NTNU
Norwegian University of
Science and Technology

# TICK-stack

Forfattere
Adrian L Lange
Vetle T Moen

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

20.05.2019

Veileder Eigil Obrestad

# Sammendrag av Bacheloroppgaven

| | |
|---|---|
| Tittel: | **TICK-stack** |
| Dato: | 20.05.2019 |
| Deltakere: | Adrian L Lange<br>Vetle T Moen |
| Veiledere: | Eigil Obrestad |
| Oppdragsgiver: | Basefarm AS |
| Kontaktperson: | Patrik Storm Olsen |
| Nøkkelord: | Monitorering, ytelsestesting, sikkerhetsvurdering, TICK, Puppet, Docker, Linux |
| Antall sider: | 52 |
| Antall vedlegg: | 8 |
| Tilgjengelighet: | Åpen |

Sammendrag:

Basefarm AS vil bruke monitoreringssystemet TICK i høyere grad enn det de allerede gjør, og ønsker derfor en vurdering av TICK-stakken i sin helhet. Prosjektgruppen har fått i oppgave å utrede ytelse og sikkerhet ved implementasjon av TICK i to utgaver. Den ene utgaven er monolittisk, der hele stakken kjører på én enkelt maskin. Den andre utgaven er mikrotjeneste-basert hvor TICK kjører distribuert over flere maskiner ved bruk av Docker og Docker Swarm. Etter å ha implementert begge utgavene vurderes de hver for seg med tanke på sikring av implementasjonen og ytelsen, og deretter sammenlignes. Prosjektgruppen fant at begge implementasjonene yter godt og kan sikres på en god måte, begge med hver sine fordeler og ulemper. Begge implementasjonene er anbefalt, men hvilken man vil ta i bruk bør bedømmes etter behov og eksisterende infrastruktur.

# Summary of Graduate Project

| | |
|---|---|
| Title: | **TICK-stack** |
| Date: | 20.05.2019 |
| Authors: | Adrian L Lange<br>Vetle T Moen |
| Supervisor: | Eigil Obrestad |
| Employer: | Basefarm AS |
| Contact Person: | Patrik Storm Olsen |
| Keywords: | Monitoring, performance testing, security evaluation, TICK, Puppet, Docker, Linux |
| Pages: | 52 |
| Attachments: | 8 |
| Availability: | Open |

Abstract: Basefarm AS wants to use the monitoring system TICK to a greater extent than what they currently do, and therefore wants an assessment of TICK in its entirety. The project group has been given the task to assess the performance and security aspects of TICK by implementing it in two different ways. A monolithic implementation where TICK runs on a single machine, and a distributed implementation where TICK runs across several machines using Docker and Docker Swarm. Once implemented, both implementations are individually evaluated and then compared. The project group found that both implementations perform well and can be sufficiently secured, each with their own advantages and drawbacks. Both implementations are recommended, although which of the two should be used depends on what is necessary and pre-existing infrastructure.

# Innhold

# Figurer

# Tabeller

# Listings

# Terminologi

**API**

"Application Programming Interface", et sett med definisjoner og protokoller brukt ved utvikling av programvare for å tillate kommunikasjon mellom komponenter.

**Black box**

Betegnelse for at man ikke har innsyn inn til noe, typisk brukt i sammenheng med interne system hos en bedrift eller programvare uten åpen kildekode.

**CentOS**

"Community Enterprise Operating System", en kommersiell og gratis GNU/Linux distribusjon med funksjonell kompatibilitet med RHEL.

**DNS**

"Domain Name System", navntjenerstandard som kobler domenenavn til IP-adresser.

**Docker Swarm**

Orkestreringsverktøy for konteinere i større skala, slik som i en klynge over flere maskiner.

**Docker**

Verktøy for kontrollering og bruk av konteinere.

**ERB**

"Embedded Ruby", et mal-språk brukt i Puppet.

**Facter**

Komponent i Puppet som håndterer fakta om noder omhandlet i Puppet konfigurasjon.

**FOSS**

"Free Open-Source Software", brukt for å beskrive programvare som kommer uten noen praktiske begrensninger i bruk, arv og omfordeling.

**FQDN**

"Fully Qualified Domain Name", et fullverdig domenenavn, brukt for å identifisere maskiner og tjenster i en infrastruktur eller et domene. Eksempelvis `maskin1.eksempel.no`.

**GNU**

Kort for "GNU Project", et enormt samarbeid for å utvikle FOSS programvare til Unix-baserte operativsystem.

**Hiera**

"Hierarchical Key-Value Lookup-system", data-definerende komponent brukt i Puppet.

**Kernel**

Kjernen i et operativsystem, som under GNU/Linux-baserte operativsystem/distribusjoner er Linux. Denne delen av operativsystemet er ansvarlig for oppstart, oversetting av maskinkode, samt maskinvarekontroll og kommunikasjon.

**OpenStack**

En skyløsning for det kommersielle og private markedet, hvor brukere kan rulle ut og kontrollere virtuelle maskiner, nettverk og lagring.

**PKI**

"Public Key Infrastructure", et rammeverk for sikker tilknytning av nøkler og andre identiteter med en tiltrodd tredjepart.

**PuppetDB**

Database-komponent i Puppet som håndterer data Puppet vet om og håndterer, slik som fakta om noder.

**Repo**

Forkortelse av det engelske ordet "repository", mest brukt i sammenheng med versjonskontroll og Git, hvor et repo er oppbevaringsstedet eller rotkilden til et prosjekt.

**RHEL**

"Red Hat Enterprise Linux", en Linux-distribusjon rettet mot det kommersielle markedet.

**RPM**

"Red Hat Package Manager", pakkebehandlingsverktøy utviklet at Red Hat for deres operativsystem.

**SkyHigh**

NTNUs private skyløsning for ansatte og studenter, en implementasjon av OpenStack.

**SQL**

"Structured Query Language", et domenespesifikt spørringsspråk brukt i sammenheng med relasjonelle databaser.

**SSH**

"Secure Shell", programvare og nettverksprotokoll brukt til å få fjerntilgang til en annen maskin.

**Stack**

En samling ressurser (programvare og tjenester), nettverk og lagring, typisk virtualisert.

**TICK**

Telegraf, InfluxDB, Chronograf og Kapacitor, en samling av programmer som danner et monitoreringssystem, utviklet av InfluxData.

**TLS**

"Transport Layer Security", en protokoll for å forsikre integritet ved kommunikasjon mellom to parter.

**TOML**

"Tom's Obvious, Minimal Language", et konfigurasjonsfil-språk utviklet for å være lett forståelig.

**TSDB**

"Time Series Database(s)", databaseløsning optimalisert til å håndtere tidsseriedata, slik som endringer i ressursbruk i én eller flere maskiner over lengre tid.

**YAML**

"YAML Ain't Markup Language", et lett forståelig data-definerende språk.

# 1 Innledning

## 1.1 Bakgrunn

Det er ikke en tjeneste hvis den ikke overvåkes. Hvis det ikke fins overvåkning så kjører du bare programvare, du drifter ikke en tjeneste [1]. Overvåkning av tjenester, også kjent som monitorering, er en vital del ved drift av moderne digitale infrastrukturer. Et monitoreringssystem gir god innsikt i hvordan systemer, nettverk og tjenester oppfører seg og hvilke ressurser de bruker. Informasjon fra et monitoreringssystem gir operatører et godt overblikk over tjenester de leverer, og gir dem mulighet til å se hvordan tjenestene har oppført seg i fortiden.

Basefarm AS er en driftsleverandør av virksomhetskritiske IT-løsninger. Med kontorer i flere europeiske land leverer de drift og håndering av private og offentlige skytjenester fra egne datasentre i Norge, i tillegg leverer de rene datasentertjenester etter behov [2]. Eksempelvis drifter Basefarm servere og skytjenester for Norwegian, Flytoget og Lånekassen, hvor de har dedikerte team til enkelte kunder.

I løpet av 2018 gikk Basefarm over til å ta i bruk en monitoreringsløsning utviklet av InfluxData[1] kalt *TICK*[2] - en samling av fire programmer som til sammen danner et komplett monitoreringssystem. TICK er utviklet spesifikt til å håndtere store mengder informasjon og har mange utvidelsesmuligheter. Basefarm bruker TICK hovedsaklig til visualisering av infrastruktur de er ansvarlige for, samt til integrering i deres alarmsystem. Dette lar de ha full kontroll over tjenester de leverer, og blir varslet automatisk om eventuelle feil og avvik i systemene de er ansvarlige for.

## 1.2 Oppgaven

### 1.2.1 Problemområde

Målet med monitorering av tjenester er å tilgjengeliggjøre informasjon om en tjeneste til de som er ansvarlig for å drifte den [3]. Denne informasjonen høstes fra grunnleggende statistikk hos en maskin, både virtuell og fysisk, samt programvare som lever på disse maskinene [4]. Denne informasjonen loggføres i høyt volum, og gir operatører muligheten til å visualisere og analysere informasjonen etter behov. Videre vil informasjonen kunne brukes til å se dannede trender i tjenester som driftes, og lar operatører justere ressurser etter estimerte fremtidige behov.

Å loggføre er kun ett aspekt ved monitorering, men man kan bruke informasjonen til noe litt mer nyttig slik som automatisering og varsling. Ved forhåndsbestemte hendelser kan man fortelle systemet hva som skal gjøres [5]. Dette kan være automatisk varsling til de som er ansvarlige for å drifte tjenesten slik at de kan rette opp feil og mangler. I tillegg kan man sette opp automatisk skalering av tjenesten basert på trender, slik at operatører ikke trenger å justere dette manuelt.

[1] https://www.influxdata.com/
[2] https://www.influxdata.com/time-series-platform/

Uten monitorering drifter man ikke en tjeneste fullt ut, og man vet aldri med sikkerhet om en tjeneste opererer korrekt uten å måle den. Uten noen form for varsling eller automatisering kan en tjeneste slutte å fungere uten at driftansvarlige vet om det, og dette etterlater ansvaret for rapportering til sluttbrukeren av tjenesten. Dette er uakseptabelt i moderne IT-drift [1].

### 1.2.2 Oppgavebeskrivelse

Basefarm benytter for tiden TICK til monitorering av deres infrastruktur hvor de har rullet ut løsningen separert inn i hver sin virtuelle maskin, uten noen form for replikering eller skalering. De ønsker nå å benytte TICK i større grad, til alt fra distribuerte systemer og mikrotjenester, til virtuell og fysisk maskinvare i deres infrastruktur. De har i denne sammenheng meldt interesse for dette som en bacheloroppgaver ved NTNU i Gjøvik.

Prosjektgruppen skal på vegne av oppdragsgiver implementere, ytelsesteste og sikkerhetsevaluere TICK-stacken i to utgaver. En monolittisk utgave hvor alle komponentene kjører på én fysisk eller virtualisert maskin, og en distribuert utgave hvor komponentene er etablert som mikrotjeneste i konteinere.

Prosjektgruppen skal høste inn og evaluere eksisterende erfaringer innen temaet og produktet, og utarbeide egne meninger for løsningene.

Begge utgavene skal sikkerhetsevalueres og ytelsestestet opp mot hverandre, for å finne eventuelle likheter og forskjeller i utgavene, og rede for hvilken som egner seg best til Basefarm sitt bruk. Dette er viktig da Basefarm drifter infrastruktur for kunder med behov for høy ytelse og sikkerhet. Oppgaven kan oppsummeres slik:

1. Det skal implementeres to utgaver av TICK-stacken.
    1. En monolittisk utgave, konfigurasjonsstyrt med Puppet.
    2. En mikrotjenesteutgave, helt eller delvis konfigurasjonsstyrt med Puppet.
2. Det skal utføres en sikkerhetsevaluering av implementasjonene.
    1. Sikring av systemet programvaren kjører på.
    2. Sikring av tjenestene.
    3. Sikring av kommunikasjon mellom tjenester.
3. Det skal utføres en ytelsestest som muliggjør en sammenligning mellom utgavene.
4. Det skal samles inn og sammenstille meninger og erfaringer om TICK som et komplett produkt.

### 1.2.3 Rammer og avgrensninger

Prosjektet skal forholde seg til programvare representert i TICK-stacken, derav Telegraf, InfluxDB, Chronograf og Kapacitor. Det skal benyttes Puppet til konfigurasjonsstyring, og det skal benyttes GNU/Linux-baserte operativsystemer. Dette er relevant for oppdragsgiver da de allerede benytter Puppet og har forståelse for systemet, samt at de primært bruker RHEL som operativsystem på deres systemer. All implementasjon skal foregå på skyløsningen tilgjengelig til prosjektgruppen ved NTNU i Gjøvik - SkyHigh.

Det skal ikke implementeres eller tilrettelegges noe for Windows-baserte installasjoner, da det ikke er relevant for oppdragsgiver. Det er heller ikke nødvendig å se på alternative løsninger eller utbedringer til TICK-stacken.

## 1.3 Målgruppe

Målgruppen til denne rapporten og eksempeloppsettet er primært oppdragsgiver. Prosjektgruppen håper at rapporten, eksempeloppsettet samt den utviklede Puppet-modulen vil være nyttige for andre aktører som er interessert i å anvende TICK-stacken til monitorering.

## 1.4 Gruppemedlemmene

Begge prosjektgruppemedlemmene er på siste semester i bachelorstudiet *IT-drift og informasjonssikkerhet* ved NTNU i Gjøvik. Gruppemedlemmene har størst interesse for IT-drift, med varierende interesseområder slik som programmerbar infrastruktur, programmering, system og nettverksadministrasjon, og automasjon.

## 1.5 Roller

Oppdragsgiver er Basefarm AS, representert av Patrik Storm Olsen og Audun Huseby. Veileder for prosjektgruppen er Eigil Obrestad, ansatt ved IIK/IE-fakultetet ved NTNU i Gjøvik. Gruppeleder for prosjektet er Adrian Lund-Lange, men med kun to gruppemedlemmer har denne rollen liten betydning.

## 1.6 Rapportstruktur

Rapportens videre innhold er inndelt i følgende kapitler:

- Kapittel 1, Innledning introduserer rapportens innhold.
- Kapittel 2, Teori dekker all grunnleggende teori for resten av rapporten.
- Kapittel 3, Design beskriver hvordan teknologier blir tatt i bruk. Videre beskrives de generelle designvalgene tatt under planlegging av den monolittiske og den distribuerte utgaven av TICK.
- Kapittel 4, Utvikling av modul gir et innblikk i hvordan Puppet-modulen ble utviklet.
- Kapittel 5, Implementasjon beskriver hvordan utgavene av TICK ble implementert.
- Kapittel 6, Sikkerhet beskriver sikkerhetsevalueringen av komponentene i stacken, samt funn fra denne prosessen.
- Kapittel 7, Ytelse beskriver hvordan ytelsestesting blir utført, samt resultater fra dette.
- Kapittel 8, Konklusjon viser og diskuterer resultater, og avslutter rapporten.

# 2 Teori

Prosjektet omhandler i stor grad programmerbar infrastruktur, så det er viktig å beskrive teorien bak konsepter, komponenter og strategier som prosjektet baseres på. Dette kapitlet diskuterer viktige temaer relevant for oppgaven, slik som monitorering, tidsseriedata, orkestrering av maskinvare, provisjonering av tjenester, og relevante støtteteknologier.

## 2.1 Monitorering

Monitorering handler om at de som drifter et system skal ha muligheten til å være allseende og allvitende [1]. Dette er i praksis umulig uten et system som kan samle inn, lagre, analysere, og visualisere metrikkmålinger. Den type data som samles inn er *metrikk*, definert som et punkt med data om en spesifikk ressurs eller type forespørsel, og en *metrikkmåling* er en metrikk med et tidsstempel [6]. Et monitoreringssystem kan deles opp i fire punkter [7]:

**Innsamling** er prosessen der man samler metrikk om kjørende tjenester (f.eks. en nettjener) og undersystemer (bl.a. prosessorbruk, minnebruk), og deretter lagrer unna denne informasjonen som en metrikkmåling. Dette gjøres typisk med en agent som kjører på alle systemer som skal monitoreres.

**Lagring** gjøres typisk sett i en database. Denne bør være optimalisert for lagring og behandling av metrikkmålinger.

**Analyse** handler om å se etter trender og sammenhenger i et stort system. Dermed kan man handle proaktivt når en ser at for eksempel innkommende trafikk til en nettjener trender oppover. Dette kan utføres av samme system som lagrer metrikkmålingene, men for økt ytelse kan det være fornuftig å benytte en selvstendig tjeneste til dette.

**Visualisering** er et nyttig verktøy for å se hvordan et helhetlig system oppfører seg i sanntid samt en kort periode tilbake i tid. Man kan se på et slikt verktøy som en informasjonsradiator, som gir fra seg mye informasjon som er lett å ta inn.

Monitorering gjør det mulig å jobbe proaktivt da en kan overvåke trender i et system, og dermed skalere opp en gitt tjeneste før den bryter sammen under høy belastning, eller skalere ned ved lav belastning for å spare ressurser og redusere kostnad.

## 2.2 Tidsseriedata

For at behandling av metrikkmålinger skal være så effektivt som mulig har man i nyere tid benyttet et format som heter *tidsserie* [8]. Dette er et format hvor identifikasjonen til en metrikkmåling er tidsstempelet. Tidsstempelet er tidspunktet metrikk ble innhøstet av innsamlingsagenten og sendt til lagringstjenesten. Dette gjør det mulig å hente ut metrikkmålinger i et visst tidsintervall på en svært effektiv måte, og gjør det dermed lettere å analysere hendelser.

Tidsseriedata må lagres et sted som er optimalisert til å oppbevare og søke gjennom historikk. Tradisjonelle relasjonelle databaser håndterer denne oppgaven på en tilfreds-

stillende måte, men etter hvert som antall agenter øker så vil behovet for effektivitet gå opp [9]. *Time Series Database* (TSDB) er en egen kategori av databaser som har oppstått de siste årene etter behovet for å lagre store mengder med informasjon som denne typen metrikk genererer, og er optimalisert til å aggregere denne informasjonen [10][11]. TICK-stacken inneholder en tidsseriedatabase kalt InfluxDB, og er den primære årsaken til at TICK ble utviklet.

En TSDB har gjerne støtte for grunnleggende matematiske metoder direkte i spørringer, samt filtrering basert på mønstre. Enkelte implementasjoner har òg statistiske funksjoner som hjelper med å håndtere og filtrere data. I InfluxDB sitt tilfelle så er spørringsspråket InfluxQL (Influx Query Language). InfluxQL er svært likt tradisjonell SQL, men har støtte for sammenfletting av resultater hvis man skal finne produktet av to målinger.

```
select cpu_usage * requests
```

Listing 2.1: Produkt av CPU-bruk og HTTP-forespørsler.

