http://staf.sourceforge.net/
17	Desired State Configuration for Linux	1.1.1-294	31.08.2016	https://github.com/Microsoft/PowerShell-DSC-for-Linux/releases
18	Open PC server integration (Opsi)	4.0.7	29.08.2016	https://forum.opsi.org/viewtopic.php?f=10&t=8601
19	Bcfg2	1.3.6	11.06.2015	http://bcfg2.org/download/
20	NOC	15.05.1	21.05.2015	https://github.com/nocproject/noc
21	Synctool	6.2	28.03.2015	https://github.com/walterdejong/synctool/releases/tag/v6.2
22	ISconf	4.2.8.250	14.10.2014	https://github.com/stevegt/isconf4
23	Radmind	1.14.1	13.12.2010	https://sourceforge.net/projects/radmind/
24	PIKT	1.19.0	10.09.2007	http://pikt.org/pikt/news.html

B.3 Runde 1: Popularitet på nett

B.3.1 StackOwerflow-tags

Kandidat	StackOwerflow-tags	Poeng
Ansible	8473	15
Chef	6515	14
Capistrano	4338	13
Puppet	3448	12
Salt	920	11
Juju	48	10
CFEngine	35	9
Rex	18	8
Spacewalk	0	0
OCS Inventory NG with GLPI	0	0
Quattor	0	0
SmartFrog	0	0
cdist	0	0
Rudder	0	0
Local ConFiGuration system (LCFG)	0	0

Kilde: https://stackoverflow.com/tags, hentet 07.02.2018, kl. 13.30. Noen av tallene er lagt sammen av flere tags, f.eks *Chef* og *Chep recipe* og *Capistrano* og *Capistrano3*.

For Salt er det bare brukt taggen *Saltstack* fordi taggen *salt* er brukt om salting av passord.

B.3.2 GitHub-stars

Kandidat	GitHub-stars	Poeng	Kilde
Ansible	28267	15	https://github.com/ansible/ansible/stargazers
Capistrano	10195	14	https://github.com/capistrano/capistrano/stargazers
Salt	8540	13	https://github.com/saltstack/salt/stargazers
Chef	5202	12	https://github.com/chef/chef/stargazers
Puppet	4859	11	https://github.com/puppetlabs/puppet/stargazers
Juju	1088	10	https://github.com/juju/juju/stargazers
Rex	565	9	https://github.com/RexOps/Rex/stargazers
Spacewalk	351	8	https://github.com/spacewalkproject/spacewalk/stargazers
CFEngine	279	7	https://github.com/cfengine/core/stargazers
cdist	182	6	https://github.com/ungleich/cdist/stargazers
Rudder	159	5	https://github.com/Normation/rudder/stargazers
OCS Inventory NG with GLPI	91	4	https://github.com/OCSInventory-NG/OCSInventory-Server/stargazers
Quattor	7	3	https://github.com/quattor/configuration-modules-core/stargazers
Local ConFiGuration system (LCFG)	3	2	https://github.com/lfex/lcfg/stargazers
SmartFrog*	-	0	ikke på GitHub

*Smartfrog er ikke på GitHub, men ligger på Sourceforge med 11 reviews.

B.3.3 Google Trends

Google Trends-utvikling siste 5 år

Kandidat	Utvikling siste 5 år	Poeng
Ansible	3504,60%	15
Puppet	2275,10%	14
Chef	795,79%	13
OCS Inventory NG with GLPI	532,18%	12
Salt	482,76%	11
Spacewalk	250,57%	10
Capistrano	166,67%	9
CFEngine	111,11%	8
Juju	81,23%	7
Quattor	34,48%	6
SmartFrog	26,82%	5
Rex	-	0
cdist	-	0
Rudder	-	0
Local ConFiGuration system (LCFG)	-	0

Google Trends utvikling siste 2 år

Kandidat	Utvikling siste 2 år	Poeng
SmartFrog	258,27%	15
Quattor	92,13%	14
Ansible	76,14%	13
Spacewalk	22,17%	12
Salt	-12,37%	11
Puppet	-19,58%	10
OCS Inventory NG with GLPI	-24,36%	9
Capistrano	-31,90%	8
Chef	-39,49%	7
Juju	-52,91%	6
CFEngine	-86,69%	5
Rex	-	0
cdist	-	0
Rudder	-	0
Local ConFiGuration system (LCFG)	-	0

Alle Google Trends tall er relativ til det høyeste søketallet, på en gitt dato, dvs. Ansible i dette tilfellet. Kategori valgt er *software*. Tallene i tabellen er beregnet ut fra søketall fra *Google Trends* med bruk av Trend()-funksjonen i *Google Regneark*.

Kilder til Google Trends-tall

- https://trends.google.com/trends/explore?cat=32&date=2013-02-07%202018-02-07&q=%2Fm%2F0k0vzjb,%2Fm%2F03d3cjz,%2Fm%2F05zxlz3,%2Fm%2F0hn8c6s,%2Fm%2F06bmn8
- https://trends.google.com/trends/explore?date=2013-02-07%202018-02-07&q=%2Fm%2F0k0vzjb,%2Fm%2F0h68cj_,%2Fm%2F0_lmq0n,%2Fm%2F047qmzp,%2Fm%2F046shd
- https://trends.google.com/trends/explore?date=2013-02-07%202018-02-07&q=%2Fm%2F0k0vzjb,%2Fm%2F0j_3yxh,%2Fm%2F0113s97m,%2Fm%2F02rgr9f,lcfg
- https://trends.google.com/trends/explore?date=2013-02-07%202018-02-07&q=quattor,smartfrog,%2Fm%2F0k0vzjb

B.3.4 Sammenlagt poengsum fra runde 1

Kandidat	StackOwerfl.	GitHub	Google, 5år	Google, 2 år	Sum
Ansible	15	15	15	13	58
Puppet	12	11	14	10	47
Salt	11	13	11	11	46
Chef	14	12	13	7	46
Capistrano	13	14	9	8	44
Juju	10	10	7	6	33
Spacewalk	0	8	10	12	30
CFEngine	9	7	8	5	29
OCS Inventory NG with GLPI	0	4	12	9	25
Quattor	0	3	6	14	23
SmartFrog	0	0	5	15	20
Rex	8	9	0	0	17
cdist	0	6	0	0	6
Rudder	0	5	0	0	5
Local ConFiGuration system (LCFG)	0	2	0	0	2

B.4 Runde 2: Kravspesifikasjon

Kandidat	Automatisk konfig av maskiner	Idempotent Operasjon	Skalerer til flere maskiner	Eksternalisert konfigurasjon	Skriptbart interface	Tilsynsløs operasjon
Ansible	ja	ja	ja	ja	ja	ja
Puppet	ja	ja	ja	ja	ja	ja
Salt	ja	ja	ja	ja	ja	ja
Chef	ja	ja	ja	ja	ja	ja
Capistrano	nei	uvisst	ja	ja	ja	ja
Juju	nei	uvisst	ja	ja	ja	ja
Spacewalk	n/a	n/a	n/a	n/a	n/a	n/a
CFEngine	ja	ja	ja	ja	ja	ja
OCS Inventory	n/a	n/a	n/a	n/a	n/a	n/a
Quattor	ja	uvisst	ja	ja	ja	ja
SmartFrog	antakelig	antakelig	antakelig	antakelig	antakelig	antakelig
Rex	ja	uvisst	ja	ja	ja	ja
cDist	ja	ja	ja	ja	ja	ja
Rudder	n/a	n/a	n/a	n/a	n/a	n/a
LCFG	ja	antakelig	ja	ja	ja	ja

Kilder kravspesifikasjon

- http://docs.ansible.com/ansible/latest/playbooks_intro.html
- https://docs.puppet.com/references/glossary.html#idempotent
- https://docs.puppet.com/puppet/4.5/architecture.html
- https://docs.saltstack.com/en/latest/glossary.html
- https://docs.saltstack.com/en/latest/topics/states/
- https://docs.saltstack.com/en/latest/topics/transports/raet/index.html
- https://docs.chef.io/server_components.html
- https://docs.chef.io/chef_overview.html
- https://docs.chef.io/resource.html
- https://docs.chef.io/chef_overview.html
- http://capistranorb.com/documentation/overview/what-is-capistrano/
- https://github.com/capistrano/capistrano/blob/master/README.md
- https://jujucharms.com/docs/stable/about-juju
- https://access.redhat.com/products/red-hat-satellite
- https://docs.cfengine.com/docs/3.10/guide-introduction.html
- https://docs.cfengine.com/docs/master/guide-special-topics-change-management.html
- https://www.ocsinventory-ng.org/en/
- http://www.quattor.org/documentation/index.html
- http://www.quattor.org/documentation/getting-started.html
- https://sourceforge.net/p/smartfrog/wiki/Home/
- https://www.rexify.org/docs/guides/start_using__r__ex.html
- https://nico.schottelius.org/software/cdist/man/latest/cdist-manifest.html
- https://nico.schottelius.org/software/cdist/man/latest/cdist-why.html
- https://www.rudder-project.org/rudder-doc-4.3/rudder-doc.pdf
- http://www.lcfg.org/doc/guide.pdf

B.4.1 Kommentarer til kandidater som ikke gikk videre

I første iterasjon av utvalget for krav vil ta med kandidater som oppfyller alle krav. Under nevnes årsakene til at noen kandidater ble tatt ut.

Capistrano

Capistrano lar brukeren konfigurere webtjenester men mangler muligheter for å konfigurere på OS/maskin tjenester. Men det kan brukes i samarbeid med andre CM-systemer for å få denne type funksjonalitet.
http://capistranorb.com/documentation/overview/what-is-capistrano/

Juju

JuJu Fokuserer hovedsakelig på cloud tjenester som azure aws og gcp, mens oppdragsgiver ønsker å konfigurere maskiner lokalt. Juju skal kunne kjøre lokalt ved hjelp av "LXD cloud", Juju er et mer høytliggende CM-system og har den manglende funksjonalitet når det kommer til konfigurasjon av operativsystemet. Dette gjør at det ofte brukes med Chef, Puppet og Ansible, enten som et bindepunkt mellom dem eller for høytliggende software konfigurasjon.
https://jujucharms.com/docs/2.3/about-juju

Spacewalk

Spacewalk er et system for å tilby Red Hats pakkebrønner lokalt i eget nettverk. Det er upstream til deres produkt, Red Hat Satellite. Det er dermed for sentrert rundt pakkehåndtering for RHEL-baserte Linux-distroer.
https://access.redhat.com/products/red-hat-satellite
https://spacewalkproject.github.io/

OCS Inventory NG

Dette systemet er i hovedsak et system for styring av inventar og utrulling av programvare, og er et stort system med mye forskjellig funksjonalitet, men er ikke spesifikt ment for konfigurasjonsstyring, slik vi definerer det med våre krav til hovedfunksjonalitet.
https://www.ocsinventory-ng.org/en/

SmartFrog

SmartFrog hadde ingen dokumentasjon som kunne forklare funksjonalitet. Det lille som var å finne var på en HP-nettside som enda virker å bare bli vedlikeholdt uten noe fremtidig mål. Det var et system utviklet av HP Labs i Bristol i England, men har nå blitt lagt ned og overtatt av en annen organisasjon. SmartFrogs nettsider sier ingen ting om systemets fremtid.
http://www.hpl.hp.com/research/smartfrog/releasedocs/index.html