I spørringen fra eksempelkode 2.1 flettes de to tidsseriene 'cpu_usage' og 'requests' basert på overlappende tidsområder i begge og svarer med én enkelt sammensatt tidsserie. Med dette kan man se hvor trafikk relaterer til ressursbruk. Dette er delvis likt JOIN i tradisjonelle databaser, men alle datapunkter blir flettet sammen på det de to resultatene har til felles: tid.

TSDB-er lar ofte brukere ta i bruk filtre som fungerer mot mønstre i tidsserier, slik at man enkelt kan hente ut typisk informasjon uten mye arbeid.

```
select onpeak(cpu_usage)
```

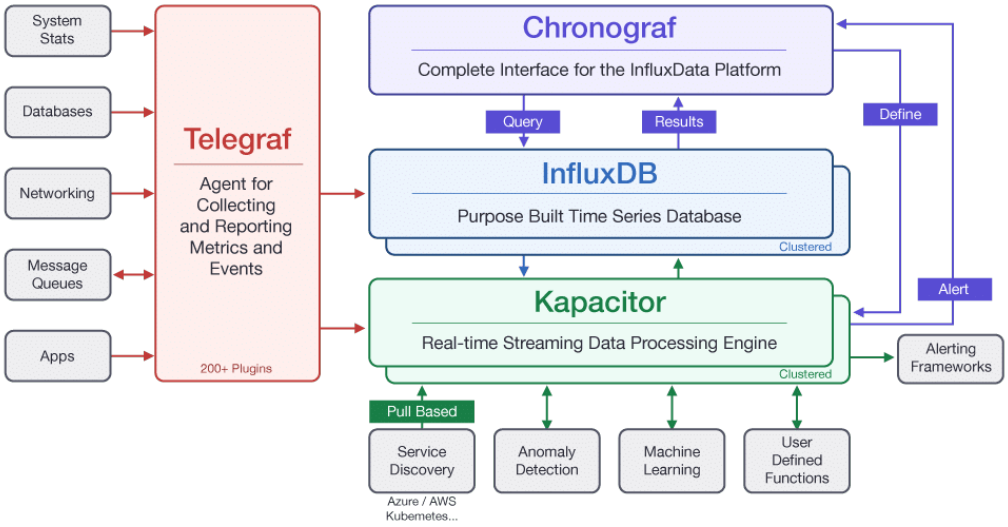
Listing 2.2: Høyeste verdi i en tidsserie.

I spørringen fra eksempelkoden 2.2 vil spørringen svare med tidspunktet hvor prosessorbruk nådde topp. Simplifiseringen av spørringer er det som appellerer for TSDB-er når man må håndtere denne type data.

Det finnes mange gode tidsseriedatabaser, men hovedfokuset i dette prosjektet er InfluxDB, en av komponentene i TICK. TICK har i senere tid blitt svært populært grunnet InfluxDBs popularitet og stackens modulære oppbygging [12]. Hver komponent er utvidbar med at man kan overvåke mange populære tjenester med offisiell støtte. Visualiserings- og varslingskomponentene kan erstattes med andre, noe som gjør at TICK-stacken kan utvides og tilpasses den enkelte bedrifts behov.

## 2.3 TICK

InfluxData, et selskap basert i San Fransisco, har utviklet en komplett løsning for monitorering og tidsseriedata. Denne heter TICK[1], og er en samling av fire komponenter - Telegraf, InfluxDB, Chronograf og Kapacitor.




Figur 1: Overblikk over arkitekturen i TICK, slik InfluxData definerer den.

Alle komponentene i TICK er åpen kildekode, uten noen form for kostnad eller lisensiering. TICK er ett av tre produkter InfluxData leverer, hvor de to andre baserer seg på TICK i bunn. De to andre produktene de leverer er *InfluxCloud* og *InfluxEnterprise*.

*InfluxCloud*[2] er en løsning hvor InfluxData tilbyr å levere TICK som en ferdig løsning som de vil drifte for deg. Du trenger bare å koble til agenter som samler metrikken og sende dette til resten av systemet hos de, og alt skal fungere. Denne tjenesten kjøres eksklusivt på Amazon sin skyløsning *AWS*[3]. De tilbyr mulighet til sikkerhetskopiering, skalering, og redundans i alle komponentene rett ut av boksen, noe man ikke har mulighet til med TICK alene.

*InfluxEnterprise*[4] er identisk til InfluxCloud, hvor den eneste forskjellen er at man kan kjøre det hvor enn man vil, om dette skulle være en annen tjenestetilbyder enn AWS eller på egne servere håndtert av bedriften. InfluxData vil i dette tilfellet ikke drifte løsningen.

Disse to løsningene er, i motsetning til TICK alene, betalte løsninger.

[1] https://www.influxdata.com/time-series-platform/
[2] https://cloud.influxdata.com/
[3] https://aws.amazon.com/
[4] https://www.influxdata.com/influxenterprise/

### 2.3.1 Komponenter i TICK

Komponentene har hver sin spesifikke oppgave og er løst koblet sammen slik som vist i figur 1. All kommunikasjon foregår via HTTP API-kall, som gjør at hver komponent kan implementeres uavhengig av hverandre. Hele TICK stacken er utviklet i Go[5], som gjør det lett å kompilere til flere systemarkitekturer og gir TICK høy portabilitet.

#### Telegraf

Telegraf er en datasamlende agent som man installerer og kjører på hver maskin man vil monitorere [13]. For hver man maskin må man definere et sett med kilder (inputs) og destinasjoner (outputs). Kilder kan være informasjon om systemets belastning, eller spesifikk informasjon om en programvare som kjører på systemet, slik som en lastbalanserer. Destinasjoner er typisk databaser og aggregeringssystemer, slik som InfluxDB og Kapacitor, hvor man lagrer og behandler informasjonen som blir høstet inn. Man kan definere flere kilder og destinasjoner i samme konfigurasjon.

Telegraf er integreringsbasert, som eksempelvis vil si at hvis man vil overvåke et sett med konteinere i Docker så er det støtte for dette. Den har og støtte for utvidelser, så om den ikke har støtte for nettopp det du vil monitorere ut av boksen så kan dette utvikles etter behov [14].

#### InfluxDB

I hjertet av løsningen sitter InfluxDB [15], en database designet fra bunn med tidsserier i fokus. InfluxDB viser høy ytelse for input, komprimering og sanntidsspørring for all data [12]. Informasjon sendes til InfluxDB fra agenter slik som Telegraf, og deretter leses via spørringer fra kommandolinje, innebygd HTTP API, eller ved spesialbygde verktøy slik som som Chronograf og Kapacitor.

InfluxDB kan håndtere millioner av datapunkter i sekundet, så med denne mengden innkommende data som skal lagres for å etablere historikk er lagringsbehov en bekymring. InfluxDB komprimerer automatisk all innkommende data, og gir deg mulighet til å justere detaljnivået for eldrende data automatisk for å spare plass [16].

Spørringer skjer med et egetdefinert spørringsspråk kalt "InfluxQL". Med basis i tradisjonelle spørringsspråk slik som SQL kan man kan lett kjenne seg igjen om man kommer fra andre løsninger. Man har også muligheten til å benytte et mer moderne spørringsspråk kalt Flux[6]. Flux er designet spesifikt for InfluxDB og tidsseriedata, og er fortsatt under utvikling.

```
SELECT mean("total_usage") FROM "cpu" GROUP BY time(1h)
```

Listing 2.3: Gjennomsnitt av total prosessorbruk for hver time.

I eksempelkoden 2.3 hentes gjennomsnittsnivået av total bruk av en prosessor den siste timen. Hvis man er kjent med SQL fra før vil man legge merke til hvor like disse to spørringsspråkene er. Fordelen med dette er at man bruker kortere tid på å lære seg InfluxQL.

I Enterprise-versjonen av InfluxDB har man mulighet til å skalere, det vil si at man har flere instanser av InfluxDB kjørende, som kommuniserer og synkroniserer mellom seg for å danne det komplette bildet av databasen. Dette øker ytelsen til databasen ved at man

[5] https://golang.org/project/
[6] https://docs.influxdata.com/flux/

kan ha flere destinasjoner, samt at det øker robustheten ved at man har redundans.

### Chronograf

Chronograf er den visuelle delen av TICK, en nettsidebasert løsning som lar deg visualisere spørringer og metrikk fra databasen, med høyt fokus på å kunne tilpasse hva som visualiseres etter behov [17]. Med egendefinerte dashbord kan man lett få overblikk i hvordan tjenester og maskinvare oppfører seg, alt etter hva man behov for å se. Chronograf brukes også til å konfigurere InfluxDB og Kapacitor, hvor man etablerer databaser og brukere, samt definerer utløserpunkt (triggere) for Kapacitor.

Chronograf har muligheter for "Single-Sign-On"-innlogging ved hjelp av eksisterende løsninger tilbudt av GitHub, Google, Auth0 og flere [18]. Rollebasert innlogging lar man konfigurere hvem som kan se hva, og hvilke rettigheter hver enkelt bruker eller gruppe har.


Figur 2: Eksempel på et dashboard i Chronograf.

### Kapacitor

Kapacitor har to konkrete egenskaper: filtrering av data i databasen, samt varsling om metrikk er utenfor definerte grenser [19]. I Kapacitor kan man sette opp triggere etter behov, som kan gjøre alt fra varsling over epost til å kunne skalere hele stacken automatisk basert på mengde trafikk. Man definerer avanserte dataprosesserignsalgoritmer i et språk kalt TICKscript [20].

Det er og mulighet til å bruke Kapacitor som et filtreringssystem til InfluxDB for å kontrollere downsampling og arkivering av foreldet metrikk.

Kapacitor har i likhet med Telegraf støtte for utvidelser. I denne sammenheng kan

man koble til anomalitetsdeteksjonsystemer, hvor Kapacitor kan dynamisk lære hva som er avvikende eller støyende metrikk. Man kan bruke predefinerte regler i sikkerhetssammenheng, slik at man kan bli varslet om potensielle problemer.

Enterprise-versjonen av Kapacitor har og mulighet for skalering på lik linje med InfluxDB.

### 2.3.2 Sikkerhet i TICK

All kommunikasjon mellom komponentene kan, og bør sikres med TLS[7] og InfluxDB har støtte for å legge til brukere med passord og sette opp autentiseringsnivå per bruker. TLS baserer seg på sertifikater for å sjekke autentisiteten til avsender, kryptere trafikk og verifisere integriteten til innholdet.

Sikkerhet ved lagring av data er ikke støttet direkte i noen av komponentene, så om dette er et behov må det skje på operativsystem- eller maskinvarenivå. Man må selv vurdere sensitiviteten til lagrede data om dette er nødvendig, da det har mulighet til innvirkning på ytelse.

### 2.3.3 TICK i produksjon

Eksempelvis av eksisterende erfaringer av TICK i produksjon finner man et fåtall, da dokumentasjonen for hele stacket og enkeltkomponentene er veldig god, og har mange eksempler tilgjengelig. Et annet funn er at de fleste som vurderer eller bruker TICK bruker ikke TICK i sin helhet, men heller bytter ut komponenter etter behov. Mest typisk er å bytte ut Chronograf med andre grensesnitt slik som Grafana[8].

Rudi Martinsen, senior sky-arkitekt hos Intility[9] har skrevet innlegg om InfluxDB, Kapacitor og TICK i sin helhet, og roser hele stacken for sin simplisitet og effektivitet [21][22][23]. Docker-kaptein[10] Gianluca Arbezzano argumenterer for at en spisset TSDB som InfluxDB blir stadig mer nødvendig nå som mikrotjenester har blitt mer utbredt [24]. Arbezzano holder samme formening som Martinsen og roser TICK for simplisitet og effektivitet. Begge to er fornøyde med at TICK er en svært modulær stack hvor man kan ta i bruk de komponentene man selv ønsker.

## 2.4 Støtteteknologier

For å kunne gjennomføre dette prosjektet er det best å basere arbeidet på støttende teknologier. For å kunne installere TICK behøves et sett med virtuelle tjenere som kan kjøre infrastrukturen, samt verktøy for automatisering av oppsett. Dette åpner opp for at infrastrukturen kan opprettes, endres, utvides, reduseres, eller slettes etter behov.

### 2.4.1 Virtualisering

TICK vil kjøre på virtualisert maskinvare, da det er langt enklere å forholde seg til både implementasjonsmessig og økonomisk enn fysisk maskinvare. Oppdragsgiver har egen løsning for dette som hadde vært mulig å bruke til prosjektet, men de så helst at prosjektgruppen brukte andre løsninger. Heldigvis har NTNU i Gjøvik en egen skytjeneste tilgjengelig for studentene til nettopp slike formål, så alt arbeid foregår der. Skytjenesten

[7] https://en.wikipedia.org/wiki/Transport_Layer_Security
[8] https://grafana.com/
[9] https://www.intility.no
[10] Docker Captain er en distinksjon gitt til ekspertmedlemmer av Docker-samfunnet med lidenskap om å dele kunnskap.

er i bunn en OpenStack-implementasjon[11], hvor orkestrering- og prosvisjoneringsverktøy som allerede finnes for løsningen vil bli tatt i bruk.

### 2.4.2 Mikrotjenester

I motsetning til tradisjonell skalering med flere virtuelle maskiner har det i nyere tid blitt mer populært med konteinerteknologi. I stedet for at hver "maskin" på forhånd definerer minne, prosessorkraft, og lagringskapasitet har konteinere felles kernel med maskinen de kjører på, og bruker kun de ressursene som kreves mens konteineren kjøres [25]. Dette gir mulighet til å orkestrere i større skala uten å være begrenset til høyere krav til ressurser slik tradisjonell virtualisering har, og krever mindre arbeid for å sette opp.

Det er ikke satt krav til hvilke teknologier som skal brukes til dette, så valget gikk til Docker som er mest brukt og gruppemedlemmene har kjennskap til teknologien. InfluxData støtter og Docker med offisielle versjoner av TICK-komponentene, som ble avgjørende faktor i valget.

### 2.4.3 Orkestrering

Orkestrering handler om å sette opp en infrastruktur for å levere en eller flere tjenester. Når orkestrering utføres så vil et sett med tjenere opprettes og tildeles et sett med ressurser. Ressurser i dette tilfellet er prosessorkraft, arbeidsminne, lagringskapasitet og nettverkskomponenter [26]. For at et orkestreringsverktøy skal være verd tiden man bruker på å sette seg inn i det så må det oppfylle noen krav: det må være programmerbart, det må kunne kjøre når en selv velger, og man må kunne la enkeltpersoner definere egne maler for en tjeneste selv [27].

*Programmerbarhet* vil si at man kan skrive en mal og taste en kommando med denne malen som argument, og verktøyet vil sette opp infrastrukturen som definert. I en mal defineres hvor mange tjenerne som skal settes opp med en gitt mengde ressurser og vertsnavn. Dette vil også si at man har støtte for konfigurasjonsstyring av denne malen, slik at flere kan jobbe med samme infrastruktur med minimalt av konflikt.

*Kjøre når en selv velger* vil si at man ikke er avhengig av at et system kjører en kommando til en gitt tid. Dette handler om at man skal kunne sette opp en infrastruktur når det trengs.

*Å la enkeltpersoner definere egne maler* kan høres risikabelt ut, men fordelen vil være at hvis en utvikler vil få en nett-tjeneste oppegående så kan han gjøre det selv uten å gå via et helpdesk-system, som kan forårsake at unødvendig tid blir sløst bort på venting.

OpenStack har eget orkestreringsverktøy kalt Heat[12], som gruppemedlemmene er kjent med, og er godt egnet til prosjektets formål. I Heat deklareres infrastrukturen som kode, og man bruker en klient for å iverksette utrulling eller ødeleggelse i kommandolinje.

For orkestering av konteinere tas i bruk Docker Swarm[13], da det er integrert i verktøyspakken til Docker, er godt egnet for demonstrasjon av mikrotjenestearkitekturen for prosjektet, og gruppemedlemmene har kjennskap til det.

[11] https://www.openstack.org/
[12] https://wiki.openstack.org/wiki/Heat
[13] https://docs.docker.com/engine/swarm/

### 2.4.4 Provisjonering

Provisjonering handler om å sette opp en tjeneste med et viss konfigurasjon. Før i tiden måtte man besøke hver tjener en etter en for å manuelt sette opp en tjeneste. Moderne drift har gått i retningen *programmerbar infrastruktur* og at man i minst mulig grad skal inn på en tjener for å gjøre noe manuelt [28].

Med gode provisjoneringsverktøy kan man programmatisk definere en tjeneste, og flytte grovarbeidet med å installere, konfigurere, og aktivere tjenester over til verktøy som følger standardiserte prosesser. Ved endring av en definisjon vil tjenesten bli oppdatert på en av to måter: uforanderlig (immutable) eller foranderlig (mutable) [29]. Uforanderlig vil si at når konfigurasjonsdefinisjonen endres vil tjeneren slettes og bli reetablert fra bunn. Dette sikrer at det ikke eksisterer tidligere konfigurasjon som kan forårsake problemer. Sistnevnte gjør at tjeneren ikke slettes, men provisjoneringsverktøyet kjøres på nytt for å oppdatere konfigurasjonen. Provisjonering er også kjent som konfigurasjonsstyring.

Ett av kriteriene til oppgaven er å bruke Puppet som provisjoneringsverktøy (kapittel 1.2.3), men det finnes et hav av alternative teknologier med varierende syn på hvordan konfigurasjonsstyring bør gjøres. Puppet er skrevet i Ruby[14] og utviklet av Puppet [30]. Puppet lar en komponere installasjon, konfigurasjon, og aktivering av en tjeneste, et program, eller provisjonering av en hel tjener med flere tjenester. Det er og støtte for separasjon av data fra konfigurasjon i Puppet ved bruk av Hiera [31]. Med Hiera kan man sette variabler med bestemte verdier basert på vertsnavn eller andre faktorer på tjeneren som provisjoneres utenfor konfigurasjonen, og gjør det mulig å dele konfigurasjonen innad i bedriften uten å spre hemmeligheter til de som ikke trenger eller bør se det.

Beste praksis i produksjon med Puppet består av at man har en dedikert maskin som distribuerer konfigurasjonen for Puppet til alle andre maskiner. Denne maskinen blir dedikert til å være en Puppet-mester, mens alle andre maskiner er Puppet-agenter. Man setter opp agentene til å kjøre Puppet klienten på intervall, som så spør mesteren om det er noe nytt som skal gjøres basert på hva mesteren vet [32]. Agentene vil også passe på at maskinen alltid har en kjent tilstand.