Rudder

Rudder dreier seg om å sjekke systemers tilstand og som et brukervennlig (web)grensesnitt for ikke-eksperter å kunne rulle ut konfigurasjoner. Utrullingen skjer da antakeligvis gjennom et annet system som kommuniserer med Rudder. Videre vil Rudder rapportere helsetilstand og samsvar mellom implementasjon og policyer ved bruk av templates. Det er dermed ikke et CM-system i seg selv, men tyder på å være et nyttig tillegg til et CM-system.
https://www.rudder-project.org/site/about/what-is-rudder/

LCFG

Dette systemet tyder også på å ha blitt brukt til konfigurasjonsstyring av Unix-(lignende)-systemer. Rent teknisk sett virker det som det oppfyller det meste av krav, men brukermanualen er datert til 6. januar 2005, som er fryktelig lenge siden. Nettsidene deres for øvrig virker også ganske døde på aktivitet. Det virket litt rart at vi derfor sitter med denne med tanke på at siste stabile versjon var innenfor vårt krav om det. Vi baserte oss på versjonsnummereringen, som antakeligvis var en kombinasjon av årstall, måned og dato.
http://www.lcfg.org/doc/guide.pdf

B.4.2 CFEngine, Quattor, cDist, Rex

CFEngine er den en av de kandidatene som viste seg som en verdig kandidat for å ta med i videre vurdering. Men basert på vår poenggiving i runde 1 når den altså ikke opp blant de mest populære og mest brukte systemene. Quattor, cdist og Rex virker også som greie systemer, men når ikke engang opp til CFEngine i popularitet

C Individuell vurdering og utvalg

C.1 Fordeler og ulemper med Ansible, Chef, Puppet og Salt

Ansible

- Pluss:
  - Enkelt å komme i gang med og enkelt å lære seg
  - Fleksibelt og gir stor mulighet for tilpasning
  - Mye brukt i mindre til medium skala servermiljøer
- Minus:
  - Varierende kvalitet på rollene i Galaxy
  - Skalering/ytelse kan være en utfordring
  - Kjører ikke automatisk "out of the box"

Chef

- Pluss:
  - Oversiktlig syntaks, lite bruk av spesialtegn
  - Kommer komplett med utviklingsverktøy (testing inkludert)
  - Mye brukt i store servermiljøer
- Minus:
  - Sentrert rundt utviklingsverktøyet ChefDK
  - Ikke helt intuitivt å jobbe med ulike brancher/versjoner av infrastrukturen
  - Versjon mellom tjener og klient følger ulik versjonering

Puppet

- Pluss:
  - God kvalitet og godt utvalg av moduler i Puppet Forge
  - "Modent" produkt som har stort brukersamfunn
  - Mye brukt i store servermiljø
- Minus:
  - Relativt bratt læringskurve og høy terskel for etablering
  - Kan kreve avansert forståelse av Ruby for å utvide med egendefinert funksjonalitet

Salt

- Pluss:
  - Skalerer godt til store servermiljø
  - Tilpassingsvennlig arkitektur
  - Mye innebygget funksjonalitet
- Minus:
  - Ingen ekvivalent til Forge/Galaxy/Supermarket for deling av modeller
  - Ikke like mye brukt som de andre tre kandidatene
  - Må bruke Jinja2[1] for å gjøre logikk i YAML-filer

C.2 Resultatmatrise fra individuell vurdering

Se tabell 1.

[1] https://docs.saltstack.com/en/getstarted/config/jinja.html

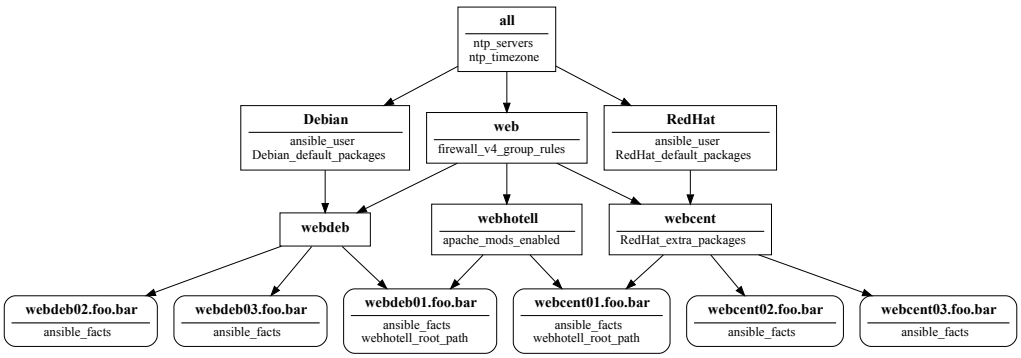
	Ansible	Puppet	Chef	Salt
Enterprise-løsning	Ansible Engine	Puppet Enterprise	Chef Automate	SaltStack Enterprise
Økosystem	Galaxy	Forge	Supermarket	GitHub
Online dokumentasjon	Ja	Ja	Ja	Ja
Run-time dokumentasjon	Ja	Ja	Ja	Ja
Offisiell opplæring	Ja	Ja	Ja	Ja
Bøker	Ja	Ja	Ja	Ja
Kurs	Ja	Ja	Ja	Ja
Synkronisert konfig.	Manuelt, automatisk m/AWX el. tredjepartsverktøy	Regelmessig fra node	Utgangspunkt: regelmessig fra node	Iverksatt av master
Push / pull	Push	Pull	Pull	Push
Deklarativt språk	YAML	Puppet DSL	Chef DSL	YAML, (Python)
VCS	Ja	Ja	Ja	Ja
Test-muligheter	Kitchen Ansible (TestKitchen)	Ruby spec	Testkitchen	Ja
Tilgangskontroll	Nei, men via AWX/Tower	Nei, Puppet Enterprise	LDAP/AD	LDAP/AD
Sikker kommunikasjon	Ja: SSH	Ja: HTTPS/TLS	Ja: HTTPS/TLS	Ja: kryptert
Sikker konfig.data	Ansible-vault	EYAML, med 1 felles nøkkel	Databags	GPG, med 1 felles nøkkel
Autentisering	Nøkler	Sertifikater	Nøkler	Nøkler
Eksponering	klient: 22	tjener: 8140	tjener: 22, 80, 443	tjener: 4505, 4506
Kan styre brannmur?	Ja	Ja	Ja	Ja

Tabell 1: Resultater fra individuell vurdering

D Ansible

D.1 Gruppevariabler

D.2 Tidsforbruk for kjøring av site.yml ved endring av ansible.cfg



Figur 6: Eksempel på gruppevariabler for flere webservere

	Uten SSH-pipelining	Med SSH Pipelining	Med SSH-pipelining (Strategy = free)
	124.567	106.852	77.697
	127.249	104.330	77.149
	123.312	102.703	80.686
	124.786	101.076	75.900
	124.381	101.166	76.097
	122.817	102.551	76.148
	122.127	104.926	77.104
	125.204	102.648	76.877
	122.661	101.972	78.019
	123.543	101.524	76.174
Snitt (i sekunder)	124.065	102.975	77.185
Reduksjon (prosent)		17.0	37.8

Tabell 2: Tidsforbruk for kjøring av site.yml ved forskjellige innstillinger i Ansibles konfigurasjonsfil, ansible.cfg

E Puppet

E.1 Lange kodeeksempler

Listing E.1: Kode fra `./production/site/web/ssh.pp`

```
# SSH-serverkonfigurasjon for web-maskiner
class web::ssh {

  # Oppslag i Hiera
  $web_ssh_options = lookup('$web::ssh::server_options',
                            merge => hash)
  $common_ssh_options = lookup('$ssh::server_options',
                               merge => hash)

  # Merge resultat
  $ssh_options = $web_ssh_options + $common_ssh_options

  # Deklarering av ssh-modul
  class { "ssh::server":
    options            => $ssh_options,
    validate_sshd_file => true,
  }
}
```

Listing E.2: Utdrag fra `./production/site/web/data/webhotell.yaml`

```
web::webhotell::vhosts:
  site1.webcent01.foo.bar:
    servername: "site1.webdeb01.foo.bar"
    port: "80"
    docroot: "/var/www/site1"
    directories:
      - path: "/var/www/site1"
      - allow_override:
          - "AuthConfig"
          - "ALL"
  site1.webcent01.foo.bar_ssl:
    servername: "site1.webdeb01.foo.bar"
    port: "443"
    docroot: "/var/www/site1"
    ssl: true
    #ssl_cert: "PATH/TO/CERT"
    #ssl_key: "PATH/TO/KEY"
```

F Arbeidslogg

F.1 Ukevis oppsummert logg

Uke 2

Den første uka med prosjektarbeidet har gått med på å starte planlegging av prosjektarbeidet, valgt gruppeleder, signert kontrakt med oppdragsgiver, representert ved Erlend Midttun, hadd første møte med veileder, Eigil Obrestad, og avtalt møte med oppdragsgiver til kommende uke, og valgt én tidligere bacheloroppgave hver å lese.

Uke 3

Denne uken startet skriving av prosjektplan, bestemmelse av øvrige roller, lest alle formelle dokumenter relatert til prosjektet, samle grunnleggende informasjon til senere prosjektarbeid. Denne uken har vi hatt møte med både Erlend og Eigil. I møtet med Erlend stilte vi en del spørsmål og diskuterte prosjektet. Han skulle sende over en mer konkret liste over arbeidskrav. I møtet med Eigil denne uka, ble vi enige om at tre statusrapporter skal skrives til han og Erlend innen bestemte datoer. Eigil sørget også for å opprette prosjekt i SkyHiGh med nødvendige maskinressurser.

Uke 4

Fortsatt arbeid med prosjektplan, og fikk sendt komplett utkast til Eigil for diskusjon på møtet samme uke. Etter dette møtet ble forbedringer, på mangler påpekt av Eigil, igangsatt. Prosjektplanen har nå fått Gantt-diagram for tidsplan, problemstilling, og beskrivelse av arbeidsmetodikk. Nytt utkast til prosjektplanen ble igjen sendt til Erlend og Eigil. Den ble til slutt lagt inn i ShareLaTeX for mer fancy stil. Vi mottar signert kontrakt fra Erlend.

Uke 5

Den endelige prosjektplanen blir levert i BlackBoard torsdag denne uken. Vi har på dette tidspunktet arbeidet i overkant mye med prosjektplanen, siden noen delkapitler ble skrevet om opptil flere ganger. I møte med Eigil denne uken, diskuterte vi planen og våre tanker for det videre prosjektarbeidet. Likevel starter arbeid med å formulere krav for CM-system og vurderingskriterier til utvalgsprosessen. Mot slutten av uken hadde vi hatt en produktiv tankeprosess (ved bruk av Coggle.it) for å finne krav og vurderingskriterier. Dette arbeidet markerte passering av første milepæl.

Uke 6

Arbeid starter med å formulere krav og vurderingskriterier til utvalgsprosessen. Vi planlegger nærmere prosessen for utvalget i flere faser basert på kravene, blant annet utgivelsesdato, tagger på StackOverflow, stjerner i GitHub, og søkestatistikk fra Google Trends. Et initielt utvalg blir tatt, basert først på listen over CM-systemer på WikiPedia, og deretter silet ut for siste dato for utgivelse innen siste år (runde 0). Tall fra StackOverflow, Github og Google Trends for de gjenværende kandidatene blir ført inn i regneark og rangert (runde 1). Så blir de gjenværende kandidatene silt ut basert på om de oppfyller våre krav (runde 3). Til slutt ble fire kandidater valgt ut for videre vurdering. Vi foreslår overfor Erlend at vi tar turen til Trondheim, og han er positiv til dette. Reisen blir bestemt til 15. og 16. mars, uke 11.

Uke 7

Denne uken blir gitt litt mer prioritet til arbeid med et annet emne, siden første eksamen blir holdt mandag 19.02 påfølgende uke. Vi fordeler ansvar for vurdering for de fire gjenværende kandidatene: Stein Ove får Ansible, Andreas får Chef, Audun får Salt, og alle jobber med Puppet til slutt. Første statusrapport blir levert onsdag 14.02. I møte med Eigil diskuterer vi arbeidet fram til dette tidspunktet, og foreslår at vi reiser til Trondheim for å presentere våre funn fra vurderingen. Eigil er positiv til dette.

Uke 8 og 9

Alle jobber etter egen tidsplan disse ukene, siden arbeidet er individuelt. Arbeidet består av å fordype seg og vurdere hvert system basert på vurderingskriteriene. Alle setter opp sitt eget lille eksempel med maskiner i SkyHiGh. Billetter og hotell for Trondheimsturen blir bestilt.

Uke 10

Denne uken starter felles oppmøte igjen. Vi fordeler arbeidsoppgaver for skriving om Puppet, som skal være ferdig vurdert fredag 9.03. Et lite oppsett med maskiner blir også etablert for Puppet. Vi starte å tenke på møtet

vi skal ha i Trondheim følgende uke.

Uke 11

Et sammendrag og presentasjonsslides av våre funn fra vurderingen av de fire kandidatene blir skrevet for hver av de. Sammendraget blir sendt til Erlend slik at de kan lese dette før vi kommer. Torsdag denne uken går med på reise. Vi forsøker så langt det lar seg gjøre å lese mens vi reiser. Fredagen består av møtet med Erlend, Øystein Viggen og andre interesserte som tittet innom. Etter presentasjonen og en del diskusjon rundt kandidatene, blir det bestemt at vi går videre med Ansible og Puppet til eksempeloppsett. Vi er fornøyd med møtet, og reiser hjem igjen.

Uke 12

Vi får tilsendt en liste med tjenester fra Erlend, som de ønsker vi skal vise oppsett av ved hjelp av Ansible og Puppet (NTP, SSH, styring av Apache-moduler, styring av virtualhosts i Apache, og oppsett av Sympa (tjeneste for mailinglister). Vi legger en plan for oppsettet, og diskuterer noen av problemstillingene med tjenestene (blant annet nodegrupperinger i Puppet).

Uke 14

Det videre med eksempeloppsett for Ansible og Puppet. Styring av Apache-moduler og virtualhosts er ferdig. SSH-konfigurasjon har vi et fungerende eksempel av (for ulike innstillinger avhengig av maskin som mottar), NTP var en enkel oppgave. Sympa har vist seg å være vanskeligere, siden ingen skikkelige roller eller moduler eksisterer på Ansible Galaxy og Puppet Forge.

Uke 15

Arbeid med eksempeloppsett for både Ansible og Puppet er i hovedsak ferdig, bortsett fra et bra eksempel for oppsett av Sympa med begge. Kommentering og dokumentasjon gjenstår og vil bli gjort på et senere tidspunkt. Arbeid med Puppet blir fortsatt ut uken, og vi får litt hjelp av Eigil om merging av hasher i Hiera-data.

Uke 16

Tirsdag består i hovedsak av planlegging av rapportskriving. Vi starter med å samle litteraturen som er nødvendig for teori, samt diskusjon flere steder i rapporten. Vi tenker også på kapitteloppsett, bruk av figurer og kodeeksempler, også videre. Resten av uken går til skriving.

Uke 17

Det skrives videre på rapporten, om det initielle utvalget og det samles mer relevant teori som tas ibruk i teori delen. Det blir noe ekstra jobbing på kveldstid for å forøke å få sympa oppsettet i mål.

Uke 18

Vi bestemmer oss for å dele rapportens beskrivelser av arbeidet med eksempeloppsettet i to kapitler: ett for nærmere beskrivelser av aspekter (kalt Aspekter ved Anisble og Puppet), og et annet som beskriver de tekniske implementasjonene. Vi forbereder et utkast til Eigil. Han påpeker ulike mangler og tvilsomme formuleringer. Vi skriver, skriver og skriver.

Uke 19

Et nesten komplett utkast blir sendt til Eigil igjen. Denne gangen påpeker han mindre mangler og at noen deler består av mulig overflødig tekst. Resten, sier han, ser bra ut (forutsett at korrektur og omformuleringer blir gjort). Vi samler og oppsummerer oppmøteloggen, samt beregner omtrent tidsbruk. Lørdag denne helgen blir brukt til å lese gjennom hele rapporten og markere feil og mangler. Søndagen blir brukt først til å lese gjennom en annen gruppes rapport, mot at de leste vår. På søndag ble også en del annet arbeid rettet opp i. Rapporten er i hovedsak ferdig.

Uke 20

Første ting som blir gjort på mandagen er å sende det aller siste utkastet til Eigil. Møtet senere på dagen tyder på at rapporten i stor grad er ferdig. Vi hører med biblioteket angående bruk av fotnoter til lenker underveis. Dette er visstnok ikke “tillat”, og vi måtte gjøre ALLE om til referanser. Denne lista er nå gigantisk.

Alle møtereferater og gruppeloggen blir ført inn i dokumentet som vedlegg. Tirsdagen blir brukt for siste gjennomlesing og påpeking av feil. Utover ettermiddagen og kvelden får vi rettet alle ting. Til slutt skriver vi README-filer i repoene med beskrivelser av viktige ting. Til slutt leveres rapporten og kodeeksemplene i Inspera og Blackboard. Onsdag er fri for arbeid.

Prosjektet har vært en optimal utfordring. “Pick up your suffering and bear it” -JBP

Uke	Fast tid	Eget arbeid
Uke 2	16	4
Uke 3	20	4
Uke 4	20	4
Uke 5	32	8
Uke 6	24	2
Uke 7	20	12
Uke 8	20	12
Uke 9	32	12
Uke 10	32	4
Uke 11	20	8
Uke 12	32	8
Uke 13	0	2
Uke 14	28	12
Uke 15	32	12
Uke 16	32	12
Uke 17	24	8
Uke 18	32	8
Uke 19	36	8
Uke 20	20	4
SUM	472	144
TOTALSUM		616

Tabell 3: Omtrent gjennomsnittlig arbeid per person

F.2 Oppsummering av arbeidstid

Arbeidet har gått i stor grad i forhold til tidsplanen (gantt-diagram) i Prosjektplan og dens milepæler. Mot slutten ble det brukt én uke lengre på arbeid med eksempeloppsett, som gjorde at vi kom senere i gang med skriving av rapport. Arbeidet med eksempeloppsettet overlappet uansett litt med rapportskriving for å få mer korrekte kodeeksempler.

I tråd med prosjektplanen møttes vi alle planlagte dager, fra 10.00 til 14.00 (altså minst 20 timer per uke). Vanligvis har oppmøte blant alle gruppemedlemmene overgått det avtalte faste møtetidsrommet. Enkelte perioder har det vært mer individuelt arbeid i helger og på kveldstid. I perioden for skriving av rapport ble oppmøtetidspunkt fra 08.00, ofte helt til 17.00. I tillegg møtte alle de fleste fredager, selv om denne dagen kunne brukes til individuelt arbeid.

Gjennom prosjektperioden har alle gruppemedlemmene hatt noen planlagte fravær, blant annet til jobbintervjuer.

Eksakt antall felles og individuelle arbeidstimer er vanskelig å beregne, siden vi ofte har jobbet individuelt, også på grunn av at eksakt antall timer arbeid ikke ble notert. Tabell 3 viser omtrent gjennomsnittlig arbeidstid i prosjektperiodens uker.

G Statusrapporter

Disse rapportene ble sendt til vår veileder, Eigil Obrestad, og representant for oppdragsgiver, Erlend Midttun.

G.1 Statusrapport 14.02.2018

G.1.1 Hva vi har gjort så langt

Prosjektplan

Vi startet å arbeide med prosjektplanen og en del forarbeid allerede den 9. Januar. Etter å ha vært på lynkurs i prosjektplanlegging med Tom Røise, startet vi for fullt å planlegge de ulike tingene prosjektplanen vil bestå av. Vi baserte oss i stor grad på et oppsett av planen ut fra punkter Tom nevnte på kurset, og som han har brukt i sin undervisning av systemutvikling.

Vi planla et møte med Eigil den 11. Januar for å diskutere det vi skulle gjøre framover av planlegging, og prosjektet forøvrig. Uka etter, onsdag den 17. Januar, hadde vi et møte med Erlend for å diskutere deres forventninger og hva vi burde ha med av vurdering når vi skulle velge ut og teste ulike konfigurasjonsstyringssystemer. Underveis har Erlend også mottatt diverse dokumenter, både fra planlegging og vårt initielle arbeid med selve prosjektet.

Vi jobbet med prosjektplanen helt inntil fristen, den 1. Februar. I ettertid ser vi at vi brukte ganske lang tid på prosjektplanen. Vi kunne nok ha begrenset oss i fokuset på enkelte detaljer, og "planlagt planen". Til slutt hadde vi likevel en plan som beskriver bakgrunnen, oppgaven, metodikk og tidsplan, slik vi ønsket. Når planen var levert, hadde vi nådd første milepæl.

Utvalg av kandidater

Dagen etter vi hadde levert prosjektplanen, hadde vi en svært produktiv dag med å kartlegge og formulere krav samt vurderingskriterier for CM-systemene. I tillegg måtte vi legge grunnlaget for utvalgsprosessen for å kunne finne de beste kandidatene vi kunne ha med i vurderingen. Dette gjorde vi ved å beskrive ulike steg i prosessen.

Den påfølgende uka startet arbeidet med utvalg av kandidater. Først velge ut basert på release-dato (innenfor siste år). I neste omgang samlet vi bruks- og popularitetsdata for hvert system fra GitHub, StackOverflow og Google Trends. Disse dataene samlet vi i et regneark, og ga de poeng i form av rangering i omganger for hver type data. Til slutt brukte vi TRENDS-funksjonen Google Sheets på dataen fra Google Trends, for å vurdere positiv eller negativ popularitetstrend i løpet en gitt periode.

I neste runde, fjernet vi alle systemer som ikke oppfylte krav fra kravspesifikasjonen. På den måten satt vi igjen med de kandidatene som antakeligvis kunne vært kvalifisert, teknisk sett.

Vi valgte ut de 4 øverste kandidatene med begrunnelse i vår arbeidskapasitet og tidsplanlegging. Med 4 kandidater kunne vi velge én hver som vi kunne fordype oss i, i tillegg til Puppet som vi alle kan noe om fra før av og kan samarbeide om å skrive om denne til slutt.

Når vi hadde gjort ferdig utvalgsprosessen, kunne vi sette igang med individuelt arbeid. Vi skal ha et møte med Erlend den 15. Februar. Vi har også spurt han om de var interessert i et besøk av oss hos Unix-gruppen i Trondheim i uke 10 eller 11, som vi skal diskutere ytterligere på møtet.