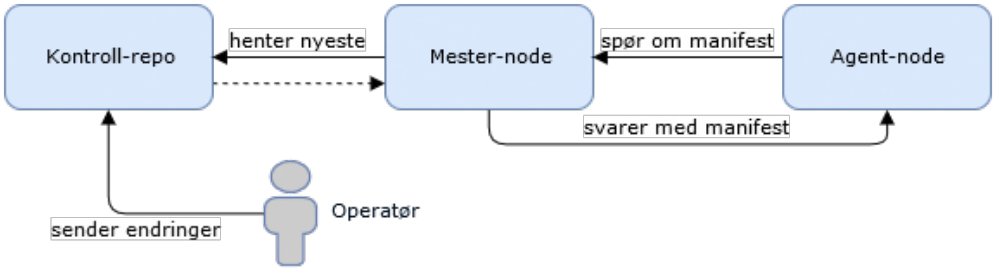
Med større infrastruktur vil man ha et sentralt sted hvor man holder definisjoner for alle tjenester og maskiner, og man passer på å føre historikken bak endringene. Til dette bruker man versjonskontroll slik som Git[15], samt en spesiell struktur for Puppet-koden. Puppet-manifester kalt profiler er ansvarlig for en enkelt tjeneste, og man ender opp med én profil for hver komponent i TICK, samt DNS, brannmur og lignende. Ikke alle maskiner skal kjøre samme programvare, så man bestemmer hvilke maskiner som skal ha hvilke profiler med inndeling i roller. Denne inndelingen gjøres basert på kjente verdier, slik som vertsnavn på maskiner, eksempelvis i kodesnutt 2.4.

```
node 'manager.lab' {
    include profile::base
    include profile::dns::server
}
```

Listing 2.4: Puppet-manifest for rollebestemmelser.

[14] https://www.ruby-lang.org/en/
[15] https://git-scm.com/

Alle Puppet-manifestene plasseres i et Git repository, oftest referert til i sammenheng med Puppet som et "kontroll-repo" [33], som gjøres tilgjengelig til Puppet-mesteren på distribuerte Git-tjenester slik som GitHub. Man bruker grener (branches) i Git for å definere alle miljø man kjører infrastrukturen i, slik som produksjon og testing.




Figur 3: Puppet-manifest distribusjon.

For at Puppet-mesteren skal kunne få nyeste konfigurasjon fra kontroll-repoet kan man bruke standard Git-operasjoner og klone ned manuelt, men dette er en operasjon man gjerne vil automatisere. Dette gjør man ved hjelp av r10k[16]. r10k er et verktøy som vedlikeholder et kontroll-repo hos Puppet-mesteren, og vil kunne installere og oppdatere eventuelle moduler definert i en "Puppetfile" i rot-katalogen for kontroll-repoet. Man vil kunne sette opp r10k til å gjøre dette automatisk på intervall, likt hvordan man lar Puppet-agenter kjøre på intervall, slik som skissert i figur 3.

### 2.4.5 Strategi

Når man skal designe en infrastruktur er det viktig å velge seg ut en strategi en følger. Tradisjonelt sett har man fulgt monolittisk tankegang, men i nyere tid så har distribuert tankegang blitt mer og mer populært.

Monolittisk tankegang er at man skriver en mal for alle relevante undersystem som leverer en tjeneste, og at alle disse systemene lever og dør samtidig. Det har blitt mindre og mindre populært å gruppere flere tjenester på én virtuell tjener, da dette kan by på single-point-of-failure og andre problemer som at man risikerer å påvirke flere tjenester når man gjør en endring til kun én tjeneste [34]. Oppdragsgiver har uttrykt ønske om å se en sammenligning mellom en monolittisk og en distribuert utgave av TICK.

Når man lager en distribuert tjeneste så vil det si at alle komponenter er såkalt løst koblet sammen. En løs kobling vil si at man ikke er avhengig av hverandre for å kunne kjøre normalt [35]. All kommunikasjon foregår som regel som et HTTP-forespørsel til et Restful API[17]. Dette gjør at hver komponent ikke trenger innsyn i hvordan de andre opererer. Når man har et sett med standariserte forespørsler er det lettere å utvikle flere tjenester til å jobbe sammen, uten at man trenger innsyn i hvordan spesifikke funksjoner fungerer.

En klar fordel med en monolittisk implementasjon er at man har mulighet til å redusere angrepsoverflaten, gitt at de undersystemene som bygger opp tjenesten ikke er avhengig av andre tjenester ute på Internet. Ved å la mest mulig av kontrolltrafikk og

[16] https://github.com/puppetlabs/r10k
[17] https://en.wikipedia.org/wiki/Representational_state_transfer

kommunikasjon mellom for eksempel komponentene i TICK flyte internt på én tjener, så blir det med ett vanskeligere å avlytte og forstyrre denne kommunikasjonen.

Ulempen med en distribuert implementasjon er kompleksiteten det innfører. Det blir flere bevegelige undersystem som må holdes styr på. Det betyr et større press på konfigurasjonsstyring og versjonskontrollering av koden som bygger opp infrastrukturen. Dette er samtidig dens fordel da det er mange små komponenter som kan, til en viss grad, skaleres og endres uavhengig av hverandre.

## 2.5 Ytelse

Når en tjeneste skal settes i produksjon så bør dens ytelse være kjent. Utrulling av en ny tjeneste kan sammenlignes med å kjøpe en bruktbil usett - den ser kanskje fin ut på papiret, men man kan ikke forutse hvordan den kommer til å oppføre seg på vegen uten å testkjøre den først. For å være sikker på dens stabilitet og kunne forutse dens ressursbruk må man teste tjenesten i forskjellige scenario, da metrikk fra monitorering kan være misforstående uten å ha et grunnlag for dens oppførsel under normale omstendigheter [36].

Ved ytelsestesting av tjenesten danner man grunnlaget for dens oppførsel, og får et bedre blikk i hvordan man kan vedlikeholde tjenesten optimalt. Dette gir det operasjonelle teamet mulighet til å kunne observere trender, utarbeide skaleringsstrategier, og reagere på kraftige rykk eller fall i ressursbruk.

## 2.6 Skalering

Skalering finnes i to former, enten ved å øke ressurser (vertikal), eller ved å replikere en tjeneste over flere maskiner (horisontal)[18].

### 2.6.1 Vertikal skalering

Alt kan i teorien skaleres i vertikal form, dvs. å utvide ressursene til maskinvaren de kjører på. Med dette gis det mer rom hvor programvare kan boltre seg, og vil da ha mulighet til å holde høyere ytelse. Denne form for skalering har vært tradisjonelt den eneste form for skalering.

### 2.6.2 Horisontal skalering

Alternativt kan man skalere horisontalt, med andre ord utvide antall maskiner programvaren kjører på. Dette krever støtte i selve programvaren, hvor den da vil kunne lenke sammen flere prosesser slik at de fungerer som en og samme enhet. Denne typen skalering er ofte referert til som "clustering". Selv om dette ofte er regnet som den beste måten å skalere på, kommer det med en kostnad på mer maskinvare (eventuelt virtualisert), samt mer arbeid i å opprette et cluster. En ekstra gevinst med horisontal skalering er økt redundanse, i at flere maskiner er ansvarlige for å opprettholde tjenesten, hvor enkelte kan gå ned uten at hele tjenesten forsvinner, noe som veier kraftig for denne typen skalering.

## 2.7 Sikkerhet

Sikkerhet er et stort fagfelt som dekker alt fra fysisk sikring av et område til analyse av kryptografiske funksjoner. Sikkerhetsfokuset i dette prosjektet kommer til å være hvordan

[18] https://stackoverflow.com/a/11715598

man kan sikre en digital infrastruktur ved å følge beste praksis. Dette innebærer verifisering av identitet med bruk av sertifikater, reduksjon av rettigheter med sterk kontroll av brukerkontoer, og filtrering av trafikk inn til tjenere.

Filtrering av trafikk kan anses som et grunnleggende og enkelt tiltak. Det som foregår i praksis er at man oppretter et sett med regler for hva slags trafikk som skal kunne passere gjennom en brannmur eller nettverksgrensesnitt [37]. Filtrering kan utføres enten av en dedikert brannmur, eller på hver tjener. For best sikkerhet så anbefales det å bruke begge deler. En regel vil typisk sett bestå av en nettverksadresse og en port som enten skal tillates eller nektes, med mulighet for å filtrere både på kilde hvor forespørsel opprettes og mål hvor forespørsel ender opp.

Kontroll av brukerkontoer innebærer å innskrenke rettighetene til en spesifikk bruker slik at tilgang er basert på hva som er nødvendig [38]. For eksempel så kan man legge til en bruker i InfluxDB som kun har leserettigheter i en gitt database. Verifisering av identitet kan gjøres på mange måter, man kan ha spesifikke kredentialer med tilhørende brukernavn som gjør at når en skal logge inn på en tjener så vil dette sjekkes enten mot en intern eller ekstern database, og deretter godkjennes eller nektes.

Verifisering av identitet brukes og ved digital kommunikasjon, for å bekrefte at den man kontakter virkelig er den de sier at de er. I data-verdenen brukes denne type verifisering ofte, i form av *Transport Layer Security* (TLS). TLS er en protokoll rettet mot å forsikre integritet mellom to kommuniserende parter, slik som mellom to maskiner over internett [39]. Dette gjøres ved at serveren har et sertifikat som er signert av en felles kjent enhet (Certificate Authority, CA), som klienten verifiserer i løpet av en håndhilsingsprosess. *Public Key Infrastructure* (PKI) er teknologien bak CA som håndterer nøkler brukt ved signering av sertifikatene, og er basert på en standard ved navnet X.509 [40]. PKI er basert på at den som signerer har et sett med private nøkler som kun brukes til signeringen og forlater aldri CA. En annen nøkkel basert på den private er distribuert offentlig og kan brukes for å verifisere sertifikatet.

# 3 Design

TICK-stacken skal implementeres i to forskjellige utgaver, monolittisk og distribuert, og stacken inneholder flere komponenter som vil måtte konfigureres på ulikt vis. Det er nødvendig å etablere hvordan alt skal settes sammen. I dette kapitlet diskuteres strategier for konfigurasjon og utrulling av TICK i de forskjellige utgavene, hvilke verktøy som tas i bruk, samt hvilke hensyn som må tas for hver utgave.

## 3.1 Infrastruktur

Ved å følge beste praksis i programmerbar infrastruktur vil hele infrastrukturen defineres på en reproduserbar måte [41]. Dette gjøres ved hjelp av orkestrering- og provisjoneringsverktøy, hvor definisjoner versjonskontrolleres. Reproduserbarheten og versjonskontrollen gir målgruppen mulighet til å anvende metodene i ettertid.

Hele infrastrukturen defineres i Puppet-manifester, som versjonskontrolleres i Git og distribueres på GitHub, som følger mester-agent strategien fra Puppets beste praksis. Puppet-mesteren får flere oppgaver, slik som navntjener for interne oppslag av maskiner, noe som gjør både konfigurasjon mellom tjenester langt lettere, og navigasjon mellom de en grei prosess ved feilsøking. For dette finnes det godt etablerte moduler for Puppet, som tas i bruk.

For mikrotjenestene vil Docker Swarm bli brukt, og Puppet brukes til å opprette og kontrollere denne svermen da det er godt etablerte moduler til rådighet. All konfigurasjon av tjenester som skal kjøre i svermen defineres i en YAML-fil[1], som Puppet distribuerer til den maskinen som er "mester" i svermen og kontrollerer at svermen alltid kjører denne som definert.

## 3.2 Hensyn til TICK

Hver komponent i TICK har et eget sett med behov som må oppfylles for at de skal fungere korrekt, både alene og når det skal kommuniseres fra en komponent til en annen.

Konfigurasjonen hos samtlige komponenter defineres i et språk kalt TOML[2], et minimalistisk språk designet for konfigurasjon med forståelig semantikk.

### Telegraf

Telegraf er ansvarlig for å samle inn metrikk på en vilkårlig maskin og deretter sende dette til en database. Minstekonfigurasjon for at Telegraf skal kjøre feilfritt er som følger:

- Hva som samles inn (kilde/input).
- Hvor ofte metrikk skal samles (intervall).
- Hvor metrikk skal sendes (destinasjon/output).
    - Dette innebærer en eller flere adresser til destinasjoner, samt port.
    - Hvis autentisering er i bruk må brukernavn og passord også supplementeres.

[1] https://yaml.org/
[2] https://github.com/toml-lang/toml

    - Hvis databasen er satt opp til å kryptere trafikk, men bruker et selv-lagd sertifikat må sertifikatet supplementeres hos Telegraf for å verifisere koblingen.

Både kilder og destinasjoner er veldig opp til hva som er nødvendig, så denne konfigurasjonen varierer veldig fra maskin til maskin.

**InfluxDB**

InfluxDB lagrer all metrikk tilsendt fra Telegraf agenter, og grunn-konfigurasjonen krever et sett med standardverdier for hvordan data lagres og minnebruk:

- Adresse og port databasen skal lytte på.
- Stier hvor data skal lagres på disk.
- Hvor lenge data skal beholdes (retention policy).

Utover disse parametrene, hvor de to siste har predefinerte standarder, kan man finspisse mange detaljer om det skulle være nødvendig, slik som nøyaktighet og bevaring av lagret data. Av sikkerhetshensyn bør man etablere autentiserting for kommunikasjon inn mot InfluxDB, samt at man bør kryptere trafikken. Krypteringen etableres enten ved sertifikater generert hos en offisiell leverandør, eller egengenererte sertifikater. Ved sistnevnte løsning må og klienter som skal snakke med InfluxDB supplementeres med sertifikatene, slik at man kan verifisere koblingen.

Retention policy bestemmer hvor lenge InfluxDB vil ta vare på metrikk, og er definert per database. Standardverdien her er uendelig, og man vil gjerne justere dette etter behov og kapasitet [42].

**Chronograf**

Chronograf er det visuelle grensesnittet mot InfluxDB og Kapacitor, og er en ren nettsidebasert løsning. Man må konfigurere minst:

- Adresse og port som Chronograf skal lytte på.
- Eventuelle sertifikater fra InfluxDB, samt egne sertifikater.

Parametere for kommunikasjon mot de andre komponentene kan defineres rett i konfigurasjonen, men er like lett å konfigurere fra grensesnittet i Chronograf når tjenesten er satt opp. Unntaket her er sertifikater, som likt med InfluxDB må genereres og konfigureres på forhånd.

Autentisering mot Chronograf skjer med tredjepart-leverandører, og er noe man bør sette opp før man setter Chronograf i produksjon.

**Kapacitor**

Kapacitor må konfigureres slik at den har mulighet til å snakke med InfluxDB, så minstekrav er slik:

- Adresse og port den skal lytte på.
- Adresse(r) og port(er) til InfluxDB.
- Eventuelle autentiseringsparametre slik som brukernavn og passord.
- Eventuelle sertifikater fra InfluxDB, samt egne sertifikater.

Kapacitor har og et drøss med andre konfigurasjonsparametre som ikke allerede er nevnt, da de er forskjellige etter behov, i likhet med Telegraf. Eksempelvis av konfigurasjoner

som går under denne kategorien er hvor varslinger skal sendes. Disse kan konfigureres fra Chronograf i etterkant.

## 3.3 Monolittisk

Per dags dato finnes kun én "offisiell" Puppet-modul for komponenter i TICK-stacken. Denne er skrevet av VoxPupuli, en anerkjent utvikler av Puppet-moduler, og dekker kun Telegraf [3]. Dette betyr at for å provisjonere InfluxDB, Kapacitor og Chronograf med Puppet trengs det egne moduler. Etter å ha undersøkt Telegraf-modulen viser det seg at den introduserer flere avhengigheter som enkelt kunne ha vært unngått, som strider mot Puppet sin egen beste praksis at en modul skal introdusere færrest mulig avhengigheter [43]. For å unngå unødvendige avhengigheter så vil en komplett modul som dekker alle fire komponentene bli utviklet.

Modulen må dekke alle målene med provisjonering:

- *Installasjon*
  Må kunne legge til nødvendig kilder for å hente ned programvare og deretter installere pakkene som kreves (RPM repository i CentOS/RHEL tilfelle).
- *Konfigurasjon*
  Må kunne sette nødvendig konfigurasjon slik at komponentene i TICK fungerer slik de skal, med standardverdier likt InfluxDatas design.
- *Tjenestekontroll*
  Må kunne starte og stoppe tjenesten slik at den kan nås av enten brukere eller andre maskiner i nettverket.

Videre bør modulen være godt støttet for flere distribusjoner, med minstekrav om RHEL-baserte. Det må enten lages fire helt separate moduler, eller én modul med underklasser som dekker behovet for alle komponentene i TICK. Konklusjonen var at én enkelt modul som dekker alle komponentene er best, da de installeres, konfigureres og styres på veldig lik måte, og dette vil unngå merarbeid. Da hver komponent i stacken trenger grunnleggende konfigurasjon som vil være tilpasset dens enkelte behov så trengs det enda en klasse som håndterer parametre og deres standardverdier. Dette vil resultere i at hver enkelt komponent med hver sin unike konfigurasjon kan inkluderes der det trengs.

Oppsummert må følgende arbeid utføres:

- En ny modul som dekker alle fire komponentene utvikles. Denne modulen må kunne utføre følgende oppgaver:
  - Installasjon
  - Konfigurasjon
  - Tjenestekontroll
- Telegraf og Chronograf trenger en høyst tilpasselig konfigurasjonsmal og trenger minimalt av konfigurasjon før de fungerer slik de skal.
- InfluxDB og Kapacitor trenger et svært omfattende sett med standardverdier til alle innstillinger som brukeren selv kan endre på ved behov. Begge har relativt høy kompleksitet i hver sin konfigurasjon med flere underliggende verdier som må eksistere for at tjenestene skal fungere nærmest feilfritt.

[3] https://forge.puppet.com/puppet/telegraf

## 3.4 Distribuert

Konfigurasjonen i distribuert sammenheng er relativt enkel i motsetning til den monolittiske, da InfluxData distribuerer offisielle Docker bilder for samtlige av komponentene[4]. Hvilke ting som kan, bør, eller må konfigureres er likt den monolittiske hvor standardverdier er forhåndsdefinerte. Det eneste man trenger her er å oppgi eventuell autentiseringsparametere, samt koblinger mellom de forskjellige komponentene.

Alle konfigurasjonsparametere defineres ved hjelp av miljøvariabler i motsetning til konfigurasjonsfiler, da dette er en alternativ måte å konfigurere alle komponentene i TICK i stedet for å bruke konfigurasjonsfiler direkte [44][45][46][47].

Med provisjonering i Puppet finnes det to valg for å håndtere konteinere: enten provisjonere enkelte konteinere hver for seg, eller kontrollere orkestreringsverktøy for konteinere fra Puppet. Sistnevnte følger prinsipper fra programmerbar infrastruktur best [41].

Puppet brukes til å provisjonere maskinene som skal delta i svermen, oppretter selve svermen mellom maskinene, samt det å fortelle svermen om tjenestene som skal kjøres. Alle tjenestene blir definert i egen konfigurasjonsfil kjent som en “compose”-fil, skrevet i YAML. Dette dekkes av offisielle moduler i Puppet.

## 3.5 Sikkerhet

En viktig del med å designe en infrastruktur er å passe på at det man implementerer av tjenester er sikret. Både maskinene som skal kjøre tjenestene samt tjenestene selv må sikres.

For å redusere angrepsvektoren mot maskinene selv vil man begrense direkte tilgang til de, ved å kun la én maskin være tilgjengelig for ekstern tilkobling med SSH, og deretter la denne maskinen være eneste vei inn i de andre maskinene i infrastrukturen [48]. Videre konfigureres brannmurer på hver maskin, hvor man hvitelister kun tilgang til tjenester fra maskiner man vet skal ha tilgang til de [49].

Kommunikasjon mellom tjenester sikres ved hjelp av autentiseringsmekanismer mellom hver enkelt tjeneste med brukernavn og passord, og man kan i tillegg sikre kommunikasjonen mellom de ved hjelp av sertifikater.

## 3.6 Skalering

Skalering i TICK-stacken omhandler kun to av komponentene. Telegraf er en agent som kjører på hver enkelt node som skal overvåkes eller har elementer som skal overvåkes, og vil dermed ikke være skalerbar. Chronograf er et visualiserende hjelpeverktøy og vil heller ikke skaleres. Da gjenstår kun to komponenter, InfluxDB og Kapacitor, hvorvidt begge har mulighet for skalering. Hvordan de skaleres er relativt likt, da det er enten ved å øke ressurser (vertikal skalering), eller replikere i Enterprise utgavene (horisontal skalering).

Hvilken skaleringsstrategi som er best varierer etter behov for den enkelte, og selv om horisontal skalering er regnet som overlegen er den ofte unødvendig, og kommer i sammenheng med TICK med ekstra kostnad i lisenser.

[4] https://github.com/influxdata/influxdata-docker

# 4 Utvikling av modul

Det finnes ingen offisielle Puppet-moduler for TICK i sin helhet, noe som betyr at for en monolittisk implementasjon er det nødvendig å utvikle en. I kapittel 3 ble det satt krav til hva en slik modul må håndtere. Dette kapitlet vil gå gjennom det praktiske arbeidet med fokus på et overordnet synspunkt om fellestrekk og hvordan modulen er sydd sammen.

All kode som resultat av utviklingen av modulen ligger åpent offentlig på GitHub[1], og leseren er velkommen til å se på, bruke og basere egen modul basert på resultatene fra prosjektet.

## 4.1 Overordnet struktur

Hver komponent i TICK trenger tre underklasser som skal installere, konfigurere, og aktivere komponenten med et sett standardinnstillinger som skal kunne endres ved behov. Puppet har selv et sett med krav om hvordan en modul bør struktureres for å holde god standard. Puppet-kode skrives i det som heter manifest, hvor man beskriver hvordan applikasjonen eller tjenesten skal håndteres. Dette blir oversatt av Puppet til instruksjoner som det underliggende operativsystemet forstår. For at en modul skal fungere så må modulen ha en base-klasse eller topp-klasse som beskrives i "init.pp". Denne lar en kalle på modulen og dens underklasser fra et kontroll-repo. I en modul med flere underklasser er det ikke nødvendig at denne klassen inneholder noe som helst, da hver underklasse tilkalles direkte.

I figur 4 kan man se strukturen for hele modulen. Puppet følger en struktur hvor manifest som er brukt av Telegraf-klassen legges i undermappen som heter "telegraf". Denne strukturen gjør at hver komponent sin dedikerte underklasse kan bruke hver sine klasser selv om det er flere med samme navn, for eksempel "install.pp". Filene listet opp under Telegraf-mappen finnes i de tre andre mappene også, men for å spare plass er de ikke tatt med.

Denne strukturen gjør det lettere å dele opp utviklingen av de forskjellige komponentene. Alternativt kan man utvikle en modul for hver komponent i TICK, noe som simplifiserer strukturen, men man mister muligheten til å gjenbruke kode. Eksempelvis vil strukturen da se ut slik som i figur 5.

InfluxData distribuerer pakker til komponentene i TICK til flere operativsystem[2], men en udokumentert mulighet er at de har distribueringskanaler for enkelte operativsystem slik som RHEL [50]. Disse defineres i en egen fil navngitt "repo.pp" som brukes av alle komponentene og lar modulen installere nyeste versjon av hver komponent med eksisterende pakkestyringsverktøy uten å kreve oppdatering av modulen.

[1] https://GitHub.com/vetletm/tick_stack
[2] https://www.influxdata.com/get-influxdb/

```
manifest
├── init.pp
├── repo.pp
├── telegraf.pp
├── influxdb.pp
├── chronograf.pp
├── kapacitor.pp
├── telegraf
│   ├── config.pp
│   ├── install.pp
│   ├── params.pp
│   └── service.pp
├── influxdb
│   └── ...
├── chronograf
│   └── ...
└── kapacitor
    └── ...
```

Figur 4: Puppet-modul mappestruktur.

```
manifest
├── init.pp
├── repo.pp
├── config.pp
├── install.pp
├── params.pp
└── service.pp
```

Figur 5: Puppet-modul mappestruktur for enkeltklasse.

## 4.2 Fellestrekk

Det første steget i utviklingen var å identifisere hvilke verdier/variabler som alle komponentene måtte ha. Herunder finner man eksempelvis om pakken skal eksistere på systemet eller ikke, et flagg som inneholder filstien til konfigurasjonsfilen og til slutt et flagg for hvilken status tjenesten skal ha (kjørende eller stoppet). Kodesnutt 4.1 viser et eksempel fra params-klassen i Telegraf.

```
class tick_stack::telegraf::params {
  # Package options
  $ensure      = 'present'

  # Config options
  $conf_path   = '/etc/telegraf/telegraf.conf'
  $template    = 'tick_stack/telegraf.conf.erb'

  # Service options
  $service     = 'running'
  $enable      = true
  $hasrestart  = true
  $hasstatus   = true
}
```

Listing 4.1: Params-klasse, Telegraf.

De tre verditypene i eksempelkode 4.1 går igjen i alle komponentene som gjør at hver underklasse blir litt lettere å sette seg inn i. I toppklassen for hver komponent inkluderes "repo.pp" filen og alle underklasser. Rekkefølgen for hvordan hver underklasse kjøres kontrolleres med "before"- og "notify"-snarveier i form av "->" og " >" symboler. Eksempelkode 4.2 viser topp-klassen for Telegraf.

```
class tick_stack::telegraf (
  # For Install-class
  $ensure      = $tick_stack::telegraf::params::ensure,

  # For Config-class
  $conf_path   = $tick_stack::telegraf::params::conf_path,
  $template    = $tick_stack::telegraf::params::template,

  # For Service-class
  $service     = $tick_stack::telegraf::params::service,
  $enable      = $tick_stack::telegraf::params::enable,
  $hasrestart  = $tick_stack::telegraf::params::hasrestart,
  $hasstatus   = $tick_stack::telegraf::params::hasstatus,

  # Specific configuration hashes:
  $global_tags = $tick_stack::telegraf::params:: ↙
      global_tags,
  $agent        = $tick_stack::telegraf::params::agent,
  $outputs      = $tick_stack::telegraf::params::outputs,
  $inputs       = $tick_stack::telegraf::params::inputs,

) inherits tick_stack::telegraf::params {

  include tick_stack::repo

  contain tick_stack::telegraf::install
  contain tick_stack::telegraf::config
  contain tick_stack::telegraf::service

  Class['tick_stack::telegraf::install']
  -> Class['tick_stack::telegraf::config']
  ~> Class['tick_stack::telegraf::service']
}
```

Listing 4.2: Topp-klasse, Telegraf.

Toppklassene vil sørge for at verdier som ikke er deklarert gjennom en node-definisjon blir satt til standardverdier ved å hente de fra den relevante params-klassen. Deretter vil toppklassen sjekke at repositoriet til pakkene er lagt inn i systemet. Til slutt utføres installasjon, konfigurasjon, og tjenesteaktivering i riktig rekkefølge. Man benytter som regel en notify-pil på tjeneste-klassen, noe som gjør at hvis manifestet blir kjørt flere ganger etter hverandre så er det kun en konfigurasjonsendring som kan utløse en omstart av tjenesten.

Installasjonsklassen henter ned verdien satt i toppklassen og sørger for at pakkens tilstand er som spesifisert, slik som vist i eksempelkode 4.3.

```
class tick_stack::telegraf::install {
  assert_private()

  $ensure = $tick_stack::telegraf::ensure

  package { 'telegraf':
    ensure => $ensure,
  }
}
```

Listing 4.3: Installasjonsklasse, Telegraf.

Man kan bemerke seg `assert_private()`-linjen i eksempelkode 4.3, 4.4 og 4.5. Denne sørger for at ingen av underklassene kan kalles som selvstendige klasser. Konfigurasjonsklassen i eksempelkode 4.4 vil sette korrekte rettigheter på konfigurasjonsfilen og sørge for at den blir oppdatert med korrekt innhold, formattert med en template-fil i TOML-språk.

```
class tick_stack::telegraf::config {
  assert_private()

  $conf_path = $tick_stack::telegraf::conf_path
  $template  = $tick_stack::telegraf::template

  file { $conf_path:
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template($template)
  }
}
```

Listing 4.4: Config-klasse, Telegraf.

Når Telegraf er installert og konfigurert kalles til slutt tjeneste-klassen hvis en konfigurasjonsendring er utført, definert slik som i eksempelkode 4.5. Ved førstegangskjøring utføres dette steget når konfigurasjonsfilen går fra å ikke eksistere til å eksistere. Dermed vil tjenesten hente inn korrekt konfigurasjon og starte opp.

```
class tick_stack::telegraf::service {
  assert_private()

  $service    = $tick_stack::telegraf::service
  $enable     = $tick_stack::telegraf::enable
  $hasrestart = $tick_stack::telegraf::hasrestart
  $hasstatus  = $tick_stack::telegraf::hasstatus

  service { 'telegraf':
    ensure     => $service,
    enable     => $enable,
    hasrestart => $hasrestart,
    hasstatus  => $hasstatus,
  }
}
```

Listing 4.5: Tjeneste-klasse, Telegraf.

Eksempelvis implementasjon av hele TICK-stacken ved bruk av denne modulen vil foregå slik som i eksempelkode 4.6, hvor i en så simplifisert utgave kun bruker standardverdier for alt av konfigurasjon.

```
node default {
  include tick_stack::influxdb
  include tick_stack::telegraf
  include tick_stack::chronograf
  include tick_stack::kapacitor
}
```

Listing 4.6: Eksempelvis bruk av modulen.

## 4.3 Spesielle hensyn

### Telegraf

For å håndtere at hver bruker av stacken vil ha ulike krav til hva som monitoreres, vil Telegraf-klassen benytte seg av "hashes" som vil itereres over i en malkonfigurasjon. En "hash" består av et sett med nøkkel-verdi-par som kan utnyttes til for eksempel konfigurasjonslinjer i en TOML-fil. I eksempel 4.7 ser man hvordan en slik hash kan være bygd opp. Noen av integreringene i Telegraf krever kun en deklarasjon og ikke en eksplisitt definisjon for at de skal samle inn metrikkmålinger. Et eksempel på dette ser man i "`'system' => {}`" som er definert som en tom hash.

```
$inputs = {
  'cpu' => {
    'percpu'   => true,
    'totalcpu' => true,
  },
  'system' => {}
}
```

Listing 4.7: Hash-eksempel, Telegraf.

Ved kjøring vil eksempelkoden 4.7 bli tolket i en ERB-malkonfigurasjon som vist i eksempel 4.8. Her viser man hvordan den opprinnelige hashen vil itereres over av Puppet og produsere en fungerende konfigurasjonsfil som Telegraf benytter seg av mens den kjører. Resultatet av dette er vist i eksempel 4.9 hvor man da har i tillegg brukt standardverdiene til de andre hashene som Telegraf krever.

```
<% if scope['tick_stack::telegraf::inputs'] -%>
    # INPUTS
    <% scope['tick_stack::telegraf::inputs'].each do |key, ↙
        hash| -%>
        [[inputs.<%= key %>]]
        <% hash.each do |key,value| -%>
            <%= key %> = <%= value %>
        <% end -%>
    <% end -%>
<% end -%>
```

Listing 4.8: Eksempel på en ERB-mal, utdrag fra Telegraf-klassen.

```
[global_tags]
  dc = "home"

[agent]
  interval = "10s"

# OUTPUTS
[[outputs.influxdb]]
  url = "http://localhost:8086" # required
  database = "telegraf" # required
  precision = "s"

# INPUTS
[[inputs.cpu]]
  percpu = true
  totalcpu = true
[[inputs.system]]
  # No key-value pairs required
```

Listing 4.9: Konfigurasjonseksempel, Telegraf.

Normal bruk uten Puppet for konfigurasjonsstyring ville en ha installert Telegraf og deretter skrevet konfigurasjonsfilen manuelt. Bruk av hashes lar brukerene enkelt definere hvordan Telegraf skal kjøre i Puppet-kode og de lar seg utvide uten problem da Puppet er idempotent av natur. Hvis brukeren ønsker å bruke alle standardverdier med unntak av hvor ofte Telegraf skal sende metrikkmålinger kan man ta i bruk modulen slik som i eksempelkode 4.10.

```
node default {
  class { 'tick_stack::telegraf':
    agent => {
      'interval' => '30s',
    }
  }
}
```

Listing 4.10: Installasjon vha. Puppet, Telegraf.

Når Puppet kjøres vil standardverdier i `inputs`, `outputs`, og `global_tags` legges inn i malen, mens `agent` vil erstattes med det brukeren angir.

### Chronograf

Man kan starte tjenesten med en del flagg eller med et sett med miljøvariabler. Denne modulen utnytter det faktum at Chronograf vil hente inn miljø-variabler fra filen ‘‘`/etc/default/chronograf`’’ og bruker dermed enkle string-variabler for alle flaggene.

Hvis man kjører modulen med kun ‘‘`include tick_stack::chronograf`’’ så vil Chronograf bli satt standardverdier likt eksempelkode 4.11.

```
HOST=0.0.0.0
PORT=8888
INFLUXDB_URL=http://127.0.0.1:8086
```

Listing 4.11: Konfigurasjonseksempel, Chronograf.

Chronograf vil lytte etter alle innkommende forespørsler på port 8888, og vil prøve å kontakte InfluxDB på angitt URL. Man kan og spesifisere autentisering mot InfluxDB her.

### InfluxDB

InfluxDB stiller store krav til hvilke standardverdier som er satt for at tjenesten skal fungere som normalt. Alt dette vil være forhåndsdefinert i params-klassen som lar brukeren kunne kjøre databasen uten å ha innsyn i nøyaktig hvilke innstillinger som må være satt. Blant disse verdiene finner en hvor innholdet i databasen skal lagres og hvor informasjon om selve databasen skal lagres. Man må og ta høyde for maksimal minnestørrelse for mellomlagring av metrikkmålinger og hvor mange metrikkmålinger en skal holde i databasen. Eksempelkode 4.12 viser noen standardverdier.

```
class tick_stack::influxdb::params {
  # Specific options for InfluxDB config
  $bind_addr = '127.0.0.1:8088'
  $meta_dir = '/var/lib/influxdb/meta'
  $data_dir = '/var/lib/influxdb/data'
  $wal_dir = '/var/lib/influxdb/wal'

  # Optionals for InfluxDB data config,
  # defaults are InfluxDBs recommended settings
  $trace_logging_enabled              = false
  $query_log_enabled                  = true
  $cache_max_memory_size              = 1048576000
  $cache_snapshot_memory_size         = 26214400
  $cache_snapshot_write_cold_duration = '10m'
  $compact_full_write_cold_duration   = '4h'
  $max_series_per_database            = 1000000
  $max_values_per_tag                 = 100000

  # HTTP config hash
  $http             = {
    'http_enable' => true,
    'http_bind'   => '":8086"',
    'http_auth'   => false,
  }
}
```

Listing 4.12: Standardverdier, InfluxDB.

### Kapacitor

Denne komponenten følger samme prinsipp som InfluxDB i det at det er satt en del standardverdier som kan overskrives etter behov.

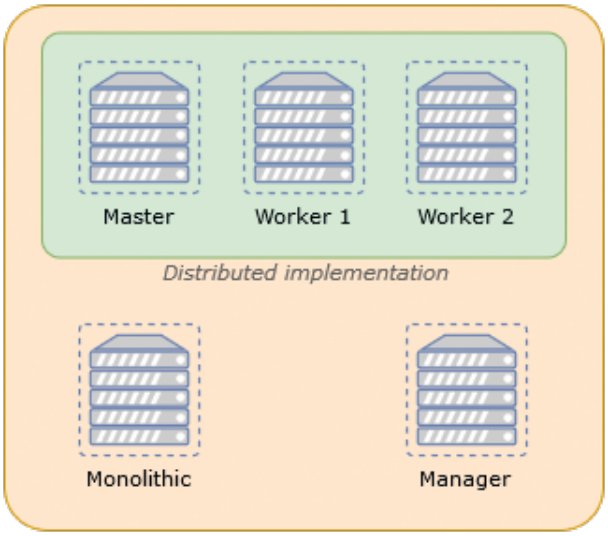
# 5 Implementasjon

Dette kapitlet gir et detaljert innsyn i hvordan hver implementasjon av TICK er utført og den grunnleggende infrastrukturen som alt kjører på.

All kode som resultat av utviklingen av infrastrukturen ligger åpent offentlig på GitHub[1], og leseren er velkommen til å se på og bruke alt.