G.1.2 Hva vi skal gjøre framover

Mot neste statusrapport, skal vi jobbe videre med hvert vårt system, i tillegg til Puppet til slutt. Etter arbeidet med dette, håper vi å få tatt turen til Trondheim for å møte gjengen som er interessert i resultatene vi har kommer fram til så langt. Vi håper å få tatt en avgjørelse for en beste kandidat allerede da. Når vi har en beste kandidat, kan vi sette igang arbeidet med oppsett for Proof of Concept.

G.2 Statusrapport 21.03.2018

G.2.1 Framgang siden forrige statusrapport (14.02)

Individuell testing av kandidater

Da vi fikk valgt ut Ansible, Puppet, Chef og Salt som de beste fire kandidatene, basert på brukstall og søkestatistikk på Google, satte vi igang med individuelt arbeid for å undersøke hvert system nærmere. Da noterte vi spesielle karakteristikker, arkitektur, arbeidsflyt, sikkerhet, ressurser/økosystem, osv. Siden alle kandidatene i

hovedsak gjorde det samme, måtte vi finne de karakteristikkene som skilte de fra hverandre, og som kan være avgjørende for Unix-gruppas valg av CM-system etter dette prosjektet.

Vi tok ett system hver; Stein Ove tok Ansible, Andreas tok Chef, Audun tok Salt og alle samarbeidet om Puppet til slutt. Fredag den siste uka dro Andreas og Stein Ove på jobbintervju i Trondheim, og Audun ferdigstilte arbeidet med Puppet.

Møte med Erlend og noen andre i Trondheim

Før møtet samlet vi alle funn for hvert system, og diskuterte hva som overordnet sett gjorde systemene forskjellig fra hverandre. Vi lagde presentasjonsslides og forberedte noen spørsmål og diskusjonspunkter til møtet.

Når vi ankom på campus Gløshaugen ble vi mottatt av Øystein Viggen og Erlend, som i hovedsak hadde møtet med oss. Vi følte det var produktivt ved at vi fikk demonstrert funksjonalitet og karakteristikk for hvert system. Slik det kom fram av diskusjonen, virket Ansible og Puppet mest interessant å gå videre med.

G.2.2 Hva vi skal gjøre framover

Vi går altså videre med å ytterligere teste Ansible og Puppet. Erlend har sendt oss ei liste over tjenester de ønsker å se konfigurasjon for. Dette består av essensielle konfigurasjoner for NTP og SSH, og mer spesifikke konfigurasjoner for web-servere. For web-serverkonfigurasjon har de foreslått 3 alternativer: Sympa, styring av Apache-moduler, konfigurering av Apache Virtualhosts for deres webhotell. Vi skal undersøke hvordan noe av dette kan gjøres i både Ansible og Puppet. Basert på det vi finner i denne delen av prosjektet, skal vi kunne komme med en konkret anbefaling.

G.3 Statusrapport 18.04.2018

G.3.1 Framgang siden forrige statusrapport (21.03)

Proof of Concept-oppsett (Eksempeloppsett)

Vi har fått jobbet en del med PoC-oppsettet siden vi kom fra møtet i Trondheim. Stein Ove har tatt seg av mesteparten av arbeidet med Ansible, Andreas har laget Heat-templates for bruk i SkyHiGh for både Ansible- og Puppet-stacker for å kunne teste begge systemene isolert. Andreas har også jobbet med å skrive Puppet-manifester for Apache-relaterte oppgaver. Både skriving av vhosts for enkelte web-tjenester, og vise hvordan en maskin for "webhotell" kan konfigureres i kombinasjon med Hiera, der alle data angående de mange vhosts lagres i sistnevnte. Da kan de simpelt slås opp i Puppetkoden og det kan én gang deklareres med klassen apache::vhosts der oppslaget av alle gjeldende vhosts inngår i en såkalt hash rett i denne ene klassedeklarasjonen.

Audun har jobbet med å finne ut hvordan noder kan grupperes og behandles ulikt derav i Puppet-systemet (både Hiera og i Puppet-koden). Dette har vist seg å være en kompleks oppgave, og kan gjøres på forskjellig vis, og, slik jeg gjorde, det kan hende man starter å se i en ende av systemet som ikke nødvendigvis gir en bra løsning på problemet. SSHd-konfigen er et eksempel på hvordan ulike maskiner og grupper av maskiner skal ha forskjellige innstillinger. Først virket "merging" i Hiera naturlig. Senere virket mønsteret "defaults with overrides" mer naturlig.

Enda utestår oppsett av Sympa på Puppet. Stein Ove fikk satt opp noe han selv mener er en hack (oppsett i Vagrant for en enkelt maskin, tilpasset vårt oppsett i SkyHiGh), og ikke et bra oppsett av en slik tjeneste, med Ansible. Det fins en modul på Puppet Forge, men har ikke blitt vedlikeholdt siden 2015, GitHub-brukeren eksisterer ikke, og støttes bare av Debian. I likhet med måten Stein Ove gjorde med sin "hack", så finnes det også et Puppet-manifest som visstnok skal sette opp alt nødvendig, og er utviklet av Sympa Community. Dette har ikke blitt testet enda. Det naturlige ville vært å fordele de ulike tjenestene og oppgavene som utgjør Sympa (httpd, db, postfix, sympa) i hvert sitt Puppet-manifest (f.eks sin egen profil-klasse). Skal man utvikle en egen Sympa-modul, eller skrive egne profil-klasser, forutsetter dette at man lærer seg Sympa tilstrekkelig til å faktisk gjøre dette (kunne manuelt sette opp og kjenne til en del funksjonalitet og konfigurasjon), noe vi ikke har fått gjort fram til nå.

Siden arbeidet framover nå i hovedsak vil være fokusert på rapportskriving, blir det ikke lengre så mye tid til å jobbe med PoC-oppsettet. Dette kan vi bruke kvelder og helger på i stedet, samt tiden etter at vi har levert rapporten.

Rapport

For rapportens del har vi skrevet ned ulike gjøremål: samle alle litteraturkilder, oppsummere arbeidet fram til nå slik at vi kan bestemme kapittelinndeling, finne alle nødvendige figurer og kodeeksempler vi har bruk for (slik at vi kan spørre om å få bruke de). Det meste av uke 16 vil nok i hovedsak bestå av å forberede skrivearbeidet, som vi sikter på å få startet med til neste uke.

G.3.2 Hva vi skal gjøre framover

I hovedsak skrive rapport, og jobbe det vi kan for å få med mer i PoC-oppsettet utenom planlagt arbeidstid.

H Møtereferater

H.1 Møter med oppdragsgiver (ved Erlend Midttun

H.1.1 17.01.2018 - Første møte med oppdragsgiver

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Erlend Midttun (Skype)
- **Sted:** Rom A132 - NTNU i Gjøvik
- **Agenda:** Første møte med oppdragsgiver

NTNU-IT har Linux-servere som de kjører CFEngine 2. Godt fornøyd, men har gått ut på dato (bakgrunn). Neste problem er at det er veldig mange alternativer der ute, kombinert med at det er ønskelig med at det ikke blir så mye arbeid.

Erlend ser det som at det er to deler i prosjektet. En rapport, en sammenligning av de ulike konf.systemene, RedHat, Ubuntu, CentOS. Windows er et eget løp, Unix-teamet har ikke noe med det å gjøre. Kriterier er nevnt i eposten (se epost/oppgavebeskrivelse). F.eks hva kan man skrive i stillingsutlysningen. Finnes det kompetanse i markedet? Det er en kjempefordel om det er OpenSource, de har hatt dårlige erfaringer med kommerse løsninger. Krav til sikkerhet er også nevnt. Audun: Vi har vært inne på mye av disse punktene i idemyldring og utdanning. Del 2 er et "proof of consept". Skal vi velge en kandidat?

Erlend: Ja. Vi har foreslått noen kandidater. Det må ikke være noen av de som står på lista. Eksempler på tjenester som konfigureres er også nevnt i eposten.

Audun: Spørsmål fra den originale oppgavebeskrivelsen: Doku/opplæring/kurs: krav til simpelhet eller funksjonsrikt?

Erlend: Ikke sikker på om man kan svare på spørsmålet. CFEngine2, arbeidsflyt, lager konfigfiler, lages av kopieres inn i CFEngine, og distribueres via CFEngine. Har litt kjennskap til Puppet, men fikk litt negative følelser. Mener kompetansen må være ganske høy for å kunne bruke Puppet - lære seg moduler, profiler, avhengigheter etc... Enkelhet er et viktig moment, mange ansatte vil ikke bruke så mye tid på dette.

Audun: Har andr erfaring med lignende systemer? Erlend: EN til har vært litt borti Puppet, men det er ikke noe preferanse for å velge Puppet.

Audun: Puppet har moduler... vedlikehold.. (Dette kan vi se mer på). Audun: Hva menes med punktet om "grunnoppsett" Erlend: Boot -> Setter IP addresser. CFEngine tar seg av resten. Gjør det fra boksen med konfigurert IP til en ferdig oppsatt tjeneste, basisoppsett. Audun: Oppsett av et knippe tjenester, er dette noe som går ut over basisoppsettet? Erlend: Riktig, feks Apache webserver.

Audun: oppgradering av de konfigurerte tjeneste, menes det å kunne bruke konf.sys til å spesifisere versjonen?

Erlend: f.eks Apache, forskjell mellom Ubuntu 14.04/16.06. Trenger forskjellige konfigfiler. Hvis man oppgraderer versjon trenger man konfigfiler tilpasset OS.

Audun: Bruker dere GIT?

Erlend: Vi bruker GIT. To bokser per i dag. Brukere logger inn på utsjekket git-repo, individuelt per bruker. Pusher endringer til konfig-server. Sendes ut mail til drift med hva som er gjort.

Audun: Puppet har god integrering med Git som et eksempel.

Audun: Har dere fulgt noen spesielle prinsipper/best practice? IaC? .... Testing gjøres på selve

Erlend: Har ikke noen form for testing i dag. Høres litt kult ut. Det eneste vi har der er på DHCP. Er en hook via Git.

Audun: Vil dere ha et GUI?

Erlend: Ikke veldig GUI-fokusert.

Audun: Har dere vurdert andre arbeidsflyt-mønstre, f.eks Docker?

Erlend: Containere er ikke blitt sett på som veldig relevant for den jobben de skal gjøre. Ikke egentlig nei...

Andreas: Kan vi få CFEngine filer?

Erlend: Skilt ut i hemmeligheter, ikke hemmeligheter. Det som ikke er hemmelig bør være mulig å få til. Erlend noterer seg dette.

Audun: Antar dere vil ha statusmøter underveis. Etter at vi har passert en milepæl... Hva med circa en gang per måned.

Erlend: OK

Andreas: Vi kan dele tilgang til repoet

Erlend: Det hadde vært kjekt. Tror dette blir bra.

Audun: har dere OpenStack?

Erlend: Har et miljø, men bruker ikke det så mye

Audun: Dere får koden uansett.

Audun: Vi prøver å lage et testmiljø basert på noen av de tjenestene dere har nevnt.