## 5.1 Infrastruktur

Ved å følge krav til oppgaven og beste praksis rundt dette, slik som diskutert i kapittel 2.4 og 3.1, vil prosjektet bli implementert i to utgaver, med en Puppet-mester-agent løsning for konfigurasjonsstyring. All maskinvare er virtualisert i tilgjengelig sky på campus, og orkestreres ved bruk av OpenStack Heat. Dette innebærer opprettelse av et kontroll-repo som versjonskontrolleres og distribueres på GitHub, som automatisk vil bli hentet inn i infrastrukturen ved hjelp av r10k. Denne løsningen blir ansvarlig for å konfigurere og vedlikeholde all infrastruktur i kjent tilstand, med mulighet til å utvide etter behov.




Figur 6: Infrastrukturoversikt.

Eksempelvis utforming av infrastrukturen vises i figur 6, hvor man finner en "manager" node med ansvar for DNS, inngang for fjerntilgang, og vil være Puppet-mester for hele infrastrukturen. På samme nivå befinner den monolittiske implementasjonen seg, som én enkelt maskin hvor et helt TICK-stack kjører. Innenfor dette befinner det seg tre noder som deltar i en Docker-sverm, med en dedikert leder og to arbeidere. Denne kjører mikrotjenesteutgaven av TICK-stacken og er uavhengig av den monolittiske, men kontrolleres likt ved hjelp av Puppet.

[1] https://github.com/p3lim/tick-stack-conf

### Orkestrering

OpenStack Heat brukes for å orkestrere hele infrastrukturen ut i skyen, hvor alle komponenter slik som nettverk, virtuelle maskiner, flytende IP-adresser og sikkerhetsgrupper defineres. Heat er definert i YAML, et data-definerende språk, med eksempelvis definering av en node vises i eksempelkode 5.1.

```
resources:
  instance:
    type: OS::Nova::Server
    properties:
      name:       { get_param: hostname }
      image:      { get_param: image }
      flavor:     { get_param: flavor }
      key_name:   { get_param: key_name }
      networks:
        - port:   { get_resource: instance_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: node.bash }
          params:
            manager_ip_address: { get_param: manager_ip }
            dns_ip_address:     { get_param: dns_ip }
```

Listing 5.1: Heat template utsnitt for definisjon av en node.

Grunnleggende programvare nødvendig for opprettelse av maskiner defineres som en del av orkestreringen, slik at infrastrukturen har noe å starte med før denne oppgaven blir tatt over av Puppets konfigurasjonsstyring. Til dette brukes en mekanisme populært brukt i orkestreringssystemer kalt "cloud-init"[2], som rett og slett bare er et normalt shell-skript som kjøres ved første oppstart av hver maskin. Her defineres grunnleggende konfigurasjon slik som nettverk, DNS og bunnkonfigurasjon til Puppet-agenter og mester slik at agenter vet hvor mester befinner seg, og mester vet hvor konfigurasjonen befinner seg. Et utsnitt av et cloudinit script vises i eksempelkode 5.2, og blir referert i eksempelkode 5.1 som "node.bash" under "user_data".

```
# install puppet
rpm -Uvh https://yum.puppet.com/puppet6/puppet6-release-el ↙
    -7.noarch.rpm
yum install -y puppet-agent

# configure the puppet's interval and master, then run it
puppet config set server manager.lab --section main
puppet config set runinterval 300 --section main
puppet agent -t # request certificate
puppet agent -t # configure

# enable and start puppet
puppet resource service puppet ensure=running enable=true
```

Listing 5.2: Cloud-Init utsnitt for initisiell oppstart av Puppet.

Når alt har startet opp vil Puppet-logikk ta over og fullføre implementasjonen.

[2] https://cloud-init.io/

### Provisjonering

Etter orkestrering av maskinvare, med grunnleggende oppsett hvor Puppet kan ta over, vil r10k først hente ned nyeste konfigurasjon. Dette automatiseres på et intervall, slik at den alltid har nyeste til enhver tid. Mens denne jobber med å laste ned konfigurasjonen og installerer alle moduler slik definert i kontroll-repoet sitter hver eneste Puppet agent og spør "hei, har det kommet noen oppdateringer?". Dette er definert til å kjøre på intervall, slik at det blir en synergi mellom henting av konfigurasjon og utrulling av den.

I konfigurasjonen er det definert grunnleggende infrastruktur slik som DNS, hvordan Puppet skal distribuere data mellom maskinene ved bruk av Hiera og PuppetDB. Videre er brannmurfunksjonalitet definert ved IPtables, slik at kun porter som trenger å være åpne blir det, og begrenser til kun å tillate tilgang fra maskiner i infrastrukturen som skal ha tilgang får det. Eksempelkode 5.3 viser konfigurasjon av PuppetDB hvor standardverdier brukes, samt hvordan brannmur blir åpnet for portene PuppetDB snakker over.

```
class profile::puppetdb {
  # Configure PuppetDB using the default parameters.
  include puppetdb
  include puppetdb::master::config

  # Accept communication on TCP port 8140 for PuppetDB
  ::profile::firewall::management { 'PuppetDB TCP':
    port     => 8140,
    protocol => 'tcp',
  }
}
```

Listing 5.3: Brannmurkonfigurasjon i Puppet.

I et kontroll-repo designerer man roller til maskiner, hvor en rolle definerer hvilke profiler som skal tas i bruk. I eksempelkode 5.4 vises hvordan roller blir tildelt maskiner i infrastrukturen. Rollene designeres basert på vertsnavn, hvor i tilfellet ved "worker"-nodene brukes regulære uttrykk for å treffe flere noder med tilnærmet like navn.

```
node 'manager.lab' {
  include ::role::manager
}

node 'master.lab' {
  include ::role::master
}

node /^worker-\d+\.lab$/ {
  include ::role::worker
}

node 'monostack.lab' {
  include ::role::monolithic
}
```

Listing 5.4: Rollefordeling i kontroll-repo.

## 5.2 Monolittisk

I kapittel 4 er det redegjort for hvordan modulen har blitt utviklet og hvordan den kan brukes i enkle manifest. Bruk av modulen går med på å tilkalle klassene for hver komponent, enten med å benytte egendefinerte verdier etter behov med `class` eller med bruk av standardvedier ved å benytte `include`. Eksempelvis i 5.5 vises konfigurasjonen for den monolittiske implementasjonen.

```
class profile::monotick {
  # Get authentication credentials from Hiera
  $telegraf_user = lookup('influx::telegraf_user')
  $telegraf_pass = lookup('influx::telegraf_pass')

  # Include with default values
  include tick_stack::influxdb
  include tick_stack::chronograf
  include tick_stack::kapacitor

  # Declare Telegraf with custom parameters
  class { 'tick_stack::telegraf':
    inputs => {
      'cpu' => {
        'percpu'   => true,
        'totalcpu' => true,
      },
      'system'    => {},
      'mem'       => {},
      'net'       => {},
      'netstat'   => {},
    },
    outputs => {
      'influxdb' => {
        'urls'      => "[\"http://127.0.0.1:8086\"]",
        'database'  => '"telegraf"',
        'precision' => '"s"',
        'username'  => "\"${telegraf_user}\"",
        'password'  => "\"${telegraf_pass}\"",
      }
    }
  }
}
```

Listing 5.5: Profil, monolittisk TICK.

Denne profilen vil sette opp InfluxDB, Chronograf, og Kapacitor med forhåndsdefinerte standardverdier. Dette vil fungere fint da alle standardverdier er satt opp slik at hele stacken bruker `localhost` som kontaktpunkt for hver komponent.

Standardverdiene til Telegraf-klassen inkluderer kun system og CPU integrasjonene for monitorering. I en mer reell infrastruktur er det fornuftig med et mer detaljert innsyn. Telegraf-klassen vil i eksempel 5.5 sette opp monitorering for minne og nettverksbruk i tillegg til system og prosessor. Til slutt legges konfigurasjonen om hvor InfluxDB befinner seg med kredentialer slik at Telegraf får skrevet sine metrikkmålinger til databasen.

Når én eller flere profiler er definert kan man bruke disse i spesifikke roller. I eksempel 5.6 ser man hvordan den monolittiske utgaven installeres ved å kalle på dens profil. Dette gjør en infrastruktur provisjonert med Puppet svært oversiktlig og utvidbar.

```
class role::monolithic {
  include profile::base
  include profile::monotick
}
```

Listing 5.6: Rolle, monolittisk TICK.

## 5.3 Mikrotjeneste

Docker har utmerket støtte i Puppet, installasjon på alle maskiner kan gjøres på én enkelt linje slik som i eksempelkode 5.7. Man kan bestemme ekstra parametre slik som å håndheve versjoner som installeres, noe man helst vil gjøre i produksjon.

```
node default {
  include 'docker'
}
```

Listing 5.7: Installasjon av Docker med Puppet.

På lik linje med den monolittiske utgaven blir roller tildelt til maskinene i mikrotjenesteinfrastrukturen, og rollene kaller på profiler slik som vist i eksempelkode 5.8. Her blir "master"-rollen designert profiler for Docker Swarm manager, etablering av stacken samt Telegraf. Arbeiderne under "worker"-rollen er veldig lik master, med hovedforskjell at de blir vervet inn i svermen.

```
class role::master {
  include profile::base
  include profile::swarm::manager
  include profile::swarm::stack
  include profile::influx::telegraf
}

class role::worker {
  include profile::base
  include profile::swarm::worker
  include profile::influx::telegraf
}
```

Listing 5.8: Profiler i roller, mikrotjeneste.

### 5.3.1 Docker Swarm

Opprettelse av sverm krever litt ekstra konfigurasjon, da Docker-modulen ikke har innebygde kontroller for å håndtere deling av tokens i sverm. Dette løses med å mellomlagre tokens som fakta på mesteren. Oppsettet vises i eksempelkode 5.9, som kjøres på ledernoden i mikrotjeneste implementasjonen.

```
class profile::swarm::manager {
  # Install Docker
  include 'docker'

  # Initiate swarm
  docker::swarm { 'sverm':
    init           => true,
    advertise_addr => $::ipaddress,
    listen_addr    => $::ipaddress,
  }
```

```
  # Create a script that will pull tokens
  $swarm_token = '#!/bin/bash
  echo "swarm_token=$(docker swarm join-token worker -q)"'

  file { '/etc/puppetlabs/facter/facts.d/swarm_token.sh':
    owner   => 'root',
    group   => 'root',
    content => $swarm_token,
    mode    => '0755',
  }
}
```

Listing 5.9: Oppsett av Swarm med Puppet.

Konfigurasjonen på alle arbeider-noder består av å hente denne fakta fra lederen og deretter bli med i svermen, slik som i eksempelkode 5.10.

```
class profile::swarm::worker {
  # Install Docker
  include 'docker'

  # Get facts from the master
  $master = puppetdb_query('inventory[facts] {facts. ↙
      trusted.certname ~ "master"}')

  # Join the swarm
  docker::swarm { 'sverm':
    join           => true,
    advertise_addr => $::ipaddress,
    listen_addr    => $::ipaddress,
    manager_ip     => $master[0]['facts']['ipaddress'],
    token          => $master[0]['facts']['swarm_token'],
  }
}
```

Listing 5.10: Bruk av FQDN for å bestemme roller i Puppet.

Med svermen startet kan man bruke de offisielle Docker-bildene fra InfluxData til å kjøre alle komponentene i TICK. Dette startes som en stack i svermen. InfluxDB opprettes slik som i eksempelkode 5.11, og de resterende komponentene i TICK opprettes på lik måte. Man deklarerer hvilket bilde man vil bruke med spesifikk versjon, hvor data skal lagres med volum og hvilke porter som skal åpnes. Alt av konfigurasjon blir gjort ved å bruke miljøvariabler, hvor i eksempelet utnyttes Hiera til å hente disse.

```
influxdb:
  image: influxdb:1.7-alpine
  hostname: influxdb
  volumes:
    - influxdb-data:/var/lib/influxdb
  ports:
    - 8086:8086
  environment:
    INFLUXDB_DB: <%= lookup('influx::db_name') %>
    INFLUXDB_HTTP_AUTH_ENABLED: 'true'
    INFLUXDB_ADMIN_USER: <%= lookup('influx::admin_user') ↙
        %>
    INFLUXDB_ADMIN_PASSWORD: <%= lookup('influx:: ↙
        admin_pass') %>
    INFLUXDB_USER: <%= lookup('influx::telegraf_user') %>
```

```
    INFLUXDB_USER_PASSWORD: <%= lookup('influx:: ↙
      telegraf_pass') %>
```

Listing 5.11: InfluxDB-konfigurasjon i Swarm.

Puppetkonfigurasjonen for å iverksette svermen med en gitt Docker stack blir slik som i eksempelkode 5.12, hvor man kopierer stack-konfigurasjonen til leder-noden og starter stacken. Her konfigureres og brannmur slik at komponentene i TICK har mulighet til å kommunisere med hverandre over flere maskiner i svermen.

```
class profile::swarm::stack {
  # Copy stack config using EPP templating
  file { '/tmp/tick-stack.yaml':
    ensure  => file,
    content => epp('profile/tick-stack.yaml.epp'),
  }

  # Start the stack
  docker::stack { 'tick':
    stack_name    => 'tick',
    compose_files => ['/tmp/tick-stack.yaml'],
    require       => File['/tmp/tick-stack.yaml'],
  }

  # Open up the firewall for all ports required by ICK
  ::profile::firewall::management { 'InfluxDB TCP':
    port     => 8086,
    protocol => 'tcp',
  }

  ::profile::firewall::management { 'Kapacitor TCP':
    port     => 9092,
    protocol => 'tcp',
  }

  ::profile::firewall::public { 'Chronograf TCP':
    port     => 8888,
    protocol => 'tcp',
  }
}
```

Listing 5.12: Definisjon av Master-node.

Telegraf vil kjøre utenfor Docker på samtlige av maskiner likt den monolittiske implementasjonen, da den har innebygde utvidelser for å monitorere konteinere fra utsiden.

# 6 Sikkerhet

Informasjonssikkerhet er et stort og omfattende tema, og man er nødt til å trekke ut de områdene som er mest relevante til dette prosjektet. Det som må sikres er kommunikasjon mellom TICK-komponentene, til dette kan man utnytte HTTPS/TLS for verifisering og kryptering av trafikk. Begrensning av tilgang til tjenestene kan løses ved bruk av brannmur-policier, samt autentisering og autorisering. Til slutt må den underliggende infrastrukturen sikres best mulig ved å følge beste praksis.

Det vil alltid være forskjellig behov hos den enkelte. Sikker kommunikasjon mellom to tjenester i et lukket nettverk er ikke alltid nødvendig og kan skape merarbeid for de som skal implementere det. Sikker lagring av informasjon som ikke er sensitiv er ikke alltid nødvendig eller praktisk å implementere, samt at det krever ekstra ressurser i bruk i form av disk- og filkryptering.

Så snart en tjeneste skal være tilgjengelig i et åpent nett, slik som for eksempel en nettjener, må sikkerhet være i fokus. Sertifikater må brukes for å ivareta konfidensialiteten av informasjonen som flyter og for å verifisere at trafikken ikke kommer på avveie. Konfigurering av tilgjengelighet på maskiner er også et viktig tema, hvor man typisk bruker brannmurfunksjonalitet for å bestemme hva som skal, og ikke skal, være åpent mot allmennheten. Dette kan man konfigurere på flere måter, enten på nettverkssiden, ved operativsystem eller direkte i tjenestene selv.

## 6.1 Sikkerhet i TICK

Sikkerheten blant alle komponentene i TICK håndteres meget likt. Trafikkflyt sikres ved hjelp av sertifikater, og man bruker enkel autentisering med brukernavn og passord for verifisering av brukere. Unntaket her er for Chronograf, som ikke har noen form for egendefinert autentisering, men heller bruker eksisterende systemer hos en tredjepart.

Alternativt kan man sette opp en revers proxy tjeneste som vil være et mellomledd for besøkende til Chronograf, hvor man kan etablere grunnleggende autentiseringsmetoder uten å involvere Chronograf [51].

### 6.1.1 Fellestrekk

Man kan benytte TLS for å sikre all trafikkflyt mellom hver komponent. For å muliggjøre dette kan man velge mellom tre typer sertifikater: “Single-domain”, “Wildcard”, og “Self-signed”. “Single-domain” og “Wildcard” må være utstedt av en CA (Certificate Authority) for å kunne verifisere identiteten til for eksempel en InfluxDB-tjener. Forskjellen mellom disse to er at “Single-domain” er unik per instans, så man må ha ett sertifikat per tjener, mens “Wildcard” betyr at man kan benytte ett sertifikat for alle instanser under samme domene. “Wildcard”-serifikater har en ulempe i at hvis den blir kompromittert vil det påvirke alle som bruker den. Begge type sertifikat kan lett anskaffes fra leverandører som Verisign, Comodo eller LetsEncrypt.

“Self-signed” er selvgenererte sertifikater som kan opprettes på en vilkårlig tjener. Dette betyr i praksis at man ikke kan verifisere identiteten til mottager eller avsender

(med mindre man setter opp en egen PKI), men kan fortsatt kryptere trafikk mellom disse [52]. Konsekvensen av å benytte et selvsignert sertifikat for TICK er at man må ignorere verifisering, slik at avsender og mottager ikke vil prøve å verifisere sertifikatet opp mot en sertifikat-autoritet. Dette setter også krav på at trafikk helst flyter internt i en bedrift og ikke ut på internett. Slike sertifikater bør kun brukes til utvikling og bør unngås i produksjonsmiljø.

Man har også støtte for autentisering av lokalt definerte brukere. Disse opprettes for eksempel i InfluxDB med et sett rettigheter som lese- og skrive-tilgang til en bestemt database. Hvis man har definert en bruker som skal benyttes til skrive-operasjoner vil Telegraf ha behov for å autentisere med disse kredentialene. På samme måte kan man definere en bruker med leserettigheter og la Chronograf autentisere mot InfluxDB med denne brukeren. Kapacitor har også støtte for å definere en lokal bruker som benyttes av klienter for å koble seg til Kapacitor-tjeneren.

Det som er viktig å nevne er trafikkflyten til de forskjellige komponentene. Det er kun Chronograf som typisk trenger å være tilgjengelig til omverden, da dette er den visuelle delen av TICK. Chronograf vil da kommunisere med InfluxDB og Kapacitor på vegne av brukeren, som betyr at disse to tjenestene kan isoleres mest mulig. Telegraf må kunne sende metrikkmålinger til InfluxDB, så dette betyr at man må sørge for streng hvitelisting inn til InfluxDB. TICK i seg selv har ikke støtte for aksesskontroll i form av bestemte IP-adresser og porter, så dette må gjøres av en brannmur som enten en dedikert enhet i front av nettverket eller lokalt på tjeneren.