Erlend: bra

H.1.2 15.02.2018 - Møte med oppdragsgiver

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Erlend Midttun (Skype)
- **Sted:** Rom A132 - NTNU i Gjøvik
- **Agenda:** Statusrapport

Oppsummering av arbeidet overfor Erlend. Mye av dette er forklart i Statusrapport 14.02.2018

H.1.3 16.03.2018 - Møte med oppdragsgiver i Trondheim

- **Tidspunkt:** 09:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Erlend Midttun og Øystein Viggen
- **Sted:** Møterom, NTNU IT Gløshaugen, Trondheim
- **Agenda:** Infomøte om utvalget og vurderinger

Det diskuteres nærmere hvordan CFEngine i dag anvendes av Unix-gruppen. Blant annet gruppering/klassifisering, hvordan maskiner kan være like, bortsett fra enkelte ting som lisensfiler. De styrer over 400 maskiner. Prosjektgruppen presenterer funn fra vurderingsprosessen om Ansible, Puppet, Chef og Salt. Puppet og Ansible er de mest kjente navnene. Puppet har blitt brukt eller brukes av enkelte i Unix-gruppen allerede.

Det blir bestemt at Ansible og Puppet blir tatt videre til eksempeloppsett. Vi takker for hyggelig møte.

H.2 Møter med veileder, Eigil Obrestad

H.2.1 11.01.2018 - Innledende veiledningsmøte

- **Tidspunkt:** 12:00 - 12:30
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Vi innleder prosjektarbeidet med dette møtetet. Tidspunkter for framtidige veiledningsmøter blir satt til torsdager klokken 12:30. Eigil informeres om gruppeleders kontaktinformasjon. Det er viktig at vi kommuniserer med IT-avd. i Trondheim gjennom Erlend, for bedre forståelse av problemet og oppgaven vi har foran oss. Selv om det er tidlig i prosjektarbeidet, kan det være lurt å se på noen kjente systemer allerede på dette tidspunktet. Vi skal også starte å se på tidligere bacheloroppgaver og annen teori.

H.2.2 18.01.2018

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Kan bli mye med eksempeloppsett for alle de forskjellige CM-systemene som finnes der ute. Sikkerhet diskuteres, burde vurdere hvordan hemmeligheter lagres, sikker overføring av data, autentisering og tilgangskontroll. Ressurser i Openstack tildeles og et prosjekt opprettes for oppgaven.

H.2.3 25.01.2018

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Får tilbakemelding om utkastet av prosjektplanen som ble sendt dagen før til Erlend og Eigil. Lager punktliste av tilbakemeldingene vi får her.

H.2.4 01.02.2018

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Bestemmes at det lages en kravspesifikasjon og vurderingsliste før vurderingen, for å være mest mulig objektiv. Det noteres ned flere forslag til vurderingselementer og vektleggingsskalaer. Muligens også skrive en spørreundersøkelse for å kartlegge Unix-gruppens preferanser.

H.2.5 08.02.2018

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Tankekart om krav og kriterier er ferdigskrevet. Prater om å ta en tur opp til Tronheim på et senere tidspunkt for å presentere kandidatene. Diskuterer om den initielle rangering basert på utgivelsesdato og brukerstatistikk er troverdige og omfattende nok. Nevnes at brukermønsteret de er vant med kanskje kan være veldig annerledes fra nyere bruk av CM-systemer.

H.2.6 22.02.2018

- **Tidspunkt:** 12:00 - 12:30
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Planen er ganske klar framover, så møte ble kort. Det Bestemmes at det individuelle arbeidet med hvert sitt CM-system skal være ferdig innen Uke 9 og jobbe sammen med Puppet i uke 10. Deretter forberedes presentasjonen for møte i Trondheim.

H.2.7 08.03.2018

- **Tidspunkt:** 12:30 - 13:00
- **Tilstede:** Stein Ove Jernberg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Vi utsetter neste møte til 22. mars, siden vi reiser opp til Trondheim denne dagen. Diskusjon om dette møtet skjer altså da. For møtet er det viktig at vi planlegger noen temaer for diskusjon og spørsmål vi selv har til de. Vi bør altså få best mulig oversikt over deres arbeidsflyt og metoder i bruken av CFEngine. Dere forventninger bør også komme fram fra dette møtet. I utgangspunktet er tidsrommet fra 09:00 fredag den 16. mars satt av til dette møtet.

H.2.8 22.03.2018

- **Tidspunkt:** 12:30 - 13:00
- **Tilstede:** Stein Ove Jernberg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Arbeidet med eksempeloppsett er godt i gang. Ansible har nodeklassifisering slik Unix-gruppa allerede bruker med CFEngine. Med Puppet er ikke dette “rett fram” ut fra grunnoppsett og bruk av site.pp for mapping av klasser til noder. Bruker man en ENC, kan man oppnå dette. Foreman er én type ENC, men man kan lage sin egen bare ved bruk av et script som skriver ut nodeklassifisering i YAML-format. På denne måten kan man skrive gruppering/klassifisering i en annen database (eller så enkelt som en ren tekstfil, strukturert eller ikke) for så å la Puppet bruke scriptet til å angi hvilke klasser en node skal ha basert på det.

I tillegg med Puppet kan Hiera brukes for å gruppere konfigdata. Fra mest spesifikk nodedata til mest generell. Enten kan dataene repeteres med komplett listing av konfigdata, eller den kan slås sammen med flere.

H.2.9 09.04.2018

- **Tidspunkt:** 12:30 - 13:00
- **Tilstede:** Stein Ove Jernberg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Nodegruppering- og klassifisering er kanskje det viktigste angående CM-systemer. Det er viktig å vise hvordan dette kan gjøres. Vi har ikke bestemt oss for å velge én av de. Vi fortsetter med både Ansible og Puppet. Til slutt skal en anbefaling gjøres basert på våre undersøkelser og erfaring. Denne anbefalingen likevel ikke nødvendigvis det viktigste ved rapporten. Valg gjort underveis og eksemplene er det viktigste.

Vi bør begynne å tenke over hvordan rapportens struktur skal se ut. Det er også lurt å bruke vektorgrafikk i figurer i størts mulig grad. Vi bør altså tenke på hva som står igjen av arbeidet med eksempeloppsett og avslutte om noen ukers tid når vi starter med skriving av rapporten.

H.2.10 26.04.2018

- **Tidspunkt:** 12:30 - 13:00
- **Tilstede:** Stein Ove Jernberg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Eigil har fått lest første utkast av rapporten og kommet med en del kommentarer. Noen definisjonsproblemer fra teoridelen bør ordnes. Bruk norske ord og uttrykk i størst mulig grad. Ikke bruk personlige pronomen. Noen steder er tynt utfylt, eksempelvis i kravspesifikasjon og vurderingskriterier, og bør utfylles. Noen stikkord

og begreper bør forklares etter at de er brukt. Referer til vedlegg. Legg ting som tar mye plass i teksten til vedlegg. Kanskje ikke nødvendig med punktlister enkelte steder. Del diskusjon i egne kapitler, eller det passer seg i forhold til presentasjon av resultater. Kodeeksempler bør bli til figurer eller kodelistinger, slik at de kan refereres til ellers i teksten. Kodelistinger kan også få nummerering. Få med liste over figurer, tabeller og listinger i begynnelsen av rapporten.

For framtidige utkast må vi bemerke de delene av rapporten som er nytt eller har hatt vesentlige endringer fra forrige gang.

H.2.11 03.05.2018

- **Tidspunkt:** 12:30 - 13:00
- **Tilstede:** Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Dato for presentasjon er ikke fastsatt enda. Vi kan kanskje snakke med Hilde Bakke eller Tonje Trønnes om dette. Vi bør snart invitere oppdragsgiver nedover til én av disse dagene. Også viktig at vi skrive publiseringsavtale hvis vi ønsker å la studenter eller andre interesserte lese rapporten i ettertid.

H.2.12 09.05.2018

- **Tidspunkt:** 13:00 - 14:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Vi kan levere et siste utkast enten førstkommende fredag eller på mandags morgen etter helgen. Vi velger å gjøre sistnevnte siden vi kan bruke helgen til å gjøre flere endringer. Neste, og siste møte før innlevering, blir altså senere den mandagen klokken 13:45. Målet er uansett å bli ferdig med all skriving før fredag, og gjøre finpuss og rydde opp i helgen og påfølgende ukedagene før innlevering.

Vi må få med dokumentasjon av oppmøtetider og se hvordan tidsplan fra prosjektplanen ble fulgt. Rapporten har kanskje noen unødvendig lange formuleringer. Prøv å finn essensen og kort ned til passende lengde. Vi må også huske

holde en rød tråd gjennom hele rapporten. Vi har også bestemt oss for å la noen andre lese rapporten, mot at vi leser deres.

H.2.13 14.05.2018 - Siå ste møte før innlevering av rapport

- **Tidspunkt:** 12:00 - 13:00
- **Tilstede:** Stein Ove Jernberg, Andreas Osborg, Audun Huseby, Eigil Obrestad
- **Sted:** Eigils kontor

Vi får siste kommentarer om rapporten. Det er ikke mye mer å si, siden rapporten i hovedsak er ferdig. Vi vil nå fokusere på å legge inn manglende elementer, ordne referanser, gå gjennom og korrigere feil.

I Opprinnelig oppgavebeskrivelse

Nytt konfigurasjonssystem for Linux-servere

Oppdragsgiver: NTNU IT

Kontaktperson: Arne Dag Fidjestøl <adf@ntnu.no>

NTNU IT må på sikt bytte ut dagens konfigurasjonssystem for Linux-serverene våre med noko som er vedlikeholdt. Dagens verktøy er Cfengine versjon 2 og det er nokre år sidan den var vedlikeholdt.

Men kva for konfigsystem skal vi gå for?

Det hadde vore fint med ein evaluering av fleire konfigurasjonsystem basert på i alle fall følgande kriterier:

* Dokumentasjon/opplæring/kurs
* Brukerbase
* Basis-oppsett av server. Nettverk, programvarepakker osv.
* Oppsett av eit knippe tjenester.
* Oppgradering av konfigurasjonssystemet
* Oppgradering av dei konfigurerete tenestene
* Sikkerhet
* Generell driftbarhet. F.eks. versjonskontroll, sporbarhet, epost om endringer

Vi ser for oss noko slikt som at ein set opp eit knippe (virtuelle) maskiner og tester dei ulike mulighetene derifrå.

J Prosjektplan

BDR3900 - Konfig.sys. Linux
Prosjektplan

Audun Huseby
Stein Ove Jernberg
Andreas Osborg

Januar 2018

Innhold

Kapittel 1

Bakgrunn

Konfigurasjonsstyring er en praksis for å få kontroll på, og følge opp, inventar som en organisasjon eier. Hvis inventaret har endret seg, må konfigurasjonen (oversikten over inventaret) oppdateres, og hvis konfigurasjonen definerer en ny tilstand, må denne følges opp og endringer i inventaret må foretas. Dette er en praksis som har blitt brukt i forsvarsorganisasjoner i flere tiår[2].

I sammenheng med IT-systemer benyttes konfigurasjonsstyring for å utføre oppgavene som omfatter å oppnå den ønskede tilstanden for programvare. Konfigurasjonsstyringssystemer (CM-systemer) sørger for at systemadministratorer kun trenger å definere tilstanden de ønsker, så vil CM-systemet sørge for at tilstanden blir satt i drift. CM-systemer, og automatisering generelt, vil være til stor nytte når man administrerer store antall servermaskiner (fysiske og virtuelle), slik at konfigurasjon forblir konsistent.

Unix-gruppen ved NTNUs IT-avdeling i Trondheim benytter per i dag CFEngine[1] versjon 2 for synkronisering av konfigurasjoner på deres Linux-maskiner. Systemer som CFEngine kan benyttes når mange maskiner (fysiske eller virtuelle servermaskiner) administreres. Konfigurasjoner deklareres i CFEngine, som følgelig vil sørge for at dette blir reflektert på de maskinene som skal ha den aktuelle konfigurasjonen. Det gjør at jobben med konfigurasjonsstyring (kontroll på "inventaret", i dette tilfellet programvare) blir en enklere oppgave for administratoren. Det er en automatisert oppgave med enkle forutsetninger: Konfigurasjonsstyring handler om å sjekke egne ressurser mot en ønsket tilstand, og gjøre endringer om den ønskede tilstanden har forandret seg siden sist sjekk.

Versjonen 2 av CFEngine ble først lansert i 1998 og leverandørens støtte for systemet ble avsluttet for en stund tilbake. I dag tilbys CFEngine i versjon 3, men denne versjonen er ikke bakoverkompatibel med versjon 2. Problemet er at Unix-gruppen ikke kan oppgradere CFEngine uten å måtte skrive helt nye konfigurasjonsfiler. Siden det i dag

[1] https://cfengine.com/

finnes mange konkurrenter til CFEngine, kan man like gjerne ta en vurdering av disse også.

Unix-gruppen valgte høsten 2017 å komme med et forslag om oppgave til et bachelorprosjekt basert på denne problemstillingen. Vi i prosjektgruppa fattet interesse for denne, tok kontakt med oppdragsgiver og inngikk kontrakt.

1.1 Om oss

Vi er en gruppe på tre studenter på studieprogrammet Bachelor i drift av nettverk og datasystemer (BDR). Våre navn er Audun Huseby, Stein Ove Jernberg og Andreas Osborg. Alle tre er interesserte i IT-drift, og har fersk opplæring i bruk av Puppet i emnet IMT3005 Programmerbar infrastruktur[2], der selve praksisen for programmerbar infrastruktur (på engelsk omtalt som Infrastructure as Code) blir undervist av Erik Hjelmås[3].

Fra tidligere har vi hatt emnet IMT3441 Database- og applikasjonsdrift[4] med Kyrre Mathias Begnum[5] (nå omdøpt til IMT3003 Drift av tjenestearkitekturer[6]). Der lærte vi i hovedsak om elementer og mønstre i større tjenestearkitekturer, og gjorde alle oppgaver manuelt. Sånnsett hadde en naturlig overgang til IMT3005 der vi lærte at vi kunne automatisere alle oppgavene vi gjorde i IMT3441.

[2] https://www.ntnu.no/studier/emner/IMT3005
[3] https://www.ntnu.no/ansatte/erik.hjelmas
[4] ff
[5] https://www.ntnu.no/ansatte/kyrre.begnum
[6] https://www.ntnu.no/studier/emner/IMT3003

Kapittel 2

Rammer og mål

Denne delen inneholder beskrivelse av hva prosjektet skal oppnå, innenfor gitte rammer og begrensninger vi skal jobbe for å oppnå dette.

2.1 Økonomiske rammer

Prosjektet omfatter ikke innkjøp av materiell eller programvarelisenser. Litteratur kjøpes på individuell basis eller at kostnadene deles likt mellom medlemmene i gruppen. En økonomisk ramme eksisterer altså ikke for dette prosjektet. Eventuelle reisekostnader for gruppemedlemmene dekkes av oppdragsgiver, i tråd med avtalen.

2.2 Tidsmessige rammer

- Hoveddokumentet, rapporten, skal leveres den 16. mai 2018. Alt av arbeid skjer altså fra oppstart den 10. Januar til leveringsdato.
- Prosjektplanen, dette dokumentet, skal leveres den 1. Februar

2.3 Kravspesifikke rammer

Oppdragsgiver spesifiserer noen konkrete krav og retningslinjer som må tas i betraktning når vi gjør vår vurdering av ulike konfigurasjonsstyringssystemer (CM-systemer).

- Vurdering av tilgjengelighet dokumentasjon og opplæring
- Vurdering av brukerbase/markedsandel

- Systemet skal ha åpen kildekode
- Sikkerhet
  - Vurdering av tilgangsstyring til konfigurasjoner
  - Systemet skal ha sikker lagring av hemmeligheter (f.eks passord)
  - Systemet skal ha autentisering mellom klient og tjener
  - Systemet skal kunne styre brannmur på klientmaskin
- Systemet skal skalere bra
- Vurdering av tjener-til-klient-mekanisme ("push" eller "pull")
- Oppgraderbarhet
- Systemet skal kunne brukes med versjonskontrollsystemer (f.eks Git)

Oppdragsgiver har nevnt hva de ønsker vi leverer når prosjektet er ferdig

- Rapport med faglig vurdering
- Fungerende eksempel (Proof of Concept) av et system vi velger ut. Dette vil være vår anbefaling av et CM-system. I eksempelet skal vi kunne styre noen av følgende tjenester:
  - Apache httpd, sshd, ntpd/chrony, mysql, DNS-server, openldap-server, brukerautentisering mot LDAP (at.ntnu.no), eventuelt bruk på Linux desktop-maskiner.

2.4 Prosjektmål

Målet med dette prosjektet er å gjøre en faglig vurdering og anbefaling av ulike konfigurasjonsstyringssystemer (configuration management systems, CM-systemer) for maskiner som kan være til nytte når Unix-gruppen internt skal velge et system de mener passer best for dem. Anbefalingen skal demonstreres med et fungerende eksempel, eller Proof of Concept slik det gjerne blir omtalt.

2.5 Effektmål

Målene for oppnådd effekt er det vi i prosjektgruppa ønsker å se når et nytt system settes i drift. Vi skal ta disse målene i betraktning når vi gjør vår vurdering. Det er ikke sikkert vi har mulighet for å måle de punktene vi har beskrevet som en del av prosjektet.

- Mer effektiv og hensiktsmessig arbeidsflyt, blant annet med økt kvalitet som følge av testing/validering av kode som definerer konfigurasjon
- Forenkle og effektivisere feilsøking
- Sikkerhet
  - Sikrere lagring og distribusjon av hemmeligheter for konfigurasjoner
  - Sikkert forhold mellom CM-server og mottaker av konfigurasjon
- Et system som skalerer bra, i forhold til antall maskiner som administreres gjennom CM-systemet

Kapittel 3

Omfang

Oppdragsgiver beskriver deres situasjon med bruk av CFEngine versjon 2 som fungerende, og er stort sett fornøyd med arbeidsflyt og funksjonalitet. Likevel er det et problem at denne versjonen ikke lengre støttes av leverandør, i tillegg til at den nyere versjon 3 ikke er bakoverkompatibel med konfigurasjoner skrevet for versjon 2. I søken av et nytt system er de overveldet av mulighetene, og har simpelthen ikke formening eller anledning til å ta en intern vurdering og beslutning. Det passet dermed utmerket at vi i prosjektgruppa kunne ta på oss oppdraget med undersøking, vurdering og testing av de ulike CM-systemene.

3.1 Oppgavebeskrivelse

Gruppa skal på vegne av oppdragsgiver gjøre en sammenligning av ulike konfigurasjonsstyringssystem for Unix-plattform, komme med en konkret anbefaling av et system og utvikle et Proof of Concept. Vår problemstilling er dermed:

Hva er det beste konfigurasjonsstyringssystem for Unix-gruppa ved NTNU-IT i Trondheim?

Det Unix-gruppen trenger er altså den faglige vurderingen av et system og en demonstrasjon av hvordan ett av systemene kan rulles ut, brukes i hverdagen og i seg selv driftes. Det krever altså at vi leverer to ting til oppdragsgiver: Rapporten med vurdering, og koden til samt en demonstrasjon av Proof of Concept. Vår vurdering skal selvfølgelig baseres på Unix-gruppens krav, men vi vil også gjøre våre egne subjektive vurderinger for bruk og arbeidsflyt.

Til å begynne med skal vi lage en kravspesifikasjon for å samle alle krav og forutsetninger for vurdering av kandidatene. De krav som er nevnt i dette dokumentet, er oppdragsgivers konkrete krav (se 2.3), mens vi vil selv legge til noen som vi mener er hensiktsmessige angående eksempelvis spesifikk funksjonalitet og dokumentasjon av systemet. Kravspesifikasjonen vil sørge for at mange kandidater vil kunne siles ut

i flere runder. Det vil si at vi har noen generelle krav til å begynne med, og deretter mer spesifikke krav. Til slutt vil vi forhåpentligvis sitte igjen med de kandidatene som tilsynelatende oppfyller alle krav, men som krever vår subjektive vurdering når det kommer til bruk og arbeidsflyt.

For å gjøre en god vurdering av de aktuelle kandidatene, må vi tilegne oss et grundig inntrykk av hver av de. Vi sikter ikke på at alle gruppemedlemmene skal lære seg alle systemene, men heller at vi velger ut én eller to kandidater hver av oss kan gjøre et dypdykk i. Deretter vil vi kunne rapportere og presentere våre funn for resten av gruppen. Når vi gjør våre egne vurderinger, må vi likevel ha en plan for hva som skal vurderes, eller en slags sjekkliste der vi skal kunne beskrive eksempelvis ulik funksjonalitet og arbeidsflyt.

Den siste delen av arbeidet vil omfatte å etablere Proof of Concept-oppsettet. Før vi går i gang med dette vil vi kommunisere med oppdragsgiver og veileder før vi i prosjektgruppen best mulig kan ta en avstemning for hvilket system som vil passe Unix-gruppen. Proof of Concept-oppsettet skal inneholde tjenester de har i drift ved NTNUs IT-avdeling, som Unix-gruppen har ansvaret for. Disse tjenestene skal styres av det CM-systemet vi har valgt.

Alle våre resultater skal overleveres i rapporten og i form av nødvendig definisjonskode for Proof of Concept.

3.2 Problemområde

CM-systemer løser i hovedsak to problemer som på engelsk blir kalt "configuration drift" og "snowflake servers". Førstnevnte beskriver en uønsket tilstand der konfigurasjoner for et sett med maskiner som i utgangspunktet var konfigurert identisk, har blitt konfigurert forskjellig som følge av "ad-hoc" endringer på enkeltmaskiner over tid. Maskinene gir kanskje lik funksjon, men konfigurasjonen er ikke lenger identisk, slik det var ment. Sistnevnte beskriver en tilstand der alle ad-hoc-endringene har akkumulert til en såpass stor grad at den har endt opp som en spesiell maskin. Eller at maskinen var spesiell og unik i utgangspunktet, noe som er uønsket når man styrer flere hundre, kanskje tusen maskiner. Snowflake servers er ikke uvanlig, men bør begrenses til de man bruker som kontrollere av andre maskiner eller "jumphost" til et nettverk.

Dette er problemstillinger Kief Morris har beskrevet i [1]. CM-systemer sørger for å holde konfigurasjoner konsistent, og for å forhindre ad-hoc-endringer på et sett med like maskiner. Noe annet som er viktig med CM-systemer er at konfigurasjonene kan skrives i strukturerte filer, som YAML, JSON eller DSL (domain specific language). Disse filene bør være rene tekstfiler som kan versjonskontrolleres i f.eks Git. Unix-gruppen ved NTNUs IT-avdeling benytter i dag versjonskontroll for konfigurasjoner med CFEngine.