### 6.1.2 Telegraf

Telegraf byr ikke på noen overraskelser. Hvis man benytter HTTPS/TLS så må man spesifisere hvor sertifikatet kan hentes på noden. Videre hvis man har et selv-signert sertifikat må man hoppe over verifikasjon mot en CA. Ved bruk av brukernavn-/passordautentisering må man legge til disse punktene i konfigurasjonsfilen. Disse kredentialene blir da lagret åpent i en tekstfil, så det er viktig å sette korrekte rettigheter til denne.

### 6.1.3 Kapacitor

Kapacitor skiller seg litt ut fra de andre med at man kan benytte bruker-autentisering inn til tjeneren som kjører Kapacitor og at man kan sikre innkommende og utgående trafikk med TLS. Man vil også trenge et brukernavn og passord for å kommunisere med InfluxDB hvis dette er konfigurert. Dette vil si at hvis Chronograf skal ha tilgang til Kapacitor, så må Chronograf konfigureres med brukernavn-/passordkombinasjon som er satt på Kapacitor.

### 6.1.4 InfluxDB

I InfluxDB kan man finjustere rettigheter til de forskjellige samlingene i databasen. Standardinnstillingen er at det ikke finnes en bruker og må opprettes manuelt når en ny InfluxDB-instans settes opp. En bruker vil være admin eller ordinær bruker, hvor admin har full tilgang til alle interne databaser. Det bør alltid finnes minst én admin-bruker som kun benyttes i krisesituasjoner. Ordinære brukere bør være spesifikke til en oppgave, som for eksempel kun lese-tilgang som benyttes av Chronograf. Til dette oppretter man brukeren og gir leserettigheter til for eksempel Telegraf-databasen. I eksempelkode 6.1 opprettes to brukere, én som skal skrive til Telegraf-databasen, og hvor den andre skal lese.

```
CREATE USER "telewriter" WITH PASSWORD 'somepassword1'
CREATE USER "telereader" WITH PASSWORD 'anotherpassword2'
GRANT WRITE ON "telegraf" TO "telewriter"
GRANT READ ON "telegraf" TO "telereader"
```

Listing 6.1: Opprettelse av spesifikke brukerkontoer i InfluxDB.

Videre kan trafikk sikres ved bruk av sertifikater i likhet med de andre komponentene.

### 6.1.5 Chronograf

Chronograf kobles opp til InfluxDB og Kapacitor ved hjelp av enkel autentisering og trafikken kan sikres ved bruk av sertifikater, hvorvidt Chronograf bruker samme metode for å sikre trafikken selv. Den eneste løsningen som tilbys for autentisering er "Single-Sign-On" (SSO) [53], hvor man utnytter ekisterende autentiseringssystem (OAuth 2.0) hos eksterne etablerte aktører for å håndtere aksesskontroll. Eksempler på slike aktører er GitHub, GitLab, Google, Auth0, samt at man kan bruke generiske uoffisielle aktører.

Et av prinsippene ved SSO er at man kan gjenbruke samme kredentialer hos flere tjenester, slik som Google bruker til sine produkter, og man har i nyere tid sett Facebook åpne opp for denne form for autentisering. Ved hjelp av en god aktør har man muligheten til å danne grupper som brukere tilhører, slik at man kan bestemme på rollebasis hvem som skal ha tilgang til hva. For Chronograf betyr dette at man kan ha en administrerende rolle som har lov til å endre på konfigurasjonen hos Chronograf, mens andre kun har mulighet til å lese.

For å opprette dette trenger man en unik nøkkel til SSO-tjeneren som brukes til å kontrollere rettighetene til en gitt rolle. Med dette kan man fra SSO-tjeneren konfigurere rettighetene på global sikt istedet for hos hver tjeneste. Implementasjonen av dette er ganske rett frem, avhengig av hvilken aktør man vil bruke. I eksempelkode 6.2 blir GitHub brukt. Med denne konfigurasjonen vil man eksempelvis bruke rettigheter og roller som eksisterer hos organisasjonen "ntnusky" hos GitHub.

```
# Set JWT key
export TOKEN_SECRET=Hhi7x...Zqkh
# Client ID from GitHub application
export GH_CLIENT_ID=b86...d104
# Secret from GitHub application
export GH_CLIENT_SECRET=d9d4...9611
# Set optional session duration
export AUTH_DURATION=2h
# Set organizations to use
export GH_ORGS=ntnusky
```

Listing 6.2: SSO-konfigurasjon i Chronograf.

## 6.2 Monolittisk

Fordelen med en monolittisk implementasjon er at alle komponentene ligger på ett og samme sted, som vil si at trafikk mellom Chronograf og de andre komponentene med unntak av Telegraf ikke vil gå ut i nettverket. Ulempen er at hvis man kompromitterer en av tjenestene og får uautorisert tilgang til tjeneren så vil man også få tilgang til de andre tjenestene som kjører. Dette betyr at fokuset må ligge på generell herding av vertsmaskinen og streng tilgangskontroll.

Beste praksis følges, som innebærer å lage en bruker med kun de rettigheten som behøves og som har et sterkt passord. For fjerntilgang benyttes OpenSSH, hvor en spesifikt legger til relevante brukere i en gruppe som heter "sshaccess" og konfigurerer tjenesten til å kun autorisere brukere i denne gruppen. Et annet alternativ til dette er å bruke nøkkelbasert autentisering ved å generere et nøkkelpar der den offentlige nøkkelen blir kopiert over til maskinen, og dermed er det kun de med den private nøkkelen som har tilgang. OpenStack har innebygd støtte for nøkkelbasert autentisering så dette alternativet blir benyttet.

Videre må man kunne filtrere trafikk slik at den potensielle angrepsoverflaten[1] på maskinen blir redusert mest mulig. Her benyttes IPtables da dette leveres med operativsystemet.

Porter Telegraf krever åpnes opp etter behov slik at tjenesten har mulighet til å kommunisere med InfluxDB. Trafikk vil tillates fra utsiden inn til Chronograf. Trafikk mellom Kapacitor og InfluxDB eller Chronograf og InfluxDB vil foregå kun på den lokale tjeneren, så sikring av kommunikasjon er unødig.

## 6.3 Mikrotjenester

Konteinere har revolusjonert hvordan man kjører tjenester, der man i større grad har gått vekk fra å virtualisere maskinvare og kjøre hele operativsystem i virtuelle omgivelser til å heller kjøre et "skall" av et operativsystem på eksisterende systemer. I stedet for å emulere maskinvare for å separere virtuelle maskiner fra det underliggende systemet gjenbruker man heller i konteinere systemer i det underliggende systemet for å isolere hva konteineren har mulighet til å se og gjøre [54]. Dette gjør det langt mindre ressursmessig utfordrende å kjøre en større mengde tjenester på en mindre mengde maskinvare.

Isolering av prosesser, nettverk og filsystem i konteinere kan gi illusjonen av sikkerhet, men dette er kun sant hvis konfigurasjonen av konteiner-løsningen er konfigurert korrekt og med sikkerhet i fokus [55]. Det er fullt mulig å bryte seg ut av konteinere, noe som har blitt demonstrert [56]. Det å kjøre et gitt konteinerbilde fra hvilken som helst kilde har blitt en veldig enkel prosess, men som med all kode må denne kilden verifiseres før man kjører den.

I tilfellet hos TICK baseres alle konteinerbildene på en utgave av Debian uten noen form for ekstra steg tatt for å sikre systemet. De har og alternative bilder basert på Alpine Linux, som er en distibusjon mer rettet for en minimal og sikkert utgangspunkt. Et godt alternativ er å basere bildene på "scratch", altså en distribusjonsløs grunnmur, slik at ved eventuelle hull i programvaren vil ikke en angriper ha tilgang til et shell. Dette gjør det langt mer utfordrende å få videre tilgang til systemet.

Docker oppretter eget virtuelt nettverk på maskinen den kjører på, hvor alle konteinere i samme nettverk vil kunne nå hverandre uten å forlate det virtuelle nettverket. Har man flere tjenester som ikke nødvendigvis skal snakke med hverandre på samme maskin bør det opprettes separate virtuelle nettverk for tjenestene. Trafikkflyt mellom konteinere på flere maskiner i en sverm kommuniserer som normale tjenester etablert rett på maskinen, så man bør sikre denne kommunikasjonen ved hjelp av sertifikater der det er nødvendig.

Majoriteten av konfigurasjonen for hver komponent i TICK gjøres med miljøvariabler,

[1] Angrepsoverflate: Mulige vektorer for sårbarhetsutnyttelse for å få uautorisert tilgang til en maskin

hvor man helst vil utnytte sikre versjoner av dette, avhengig av orkestreringsverktøy for konteinere. Med Docker Swarm er dette en innebygd mulighet i stack-konfigurasjonen, hvor man kan definere de variablene som inneholder sensitiv informasjon som sikre variabler, og heller definere de utenfor konfigurasjons-malene. I Puppet er det mulig å hente slike variabler fra Hiera og utnytte maler, som blir gjort i mikrotjeneste implementasjonen.

# 7 Ytelse

Før man i det hele tatt vurderer å tilgjengeliggjøre en tjeneste er man nødt til å etablere et grunnlag for hvordan den oppfører seg i perioder med høy aktivitet. Dette grunnlaget kan brukes som et referansepunkt når tjenesten blir satt i produksjon. Slik som definert i kapittel 2.5 etablerer man grunnlaget med å utføre ytelsestester som kan være veiledende i å avdekke bristepunkter og øke forståelsen for tjenestens oppførsel.

Produktet av dette kapitlet vil være en sammenligning mellom den distribuerte og monolittiske implementasjonen av TICK for å finne ut hvilken som yter best gitt likt grunnlag.

## 7.1 Metodikk

Det finnes mange former for ytelsestester, de fleste baserer seg på å påføre tjenesten en viss belastning i form av forespørsler. Eksempelvis kan man teste hvor mange hundretalls forespørsler en database kan håndtere før den mettes og forårsaker at forespørsler droppes. Resultatene av slike tester vil benyttes til å danne et grunnlag for tjenestens normale oppførsel og til å utvikle en skaleringsstrategi. I denne delen detaljeres hvilke type tester som benyttes og hvordan man går frem med å utføre disse. I testene vil det fokuseres på InfluxDB, da det er den største komponenten i TICK-stacken.

### 7.1.1 Fremgangsmåter

Ytelsestesting er et bredt område med mange ulike strategier. For å teste InfluxDB er det to typer tester som er mest relevante: stress og spike. Hver test utføres likt mot begge implementasjonene, hvor en implementasjon testes og metrikk sendes til den andre for analyse av resultat. Hver test kjøres fem ganger, og hver gjennomgang kjøres i én klokketime. Dette automatiseres ved hjelp av standard systemverktøy som *cron*, og resultatene blir analysert i etterkant.

#### Stress

Dette er den vanligste formen for ytelsestesting. I en typisk stresstest vil belastning på systemet økes til bristepunktet nås for å se hvor lenge det overlever uten at informasjon forsvinner. Resultatet av dette kan dermed brukes som referanse til skaleringsstrategier.

#### Spike

Spike brukes hovedsakelig for å se hvordan et system oppfører seg når den utsettes for høy belastning ved ujevne tidspunkt. Man øker belastning til godt over hva systemet klarer i svært korte intervaller, og lar dette gå over en viss tidsperiode. Målet er å se om systemet er robust nok til å handtere dramatiske belastningssvingninger.

### 7.1.2 Plattform og verktøy for testing

Da det er to implementasjoner som skal testes må den underliggende infrastrukturen være så homogen som mulig for å sikre pålitelighet av målinger. Dette innebærer likt operativsystem og lik mengde ressurser. Alle virtuelle maskiner baseres på “m1-medium“

ressurspakken på SkyHigh som gir 2 prosessorkjerner, 8GB RAM, 40GB lagring, og alle maskiner kjører CentOS 7.5.

InfluxData har utviklet et eget verktøy for å generere trafikk til InfluxDB som brukes for de to forskjellige testene. Verktøyet er å finne på GitHub[1].

*Influx-stress* kjøres med opsjoner for varighet og mål, og lar den kjøre så hyppig som den klarer. Hele oppsettet for kjøring ligger under vedlegg H. Testen vil generere omlag 200.000 datapunkt per sekund, med ett sekund opphold, så gjennomsnittlig 100.000 spørringer per sekund i løpet en time.

### 7.1.3 Hva skal måles

Når man skal utføre ytelsestesting er man nødt til å definere et sett med metrikker som skal samles inn. Da testene skal utføres mot både monolittisk og distribuert utgave kan disse metrikkene brukes til å sammenligne ytelsen mellom disse utgavene, og man kan dermed gi en anbefaling om hvilke som egner seg best til produksjon. Det legges vekt på å måle prosessor- og minnebruk - de mest typiske metrikkene som er relevant til programvare direkte. Utover dette vil det bemerkes at med den mengden trafikk som genereres fra testing-verktøyet vil det også være en god del nettverkstrafikk og aktivitet på disk, men dette er derimot ikke et fokuspunkt i denne oppgaven.

### 7.1.4 Forventninger

Da begge implementasjonene kjører så likt som overhodet mulig, og de binære filene for programvaren er like, forutses det at den konteineriserte utgaven vil kreve litt mer ressurser enn den monolittiske. Dette baseres på at konteinere krever et ekstra abstraksjonslag over det virtualiserte miljøet, i motsetning til den monolittiske utgaven som kjører uten denne abstraksjonen.

## 7.2 Tester og resultat

Det ble valgt en generell fremgangsmåte som kan brukes på begge utgavene. Resultatet av hver test lagres unna i den andre utgaven. Når testene er gjennomført vil resultatet av tilsvarende tester på begge utgaver bli analysert og sammenlignet. Tidsrommet for testingen lagres unna i en loggfil, som kan brukes i spørringer for å hente ut metrikken med høy nøyaktighet.

Etter at en test er gjennomført hentes ut gjennomsnittlig bruk i løpet av hver hele time hvor testen har kjørt. Tidspunktet er i UTC tidssone, så det kompenseres med å trekke fra to timer i spørringen.

```
SELECT mean(usage_user) FROM telegraf..cpu
    WHERE (time >= '2019-04-24T10:00:00Z'
          AND time <= '2019-04-24T10:59:59Z')
          AND host='worker-0.lab'
    GROUP BY time(1h)
```

Listing 7.1: Eksempelvis spørring av metrikkdata fra tester.

Med spørringen i eksempelkode 7.1 finner man gjennomsnittet for en gitt test, som kjøres likt for all metrikk fra testene med hver deres tidsintervall. Gjennomsnittsmålinger gir et godt overblikk om hvordan testen kjørte, men gir ikke svar på om det har vært

[1] https://github.com/influxdata/influx-stress

kortere perioder med høyere eller lavere ressursbruk innad et aktuelt tidsintervall. For å kompensere for dette samles alle datapunktene inn og grafer dannes for å gi et mer detaljert innsyn i testens hele løp.

### 7.2.1 Monolittisk

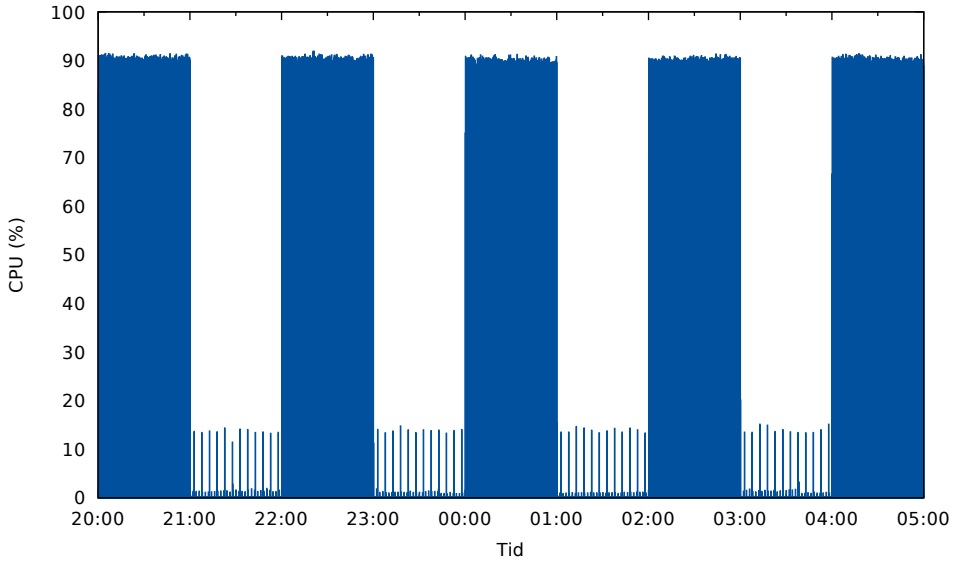
Første runde med testing endte med at det ikke ble skrevet noe data til databasen, og dermed ble det ikke analysert resultatet av dette. Videre etterforskning viste at influx-stress opprettet en egen database hos InfluxDB-instansen der den skulle teste, som tok opp alt for mye plass. For å fikse dette ble retention policy justert, som har en standardverdi på syv dager. Etter å ha justert denne ned til én time var det ikke lengre plassmangel, uten at det gjorde noen forskjell på testresultatene. I tabell 1 og 2 ser man resultatet for alle fem gjennomkjøringene av hver type test, altså stress og spike, med gjennomsnittlige verdier og verdier når systemet står stille, samt grafisk fremstilt i figur 7, 8, 9 og 10.

| Run | CPU (%) | Memory (MB) |
|---|---|---|
| 1 | 89.910 | 811 |
| 2 | 89.833 | 841 |
| 3 | 89.668 | 684 |
| 4 | 89.809 | 810 |
| 5 | 89.843 | 781 |
| Avg | 89.812 | 785 |
| Idle | ~1 | 536 |

Tabell 1: Resultat av stresstest monolittisk.