Det å definere elementer av infrastrukturen i tekstfiler, legge tekstfilene i VCS (version control system), og la et CM-system til å synkronisere konfigurasjon, er alle prinsipper av *Infrastructure as Code*[1], eller *programmerbar infrastruktur* som det heter på norsk.

3.3 Avgrensing

Prosjektets fokus er hovedsaklig på å foreslå et CM-system som passer til Unix-gruppen ved NTNUs IT-avdeling sine kriterier. Vi skal ikke utvikle en løsning som skal implementeres direkte, men bare utvikle et eksempel som kan vise og være til eksempel på bruk og arbeidsflyt av et slikt system. Siden bruk av CM-systemer som regel følger en spesiell arbeidsflyt, kommer vi til å skrive teori om dette, men det er ikke dermed sagt at Unix-gruppen skal anvende vår praksis. Det skal heller være til inspirasjon for å utvikle arbeidskulturen i et lengre perspektiv.

Kapittel 4

Prosjektorganisering

4.1 Ansvarsforhold og roller

4.1.1 Prosjektgruppa

- Audun Huseby (Gruppeleder)
- Stein Ove Jernberg
- Andreas Osborg

Prosjektgruppa blir å fordele arbeidsoppgaver mellom seg utover i prosjektet. Audun er gruppeleder og kontaktperson for gruppen. Han kan kontaktes på epost[1].

4.1.2 Oppdragsgiver

Oppdragsgiver er Unix gruppen ved NTNUs IT-avdeling i Trondheim. Vår kontaktperson der er Erlend Midttun[2], som er teamleder for Unix-gruppen.

4.1.3 Veileder

Vår veileder er Eigil Obrestad[3]. Eigil er universitetslektor for IIK under IE-fakultetet ved NTNU. Han vil bistå med veiledning angående organisering og gjennomføring av prosjektet, i tillegg til faglig veiledning.

[1] audunhu@stud.ntnu.no
[2] https://www.ntnu.no/ansatte/erlend.midttun
[3] https://www.ntnu.no/ansatte/eigil.obrestad

4.2 Regler, rutiner og formelle bestemmelser

Gruppereglene finnes i vedlegg B, og må signeres av hvert enkelt gruppemedlem. Rutinene omfatter i hovedsak rent formelle bestemmelser. Et eget dokument beskriver de formelle bestemmelsene i vedlegg A. Dette dokumentet beskriver, blant annet, forventet oppmøtetidspunkter.

4.3 Verktøy

Verktøybruk er viktig for å hjelpe oss med organisering. Her er de vi har valgt

- Oversikt over arbeidsoppgaver: **Trello**[4]
- Ukentlig tidsplanlegging: **Google Calendar**[5]
- Modell/verktøy for overordnet tidplanlegging: **Gantt-skjema/MS Office Project**[6]
- Fildeling og samskriving: **Google Drive/Docs**[7]
- Verktøy for rapportskriving: **ShareLaTeX**[8]
- Oppbevaringssted for kode: **BitBucket**[9]

[4] https://trello.com/
[5] https://calendar.google.com/
[6] https://products.office.com/nb-no/project/
[7] https://www.google.com/intl/no/drive/
[8] https://www.sharelatex.com/
[9] https://bitbucket.org/

Kapittel 5

Arbeidsplan og metodikk

5.1 Hovedinndeling av arbeidet

Prosjektet skal deles inn i tre hoveddeler, utover selve planleggingen. Prosjektets første del går ut på å gjøre en vurdering og noen enkle tester av aktuelle kandidater basert på en kravspesifikasjon. Den andre delen omfatter å velge ut en kandidat og lage et fungerende eksempel (Proof of Concept). Den siste delen gjelder kun rapportskriving. I tidsplanleggingen defineres noen milepæler. Under er utdyping av de ulike delene av prosjektet.

5.1.1 Del 0: Forprosjekt

I forprosjektet skal vi komme i gang med arbeidet. Vi etablerer kontakt med veileder og oppdragsgiver. Vi sørger også for å ta i bruk støttesystemer for prosjektarbeid. En prosjektplan og avtale med oppdragsgiver skal leveres den 1. Februar.

- **Milepæl 1: Frist for prosjektplan (1. februar)**

5.1.2 Del 1: Undersøkelse og individuell testing

I denne fasen skal gruppa først skrive en kravspesifikasjon basert på oppdragsgivers krav og de krav vi i prosjektgruppen legger til. Deretter skal vi identifisere flere mulige alternativer, for så å gjøre en siling for å redusere antall kandidater. Denne vil være basert på kriterier som f.eks antall brukere, antall utviklere, størrelse på brukerbase, stabile eiere/bakmenn, oppdateringsfrekvens. De gjenstående kandidatene (<10) vil bli vurdert nærmere ut i fra oppdragsgiver sine kriterier og eliminert til det gjenstår 3-5 reelle alternativer som blir gjenstand for den siste utvelgelsesprosessen. Der er målet å finne fram til en kandidat som gruppa kan anbefale. Den siste utvelgelsesfasen vil

også innebære praktisk jobbing med verktøyene for å få erfaring med hvordan de fungerer i praksis, men ikke fullverdig, produksjonsklart oppsett av alle kandidater. Dette arbeidet gjør vi i SkyHiGh, som er en IaaS/skyløsning vi kan styre virtuelle maskiner, nettverk og andre ressurser vi trenger for arbeidet.

- **Milepæl 2: Utsiling av kandidater gjort**
- **Milepæl 3: Vurdering av aktuelle kandidater ferdigstilt**

5.1.3 Del 2: Valg av beste kandidat og utvikle Proof of Concept

Når gruppa har valgt den kandidaten vi står inne for og vil anbefale for oppdragsgiver skal det lages et Proof of Concept. Dette vil innebære å sette opp et testmiljø der CM-systemet brukes til å konfigurere servere, samt sette opp et knippe tjenester, som Unix-gruppen administrerer i dag. I tillegg skal det også presenteres et eksempel på hvordan arbeidsflyten kan legges opp gjennom bruk av CI/CD prinsipper (med bruk av f.eks Git, automatisert testing/utrulling). Dette eksempelet skal demonstreres i SkyHiGh.

- **Milepæl 4: Valg av beste kandidat**
- **Milepæl 5: Ferdigstilling av fungerende eksempel (Proof of Concept)**

5.1.4 Del 3: Rapportering

Til slutt skal resultatene fra både del 1 og del 2 presenteres i den endelige rapporten. Her beskrives underliggende teori, ulike konsepter og generell arbeidsflyt knyttet til CM-systemer. Resultatene fra testing av våre utvalgte kandidater skal beskrives, og en mer dyptgående beskrivelse av Proof of Concept-oppsettet. Til slutt diskuteres, blant annet måloppnåelse, resultatene og forslag til videre arbeid.

- **Milepæl 6: Levering av rapport**

5.1.5 Plan for gjennomføring

Et Gantt-diagram visualiserer prosjektets overordnede tidsplanlegging. Som vist er tiden stykket opp i tre hoveddeler, som nevnt over i 5.1, med et forprosjekt. Se vedlegg C

5.2 Arbeidsmetodikk

5.2.1 Karakteristika ved prosjektet

Prosjektet skal vurdere forskjellige CM-systemer og velge ut ett. Etterhvert som antall kandidater reduseres vil systemene bli gjenstand for nærmere undersøkelse. Når antallet er redusert til de siste 3-5, vil det være naturlig å ta for seg et og et system om gangen og jobbe en periode/sprint med hver av disse. Prosjektet er på en måte planmessig i at vi har ganske god oversikt over hva som skal gjøres og hva sluttresultatet skal bli, men på en annen side vet vi ikke hvilket system som blir valgt og heller ikke detaljene rundt "Proof of Consept" som skal lages. "Proof of Consept" blir det siste som gjøres før prosjektrapporten sluttføres og denne kommer til å bli jobbet med på en mer "smidig" måte, der gruppa kan få en del input fra oppdragsgiver om ønsket oppsett.

5.2.2 Argumentasjon for valg av arbeidsmetodikk

Når det kommer til valg av modell er det ikke en metode som skiller seg direkte ut. Vårt prosjekt vil kunne jobbes med på både en planmessig og smidig måte. Begge vil føre oss i mål med et tilfredsstillende sluttresultat. Vi velger å bruke Scrum som systemutviklingsmodell for prosjektet vårt. Mye av årsaken til dette er at vi er nysgjerrig på å prøve hvordan Scrum fungerer i praksis i et virkelig prosjekt. Scrum tilrettelegger også for å dynamisk endre prosjektets mål gjennom utviklingen for å ende opp med å dekke oppdragsgivers ønsker.

5.2.3 Metodevalg og tilnærming

Gruppa skal tilpasse Scrum-metoden til å passe inn i sitt prosjekt. I og med at vi bare er tre personer på gruppa blir vi å følge en slags lettversjon av Scrum med mindre tid til møter og planlegging.

Sprint Lengde 1 uke (evt. 2 uker, hvis arbeidsmengden tilsier). Starter tirsdag kl. 10.00 og varer t.o.m tirsdag kl. 10.00 uka etter

Sprint Review Meeting / Sprint Planning Meeting Sammenslått. Tirsdager starter med sprint review meeting med sprint planning etterpå. Oppdragsgiver og veileder deltar i utgangspunktet ikke på disse møtene.

Product Backlog Oversikt i Trello over alt som skal gjøres

Daily Scrum Kort prat i starten på hver møtedag om hva vi har gjort/skal gjøre

Scrum master Samme som gruppeleder

Gruppa har ikke definert noen flere roller enn gruppeleder/kontaktperson.

Vi henter også inspirasjon fra andre systemutviklingsmodeller

- Sustainable pace (XP[1]). Vi prøver så fremt det er mulig å følge vanlig arbeidsuke, tirsdag-fredag. Det forventes imidlertid at i perioder må også lørdag/søndag brukes for å rekke deadlines.
- Felles eierskap til produktet (XP).
- Kanban-board (Trello).
- Selv om vi har sprinter vil en del av arbeidet vi gjør flyte litt over i hverandre, men med ulik vekting, litt sånn som slik som fasene i RUP[2] gjør. Eksempelvis vil rapporten være noe som jobbes litt med under hele prosjektets periode, i form av undersøkelser og datainnsamling, men arbeidet vil intensiveres på slutten under selve skriveprosessen.

[1] http://www.extremeprogramming.org/
[2] http://sce.uhcl.edu/helm/RationalUnifiedProcess/

Kapittel 6

Kvalitetssikring

6.1 Kodekvalitet

For å sikre vår egen kode mot tap og versjonskontroll, vil vi bruke Git-systemet med BitBucket for å oppbevare koden som skrives for OpenStack Heat-filer og definisjonsfiler for alle de utprøvde kandidatene for CM-systemer.

Dokumentasjon av koden vil finnes i selve koden som kommentarer, og som analyse og beskrivelse for det som til slutt skal gå til input til den endelige rapporten. Definisjonskoden er deklarativ (semantisk beskrivende) , og kan ansees som selvdokumenterende. Koden vi skriver følger ingen spesielle kodestandarder, annet enn det som er spesifisert som anbefalt syntaks av de ulike CM-systemene (linting).