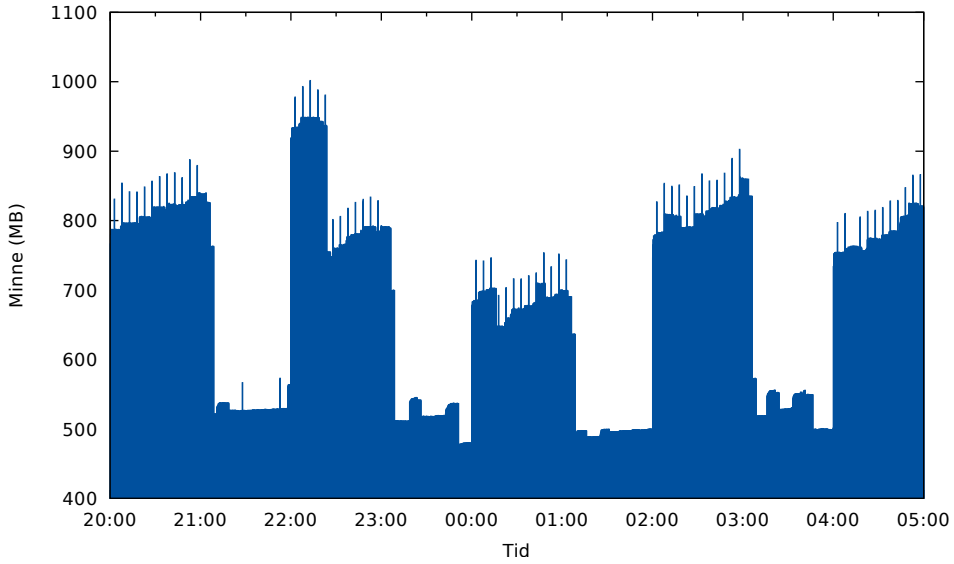
| Run | CPU (%) | Memory (MB) |
|---|---|---|
| 1 | 64.059 | 692 |
| 2 | 65.761 | 832 |
| 3 | 63.991 | 823 |
| 4 | 64.307 | 955 |
| 5 | 64.406 | 742 |
| Avg | 64.504 | 808 |
| Idle | ~1 | 536 |

Tabell 2: Resultat av spike-test monolittisk.

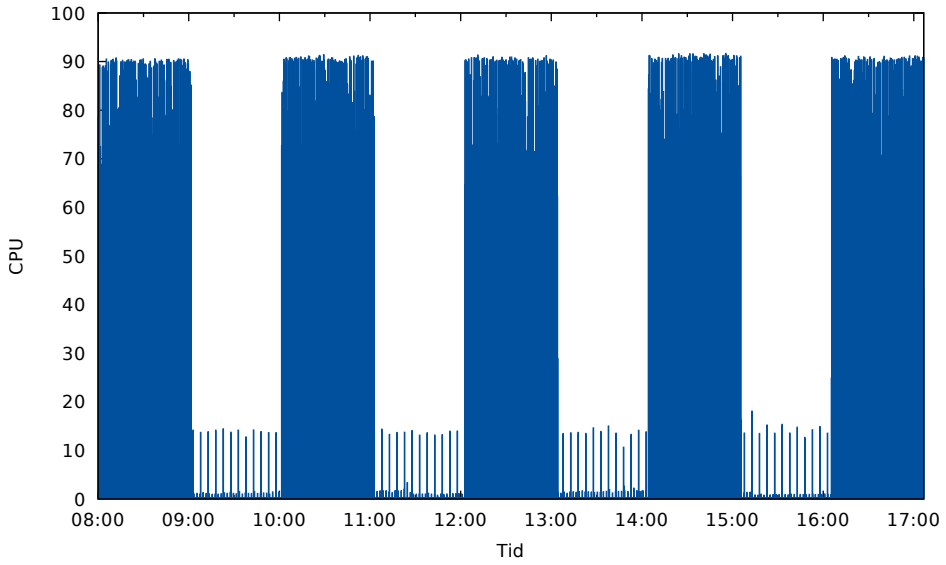



Figur 7: Monolittisk stresstest: CPU-bruk.

Prosessorbruk ligger konstant på rundt 90% gjennom hele stresstesten uten noe variasjon, mens minnebruken forteller en annen historie. Den varierer veldig mellom 150 og 500 megabyte i bruk (idle minus total), og har en tendens til å øke gjennom testens levetid. Det samme vises i spike-testen, bare med en prosessorbruk som svinger veldig, noe som er forventet med denne type test. Dette viser at InfluxDB muligens har problemer



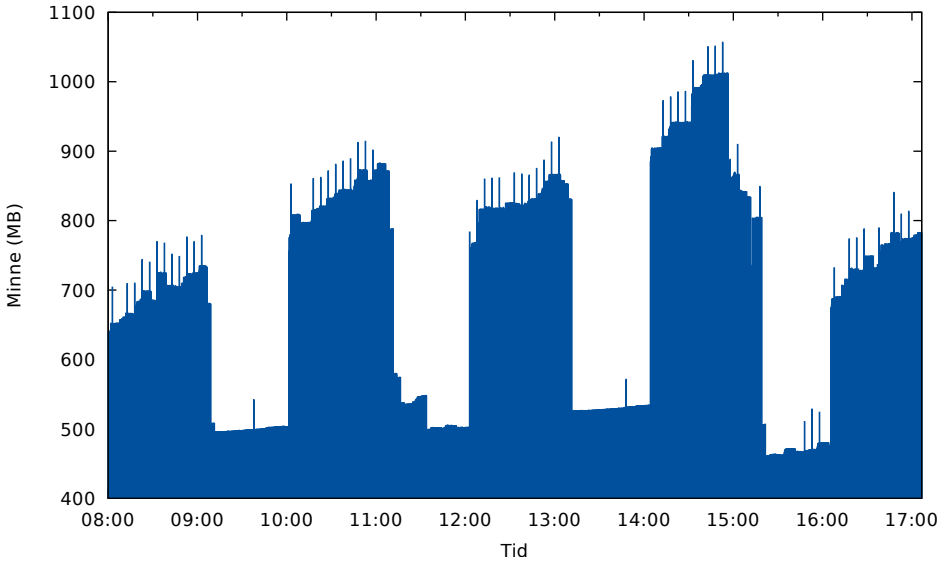

Figur 8: Monolittisk stresstest: minnebruk.




Figur 9: Monolittisk spiketest: CPU-bruk.

med minnekontroll over konstant press over lengre tid. Minnehåndtering kan utbedres ved å optimalisere konfigurasjonen, noe som ikke ble prioritert.

Utover dette klarte InfluxDB å henge med gjennom hele testløpet uten å sprenge ressursbruken eller miste data, noe som viser til kvaliteten i produktet. En monolittisk implementasjon vil egne seg godt til forskjellige mengder bruk, selv med én enkelt database, hvor den håndterer høyt, kontinuerlig volum og ujevne rykk og hopp i innkommende trafikk uten problem.




Figur 10: Monolittisk spiketest: minnebruk.

### 7.2.2 Mikrotjeneste

Etter første gjennomkjøring av stresstester ble det lagt merke til at minnebruken er nesten tre ganger så høy i forhold til den monolittiske utgaven. Problemet her var at testene ikke var konfigurert likt og hadde forskjellig retention policy på stress-dataen. Ved justering slik at de var like ble resultatet mer som forventet, altså at oppførselen var en noe mer lik som monolittisk.

I tabell 3 og 4 ser man resultatet for alle fem gjennomkjøringene av hver type test, altså stress og spike, med gjennomsnittlige verdier og verdier når systemet står stille, samt grafisk fremstilt i figur 11, 12, 13 og 14.

| Run | CPU (%) | Memory (MB) |
|---|---|---|
| 1 | 91.358 | 918 |
| 2 | 89.802 | 951 |
| 3 | 91.223 | 956 |
| 4 | 91.326 | 991 |
| 5 | 91.496 | 895 |
| Avg | 91.040 | 942 |
| Min | ~1 | 581 |

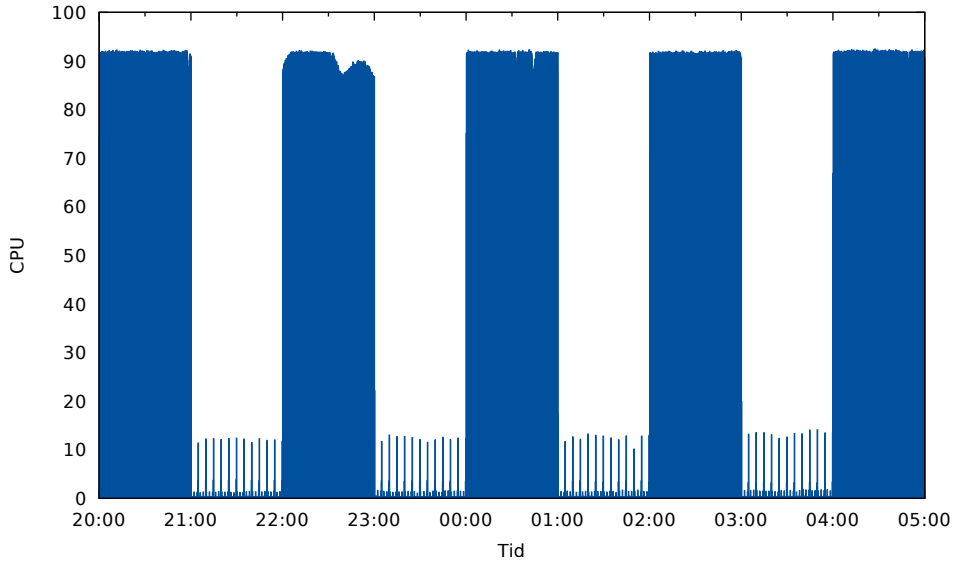
Tabell 3: Resultat av stresstest mikrotjeneste.

| Run | CPU (%) | Memory (MB) |
|---|---|---|
| 1 | 64.853 | 914 |
| 2 | 66.058 | 952 |
| 3 | 64.842 | 1003 |
| 4 | 65.606 | 1061 |
| 5 | 66.503 | 1135 |
| Avg | 65.572 | 1013 |
| Min | ~1 | 581 |

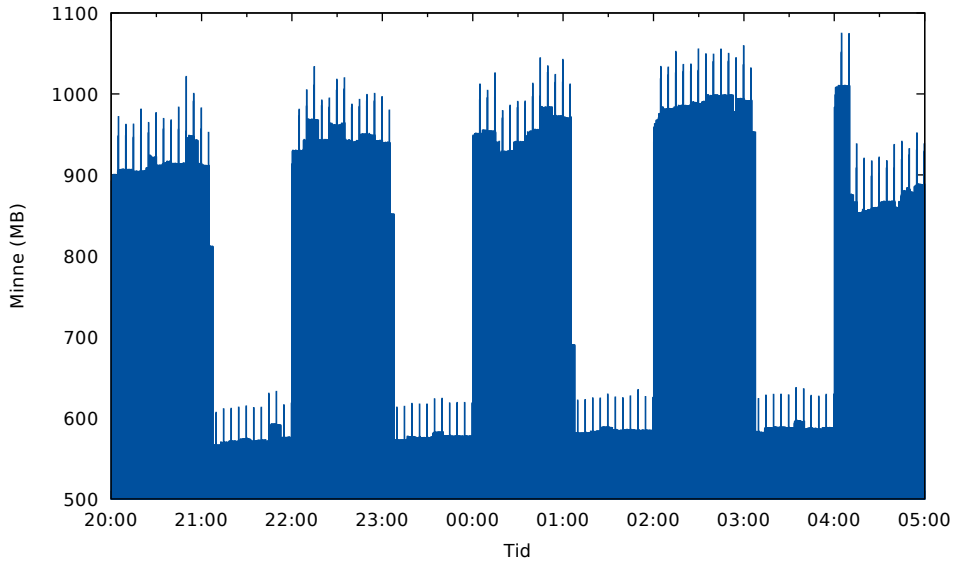
Tabell 4: Resultat av spike-test mikrotjeneste.

Prosessorbruk ligger konstant på rundt 90% gjennom hele stresstesten, og minnebruk ligger også ganske konstant. Et mulig problem her er at minnebruken mellom hver test blir noe høyere for hver kjøring, som kan vise til at konteineren eller programvaren ikke får tømt minnet korrekt. Det er en variasjon på 10-50MB for hver kjøring, så dette er ikke et problem en korrekt policy i orkesteringsverktøyet kan kontrollere.

I spike-testen vises en svingende prosessorbruk, som forventet med denne type test, men minnebruken viser igjen samme “problem” som i stresstesten hvor den øker for hver kjøring. Utover dette klarte InfluxDB i konteinere å henge med gjennom hele testløpet




Figur 11: Mikrotjeneste stresstest: CPU-bruk.

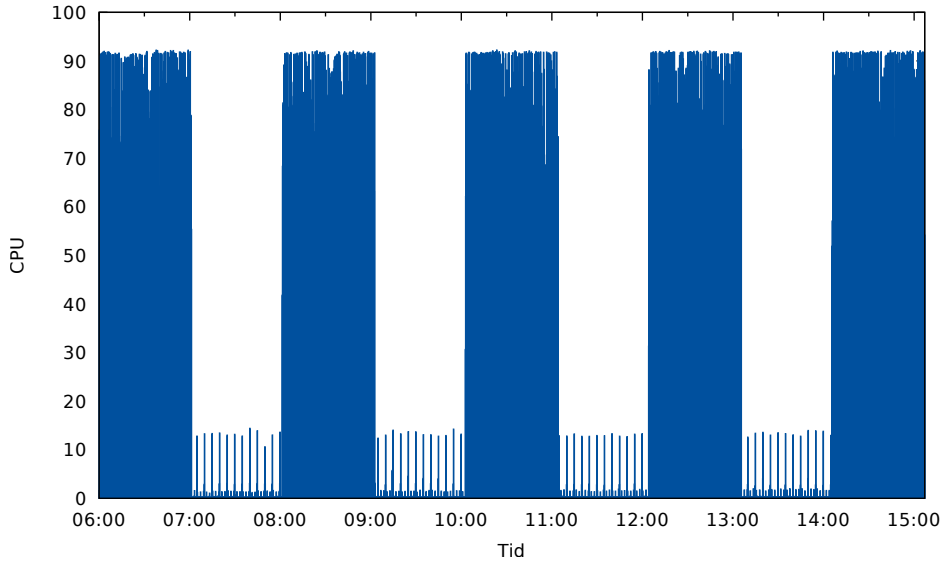



Figur 12: Mikrotjeneste stresstest: minnebruk.

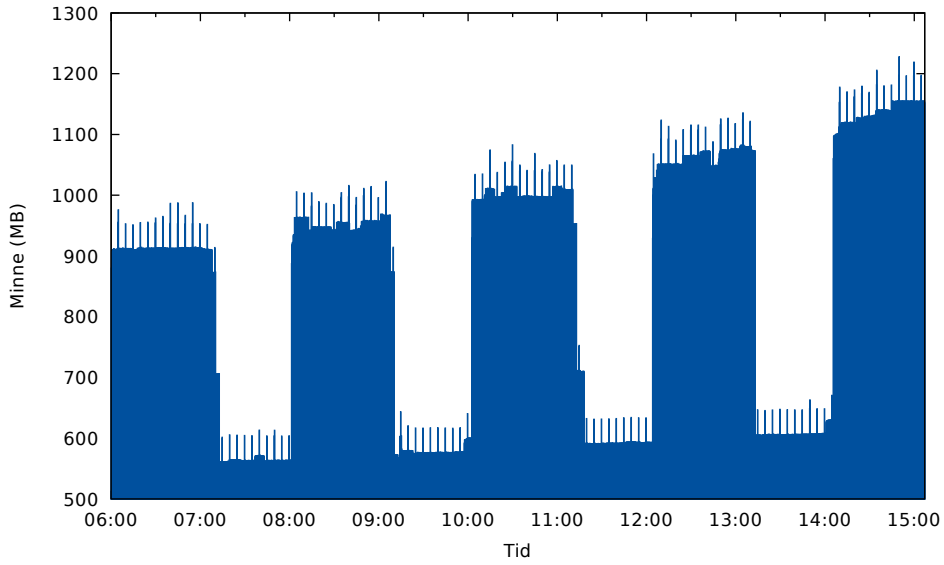
uten å sprenge ressursbruken eller miste data.

## 7.3 Sammenligning

Når man sammenligner stresstestene av de to utgavene med hverandre ser man at prosessorbruk ligger likt mellom testene. Man ser at minnebruken ved mikrotjeneste er gjennomsnittlig høyere, men mer stabil gjennom testenes løp, mens den monolittiske versjonen har en del større rykk i minnebruk. For å se om det er noe mønster til minnebruken kjøres stresstesten i en lengre periode, med resultat etter 24 timer vist i figur 15. Her vises et mønster hvor minnebruken går gradvis oppover i opptil 10 timer hvor det deretter



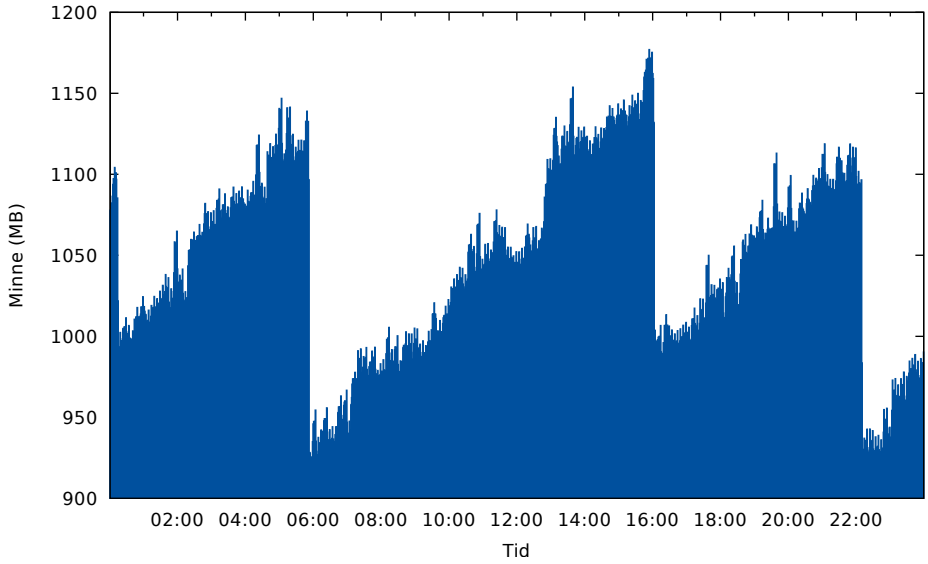

Figur 13: Mikrotjeneste spiketest: CPU-bruk.




Figur 14: Mikrotjeneste spiketest: minnebruk.

er en drastisk reduksjon av minnebruk. Uten å vite nøyaktig hva som foregår spekuleres det at metrikkdata blir mellomlagret i minnet, da det er en langt raskere lagringsplass enn disk og vil bidra til økt ytelse i InfluxDB. Det man kan ta med seg videre under planlegging av ressurskrav er at man bør provisjonere litt ekstra minne.