6.2 Risikoanalyse av prosjektet

Vi har gjort en analyse for ulike risikoer angående gjennomføring av selve prosjektet. Se vedlegg D

Bibliografi

[1] MORRIS, K. Infrastructure as code : managing servers in the cloud, 2016. 11, 12

[2] WIKIPEDIA. Configuration management — wikipedia, 2017. [Online; accessed 30-January-2018]. 5

Tillegg A

Formelle bestemmelser

Formelle bestemmelser

Involverte personer	
Prosjektgruppens medlemer	• Audun Huseby (audunhu@stud.ntnu.no) • Stein Ove Jernberg (steinoj@stud.ntnu.no) • Andreas Osborg (andreosb@stud.ntnu.no)
Prosjektleder	Audun Huseby
Veileder	Eigil Obrestad
Oppdragsgiver	NTNU IT, i Trondheim
Oppdragsgivers kontaktperson	Erlend Midttun

Grupperegler
Alle medlemmer av prosjektgruppen må signere dokument (se vedlegg 'Grupperegler')

Oppmøte for prosjektgruppen	
Faste møtetidspunkter	10.00-14.00, tirsdag til torsdag
"Åpne tider" (egenarbeid, fleksibel planlegging)	• 08.00-10.00, 14.00-16.00 • Hele fredag

Prosjektgruppens møter med oppdragsgiver og veileder	
Veileder	12.30 hver torsdag. Kan unntas ved varsel
Oppdragsgiver	Omtrent månedlig etter avtale (se også 'Prosjektorganisering')
Innlevering av statusrapport	14. Feb, 14. Mar, 18. Apr

Prosjektorganisering	
Prosjektrammeverk	Inspirert av scrum
Interne møter	● ***Ukentlige planleggingsmøter:*** ○ Oppsummering av forrige uke ○ Prioritering av oppgaver ● ***Daglig møte***: ○ informere hva som skal gjøres den dagen) ● ***Omtrent månedlig statusmøte*** med oppdragsgiver ● ***Retrospec-møte*** etter møte med oppdragsgiver
Organisering av oppgaver	● Planlegges på ukentlige planleggingsmøter ● **Trello** for organisering ● Ukentlige oppgaver plasseres i “To do”-listen i Trello
Verktøy for ukentlig tidsplanlegging	Google Calendar
Verktøy/modell for overordnet tidplanlegging av prosjekt	Gantt-skjema med Office Project
Fil- og dokumentdeling	Google Drive/Google Docs
Verktøy for rapportskriving	ShareLaTeX

Tillegg B

Grupperegler

Avtale om grupperegler

1. **Oppmøte og forventet innsats**
    a. Det er forventet at hvert gruppemedlem deltar på de faste møtetidspunktene bestemt i dokument *Formelle bestemmelser*, så fremt deltaker melder fra om fravær eller gruppa avtaler avvik fra de faste møtetidspunktene.
    b. Fravær for det enkelte medlem skal varsles og begrunnes
    c. Utenfor faste møtetidspunkt forventes det at hver enkelt gjør lovet arbeid og/eller lesing av teori
2. **Upassende atferd og avskjed**
    a. Et medlem kan avskjediges fra gruppen ved brudd på denne avtalen som følge av ikke varslet fravær, eller det majoriteten i gruppa i et tilfelle mener er upassende atferd fra ett av medlemmene. Avskjed skjer altså skriftlig med bevitnelse og signatur fra minst to av gruppas medlemmer.
    b. Om et gruppemedlem oppleves som å ha upassende atferd eller har mye ikke varslet fravær, plikter de to andre medlemmene i gruppa å varsle og diskutere forholdet med vedkommende, og komme til enighet om forventet opptreden. Mottar et medlem flere enn tre varsler, avgjøres avskjed.
    c. Ved avskjed må veileder og oppdragsgiver varsles
3. **Uenigheter**
    a. Faglige uenigheter skal avgjøres demokratisk. Om man ikke kommer til enighet, må veileder eller oppdragsgiver konsulteres
    b. Sterke uenigheter skal megles medlemmene i mellom, og avstemmes demokratisk. Om intern megling av uenighetsforholdet ikke erkjennes, vil det tas opp med prosjektets veileder
4. **Kostnader**
    a. Alle eventuelle kostnader deles likt mellom de tre gruppemedlemmene
5. **Signatur**
    a. Hvert medlem plikter å signere denne avtalen før arbeidet iverksettes
6. **Annet**
    a. Audun Huseby er prosjektets leder

Audun Huseby

Audun Huseby

Stein Ove Jernberg

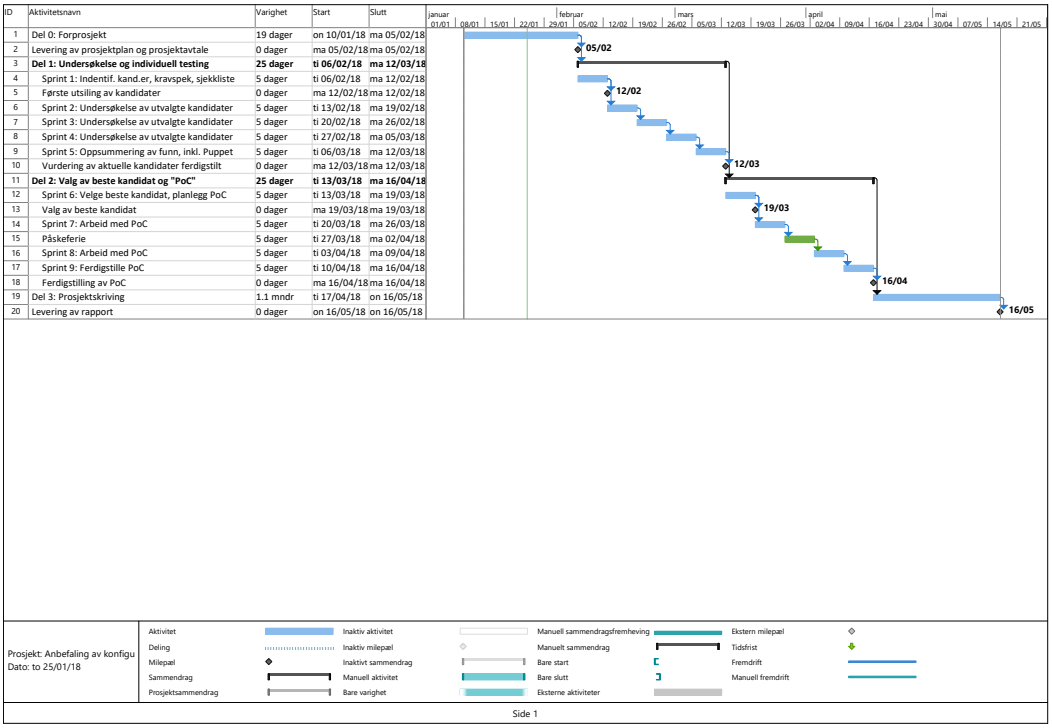
Stein Ove Jernberg

Andreas Osborg

Andreas Osborg

Tillegg C

Gantt-diagram



Figur C.1: Gantt-skjema

Tillegg D

Risikoanalyse

Risikoanalyse av prosjektet

For å sikre at prosjektet kommer i mål har vi valgt å foreta en risikoanalyse. Denne har gått ut å identifisere ulike risikofaktorer innenfor tre ulike kategorier: Teknologisk risiko, forretningsrisiko og prosjektgruppemessig risiko. Deretter har disse blitt vurdert med utgangspunkt i risikomatrisen under. For høy risiko medfører iverksettelse av tiltak for å begrense risiko.

Risikomatrise

Hver risikofaktor er blitt vurdert med tanke på hvor hvor sannsynlige de er og hvor stor konsekvens de vil ha for prosjektet. Produktet av sannsynlighet x konsekvens utgjør den totale risikoverdien. Risikoverdi (R) = Sannsynlighet (S) * Konsekvens (K).

		Konsekvens (K)				
Sannsynlighet (S)		**Ubetydelig**	**Mindre alvorlig**	**Betydelig**	**Alvorlig**	**Svært alvorlig**
	Usannsynlig	1	2	3	4	5
	Lite sannsynlig	2	4	6	8	10
	Mindre sannsynlig	3	6	9	12	15
	Sannsynlig	4	8	12	16	20
	Svært sannsynlig	5	10	15	20	25

Vurdering av risikoverdi med tiltak

Beregnet risikoverdi for hver risikofaktor vil vurderes ut fra tabellen under. For risikofaktorer som havner i grønn sone regnes risikoen som så lav at det ikke er nødvendig med tiltak. Risiko i gul, oransje og rød sone innebærer innføring av tiltak som begrenser risikoen.

1-4	Lav risiko. Ingen tiltak er nødvendig.
5-9	Medium risiko. Enkelte tiltak bør gjennomføres.
10-16	Høy risiko. Grundige tiltak bør gjennomføres.
20-25	Svært høy risiko. Flere større tiltak er påkrevd.

	Risikofaktor:	Risikoverdi (S*K):	Tiltak:	Etter tiltak:
1	*Teknologisk risiko*			
1.1	Endringer i forutsetninger for valgt CM-system. Eks. Systemet avvikles, slutt på utvikling/support	S1*K3 =3		=3
1.2	Manglende nøkkelfunksjonalitet i valgt CM-system. for dårlig vurderings-/ utvelgelsesprosess. Oppdages først under oppsett av “Proof of consept”	S2*K4 =8	God og ofte kommunisere med oppdragsgiver og veileder, så evt. Misforståelser kan tas tidlig	S1*K4 =4
1.3	Tap av data, lokalt eller i remote repo.	S2*K5 =10	Ta backup regelmessig.	S2*K2 =4
1.4	Kompromitert repo (Korruptering/sletting av filer)	S2*K2 =4		=4
1.5	Tap av nøkler. (uvedkommende får tilgang til proof of consept-oppsett)	S2*K2 =4		=4
1.6	Problemer med SkyHigh(Openstack)	S3*K3 =9	Få ressurser fra NTNU-IT Trondheim og være forberedt på å evt. måtte bruke AWS.	S3*K1 =3
2	*Forretningsmessig risiko*			
2.1	Oppdragsgiver trekker seg fra prosjektet/bryter kontrakten	S1*K4 =4		=4
2.2	Sluttproduktet oppfyller ikke oppdragsgivers kriterier	S3*K4 =12	Holde god kontakt og kommunikasjon med oppdragsgiver.	S2*K4 =8
3	Gruppemessig risiko			
3.1	Sykdom, lav alvorlighetsgrad (type forkjølelse, influensa…)	S3*K3 =9	Holde kommunikasjon med den syke under sykdomsperioden. Den syke kan bidra etter evne.	S3*K2 =6
3.2	Manglende motivasjon	S2*K1 =2		=2
3.3	Gruppemedlem trekker seg/gruppemedlem sparkes	S1*K4 =4		=4
3.4	Gruppemedlem motarbeider prosjekt-framgang / sabotasje	S1*K5 =5	Alle medlemmer har i kontrakten forpliktet seg til å bidra til prosjektet. Gruppen kan prøve å løse konflikten internt. Evt kan veileder hjelpe til med å løse eventuelle konflikter.	S1*K3 =3