Spiketestene viser relativt likt resultat for prosessorbruk, hvor mikrotjenesteimplementasjonen bruker omlag 5% mindre jevnt over. Den monolittiske utgaven er mer aggressiv på minnebruk for denne typen test, og viser igjen at bruken trender oppover mellom hver gjennomkjøring. Mikrotjeneste bruker fortsatt mer minne i løpet av en test, men den er mye mer stabil uten store rykk. Her ser man også at likt stresstesten trender




Figur 15: Mikrotjeneste minnebruk under stresstest over 24 timer.

minnebruken oppover etterhvert som de enkelte gjennomkjøringene av spike utføres.

En annen metrikk som har blitt sett på er nettverkstrafikk inn og ut. I løpet av stresstesten ble tjenerens gigabit-grensesnitt presset til ytterpunktet, og en figur av dette er ikke spesielt interessant. Resultatet tyder på at man bør provisjonere litt ekstra båndbredde til tjeneren som holder InfluxDB, som kan gjøres med å aggregere to nettverksgrensesnitt på hver tjener.

Alt i alt viser testene at forskjellene mellom implementasjonen er små, med litt svinginger i minnebruk mellom de. Den største forskjellen er heller i implementasjonen selv, hvorvidt man foretrekker konteinerløsninger eller ikke. En annen faktor som kan tas med i vurderingen er om man vil ha offisiell støtte for provisjonering eller ei, noe InfluxData leverer for Docker men ikke for Puppet.

# 8 Konklusjon

I denne rapporten vises hvordan man kan implementere TICK-stacken på to forskjellige måter, hvor begge er implementert fullt eller delvis med Puppet. Eksempeloppsettet viser hvordan man bruker den nyutviklede modulen for implementasjon av et monolittisk oppsett, samt hvordan man kan implementere stacken på en distribuert måte ved bruk av offisielle konteinerbilder. Rapporten viser hvordan man kan og bør sikre de forskjellige komponentene i stacken, og hvordan de yter under høyt trykk.

Prosjektgruppen har utviklet en Puppet-modul som vil sette opp TICK-stacken i sin helhet eller delvis, med standardverdier slik InfluxData har definert. En komplett implementasjon av begge utgavene er satt opp i en felles infrastruktur, hvor all relevant konfigurasjon er detaljert. Omfattende ytelsestesting har blitt gjennomført på begge utgavene, og har blitt sammenlignet mellom disse. TICK-stacken har vist god ytelse for begge implementasjonene, og prosjektgruppen har ingen problem med å anbefale hverken av de.

Basert på funn i rapporten vil prosjektgruppen anbefale å implementere TICK i den utgaven som best passer formålet til den som tar stacken i bruk. Om man har mindre trafikk eller lite krav til redundans vil den monolittiske fungere godt, hvor den distribuerte har offisiell støtte både for enkelt oppsett med eller uten redundans. Sikkerhet i TICK med bruk av beste praksis vil dekke de fleste krav satt av moderne bedrifter.

## 8.1 Videre arbeid

Prosjektgruppen fant flere alternative løsninger og muligheter til å utvide prosjektet, men grunnet begrenset tid ble de ikke oppfulgt. Her er noen eksempler på ting som man gjerne vil ta en titt på videre.

**Utbedring av Puppet-modul**

Modulen gjenbruker i dag mye kode i flere klasser som utfører samme type oppgave som kunne ha vært forbedret med bruk av mal-klasser og muligens delte konfigurasjonsmaler. Utviklingen kunne vært mer strømlinjeformet med automatisert testing. Dokumentasjon er noe svakere enn hva som hadde vært foretrukket. En fullverdig og godt dokumentert modul vil kunne benyttes av andre, så en offisiell lansering av modulen på Puppet Forge er en mulighet. Til slutt kunne man ha vurdert å dele modulen opp i fire nye moduler og skrevet om til å ha én for hver komponent.

**Vurdering av bedriftsversjoner**

Skalering er et veldig populært tema i fagmiljøet, hvor InfluxData har to mulige løsninger. InfluxCloud som opereres av InfluxData selv, samt InfluxEnterprise som kan implementeres på egen infrastruktur. Skalering har stor betydning på ytelse, og har egne kriterier med tanke på sikkerhet, så man vil da ta en helhetlig vurdering av disse.

### Utdype sikkerhetsvurdering

Sikkerhetsvurderingen hadde fokus på grunnleggende sikkerhet i både infrastrukturen og ved bruk av TICK-stacken. Man vil gjerne se videre på programvaresikkerhet i komponentene som danner TICK, da det viste seg at det var en mangel på tilgjengelig informasjon om dette.

### Videre ytelsestesting

Ytelsestestingen utført av prosjektgruppen var god nok til å danne et grunnlag for vurdering av ytelsen i TICK, men man vil gjerne se videre på annen statistikk som danner et det større bildet. Dette innebærer eksempelvis å vurdere nettverkstrafikk, diskytelse, samt svar-tid på forespørsler.

### Vurdering av alternative komponenter og verktøy

Implementasjonen kan utføres med vidt forskjellige orkestrering- og provisjoneringsverktøy, hvor enkelte kan ha bedre støtte for TICK-stacken, slik som Docker. Alternative komponenter til TICK er det mulig å se videre på, da mange i fagmiljøet har satt pris på at TICK er modulært og komponenter kan både erstattes og utvides etter behov, slik som Grafana[1] som visualiseringsverktøy og Alerta[2] som varslingssystem.

### Dypdykk i Kapacitor

Kapacitor er et stort, omfattende og kraftig verktøy, med langt flere muligheter enn det prosjektgruppen fikk tid til å se på. Man vil gjerne ta et dypere dykk i hva Kapacitor har å tilby, og hvordan den kan brukes til å utbedre monitorering i form av sikkerhet, aggregering og automasjon.

## 8.2 Evaluering av gruppen

### Organisering

Begge gruppemedlemmene er fornøyde med organiseringen av prosjektet gjennom dens løp. Møter med gruppemedlemmer har vært konsistente og har oppnådd dens ønskede effekt, og veileder har vært svært behjelpelig gjennom hele perioden. Periodevis kunne gruppen ha vært mer effektive, men det har ellers vært god kontinuitet.

### Arbeidsfordeling

I arbeidet med implementasjon ble de to utgavene delt opp mellom gruppemedlemmene, noe som fungerte bra. Ved enden av denne prosjektfasen beskrev hvert gruppemedlem sine synspunkter og erfaringer med deres implementasjonen og arbeidet rundt, slik at gruppen i sin helhet kunne komme med tilbakemeldinger på hverandres arbeid. Ellers har gruppen jobbet tett opp mot hverandre.

### Måloppnåelse

I henhold til prosjektets mål har alle blitt nådd slik forventet, men gruppen gikk glipp av muligheter til å kunne utvide oppgaven videre. Gruppen hadde spesielt lyst til å ta i bruk InfluxEnterprise og detaljere skalering av TICK.

[1] https://grafana.com/
[2] https://alerta.io/

#### Tidsbruk og fremdriftsplan

Den estimerte tidsplanen har vært veldig nyttig til å bestemme hvor mye tid som kunne gå med på hver enkelt del av prosjektet, og denne har blitt fulgt godt gjennom mesteparten av prosjektets løp. I løpet av ytelseskapitlet endte gruppen opp med å ligge noe etter planen grunnet uforutsette hendelser og for høy tillit til ukjent programvare. Det var planlagt ekstra tid til hver enkelt del i prosjektet fra prosjektplanleggingfasen, så dette endte opp med å ikke bli et stort problem i det lengre løp.

Det ble opprinnelig planlagt å bruke Kanban som arbeidsmetodikk, men tidlig i prosjektet viste det seg at det var mer logisk å følge en godt strukturert fossefallsmodell i stedet, som passet godt med den originale tidsplanen.

#### Læringsutbytte

Gruppemedlemmene er fornøyde med læringsutbyttet fra prosjektet. Erfaring med større konfigurasjoner i Puppet og utvikling av moduler, samt orkestrering med Docker har utfordret gruppen til å yte sitt beste. Ingen av gruppemedlemmene hadde tatt i bruk monitoreringsverktøy tidligere, og erfaringen med både implementasjon og bruk av dette har vært svært lærerikt. Arbeid med et større akademisk prosjekt over en lengre periode som arbeidsform viste seg å være den største utfordringen.

## 8.3 Avsluttende bemerkninger

Hvilken implementasjon av TICK er best for Basefarm? Det avhenger litt om hvor man vil bruke stacken, og hvordan. Da Basefarm allerede er godt kjent med og bruker Puppet til deres infrastruktur vil bruk av TICK rett på maskinvare eller virtualisert være en god løsning. Man har mulighet til å skalere ved bruk av InfluxEnterprise selv uten konteinere, men modulen utviklet i sammenheng med prosjektet støtter ikke dette.

Med den distribuerte løsningen har man en kortere vei til å sette opp TICK fra bunn, samt at det vil være lettere å skalere om man har konteinerbasert infrastruktur opprettet. Den monolittiske utgaven er lettere å sikre, da det er færre komponenter som må snakke med hverandre over nettverk i motsetning til lokalt. Begge utgavene viser høy ytelse med marginale forskjeller.

Gruppen er fornøyd med resultatene fra prosjektet og håper implementasjonseksempelet, Puppet-modulen, samt funn fra utforskning av TICK i praktisk bruk er tilstrekkelig for å vurdere løsningen. Hvilken utgave man vil gå for må baseres ut i fra hva som passer best til eget formål, eksisterende infrastruktur og kunnskap.

# Bibliografi


[1] Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. 2017. Service monitoring. *The Practice of System and Network Administration*, 3rd ed, 671.

[2] Basefarm. Om oss. https://basefarm.no/om-oss/.

[3] Morris, K. 2016. Monitoring: Alerting, metrics and logging. *Infrastructure As Code*, 3rd ed, 87.

[4] Morris, K. 2016. Metrics: Collect and analyze data. *Infrastructure As Code*, 3rd ed, 89.

[5] Opsview. IT monitoring and automation. URL: https://www.opsview.com/resources/automation/whitepapers/it-monitoring-and-automation.

[6] Data point. *Unit of observation*. URL: https://en.wikipedia.org/wiki/Data_point.

[7] Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. 2014. Monitoring architecture and practice. *The Practice of Cloud System and Administration*, 3rd ed, vol 2, 346.

[8] Time series. URL: https://en.wikipedia.org/wiki/Time_series.

[9] InfluxData. Time series databases vs. other databases. *Time Series Database Explained*. URL: https://www.influxdata.com/time-series-database/.

[10] InfluxData. Time series database. URL: https://www.influxdata.com/time-series-database/.

[11] InfluxData. Time series database explained. URL: https://www.influxdata.com/time-series-database/.

[12] DB-Engines. DB-Engines ranking of time series DBMS. https://db-engines.com/en/ranking/time+series+dbms.

[13] InfluxData. Telegraf documentation. https://docs.influxdata.com/telegraf/v1.10/.

[14] InfluxData. 2018. Write your own Telegraf plugin. https://www.slideshare.net/influxdata/write-your-own-telegraf-plugin.

[15] InfluxData. InfluxDB documentation. https://docs.influxdata.com/influxdb/v1.7/.

[16] Downsampling and data retention. https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/.

[17] InfluxData. Chronograf documentation. https://docs.influxdata.com/chronograf/v1.7/.

[18] InfluxData. Chronograf OAuth providers. https://docs.influxdata.com/chronograf/v1.7/administration/managing-security/#oauth-2-0-providers.

[19] InfluxData. Kapacitor documentation. https://docs.influxdata.com/kapacitor/v1.5/.

[20] InfluxData. TICKscript specification. https://docs.influxdata.com/kapacitor/v1.5/reference/spec/.

[21] Martinsen, R. 2017. Exploring the TICK stack. https://rudimartinsen.com/2017/12/22/exploring-the-tick-stack/.

[22] Martinsen, R. 2017. Some InfluxDB gotcha's. https://rudimartinsen.com/2017/11/07/some-influxdb-gotchas/.

[23] Martinsen, R. 2017. vSphere performance data - part 4 - InfluxDB. https://rudimartinsen.com/2017/07/13/vsphere-performance-data-part-4-influxdb/.

[24] Arbezzano, G. 2017. Infrastructure monitoring with tick stack. https://blog.codeship.com/infrastructure-monitoring-with-tick-stack/.

[25] Docker. What is a Container. URL: https://www.docker.com/resources/what-container.

[26] Morris, K. 2016. Orchestrating processes with server roles. *Infrastructure as Code*, 1st ed, 92.

[27] Morris, K. 2016. Choosing tools for infrastructure as code. *Infastructure as Code*, 1st ed, 42–48.

[28] Morris, K. 2016. Server configuration tools. *Infastructure as Code*, 1st ed, 61–65.

[29] Morris, K. 2016. Immutable infrastructure. *Infastructure as Code*, 1st ed, 70.

[30] Puppet. Puppet documentation. https://puppet.com/docs/puppet/6.4/puppet_index.html.

[31] Puppet. Hiera. URL: https://puppet.com/docs/puppet/5.3/hiera_intro.html.

[32] Puppet architecture. https://puppet.com/docs/puppet/6.4/architecture.html.

[33] Lund-Lange, A. 2019. tick-stack-conf. https://github.com/p3lim/tick-stack-conf.

[34] Morris, K. 2016. Antipattern: Monolithic stack. *Infastructure as Code*, 1st ed, 158.

[35] Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. 2014. Service oriented architecture. *The Practice of Cloud System and Administration*, 3rd ed, vol 2, 90–92.

[36] Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. 2014. Performance and scaling. *The Practice of Cloud System and Administration*, 3rd ed, vol 2, 326–328.

[37] Kurose, J. F. & Ross, K. W. 2016. Operational security: Firewalls and intrusion detection systems. *Computer Networking A Top-Down Approach*, 7th ed, 679–687.

[38] Limoncelli, T. A., Hogan, C. J., & Chalup, S. R. 2014. Authentication. *The Practice of Cloud System and Administration*, 3rd ed, vol 2, 719–720.

[39] Dierks, T. & Rescorla, E. 2008. The Transport Layer Security (TLS) protocol. URL: https://tools.ietf.org/html/rfc5246.

[40] SSH Communications Security, I. PKI - Public Key Infrastructure. URL: https://www.ssh.com/pki/.

[41] Morris, K. 2016. Principles of infrastructure as code. *Infrastructure as Code*, 1st ed, 10–13.

[42] InfluxData. Retention policy. https://docs.influxdata.com/influxdb/v1.7/concepts/glossary/#retention-policy-rp.

[43] Puppet. Beginner's guide to writing modules. https://puppet.com/docs/puppet/6.4/bgtm.html.

[44] InfluxData. Configuring Telegraf - environment variables. https://docs.influxdata.com/telegraf/v1.10/administration/configuration/#environment-variables.

[45] InfluxData. Configuring InfluxDB - environment variables. https://docs.influxdata.com/influxdb/v1.7/administration/config/#influxdb-environment-variables-influxdb.

[46] InfluxData. Chronograf configuration options - usage. https://docs.influxdata.com/chronograf/v1.7/administration/config-options/#usage.

[47] InfluxData. Kapacitor environmental variables. https://docs.influxdata.com/kapacitor/v1.5/administration/configuration/#kapacitor-environment-variables.

[48] Lowe, S. 2015. Using an ssh bastion host. URL: https://blog.scottlowe.org/2015/11/21/using-ssh-bastion-host/.

[49] Wiki, A. L. IPtables. https://wiki.archlinux.org/index.php/Iptables.

[50] Kalsin, V. 2017. How to monitor system metrics with the TICK stack on CentOS 7. https://www.digitalocean.com/community/tutorials/how-to-monitor-system-metrics-with-the-tick-stack-on-centos-7.

[51] Inc, N. Restricting access with HTTP Basic Authentication. https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication/.

[52] Desk, S. S. What are self-signed certificates and disadvantages. URL: https://www.sslsupportdesk.com/what-are-self-signed-certificates-and-disadvantages.

[53] Auth0. What is and how does single sign-on authentication work? URL: https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/.

[54] Morris, K. 2016. Containers. *Infastructure as Code*, 1st ed, 70–79.

[55] Docker. Docker security. URL: https://docs.docker.com/engine/security/security/.

[56] Twistlock. Breaking out of Docker via runC - explaining CVE-2019-5736. URL: https://www.twistlock.com/labs-blog/breaking-docker-via-runc-explaining-cve-2019-5736.

# A Oppgavebeskrivelse

Forslag til bacheloroppgave NTNU:

# TICK-stack

Oppdragsgiver: Basefarm v/Patrik S. Olsen, patrik.storm.olsen@basefarm.com

## Om Basefarm

Basefarm har erfaring fra 18 år med sikre og stabile leveranser av virksomhetskritisk IT-drift. Samtidig har vi et innovasjonsmiljø for alt det nye som skjer innenfor alt fra skytjenester til digital kundeopplevelser. Eksempler på applikasjonsdrift er databaser, web-servere, CMS-applikasjoner og portaler basert på forskjellige typer skybasert infrastruktur. Vi sikrer levering av de mest kritiske IT-tjenester bedriftene er avhengig av gjennom vår ISO27001 sertifisering, PCI-DSS godkjennelse og et eget Security Incident Response team som alltid er tilgjengelig.

## Oppgaven

Vi vurderer å benytte TICK-stack i større grad enn idag til innsamling av data fra enheter (containere, VMer, fysiske datamaskiner) i nettverkene våre. TICK-stack består av

- Telegraf: agent som samler inn datapunkt
- InfluxDB: database som lagrer tidsserie-data
- Chronograf: visualiserer og produserer grafer
- Kapacitor: alarmer og avviksdeteksjon i tidsserie-data

Vi ønsker i denne oppgaven at en gruppe på 2-3 studenter

- Samler inn og sammenstiller det som finnes online av erfaringsinformasjon fra de som har tatt i bruk TICK-stack
- Designer og implementerer en medium-skala monolittisk installasjon ved bruk av vårt konfigurasjonsstyringssystem (Puppet)
- Designer og implementerer en medium-skala mikrotjeneste-basert installasjon ved delvis bruk av vårt konfigurasjonsstyringssystem (Puppet)
- Utreder sikkerhet og ytelse for TICK-stack generelt, og for de to installasjonene spesielt

# B Prosjektplan

# 1. Mål og Rammer

## 1.1 Bakgrunn

Dette prosjektet er en avsluttende rapport som en del av bachelorprogrammet IT-Drift og Informasjonssikkerhet ved NTNU Gjøvik, der studentgrupper ved universitetet utfører en større oppgave for NTNU internt eller for en ekstern bedrift, hvor studentene får mulighet til å demonstrere kunnskap lært i løpet av deres studietid.

Gruppen, bestående av Adrian L Lange og Vetle T Moen, har fått tildelt en oppgave fra IT-selskapet Basefarm AS om å utføre et prosjekt for dem. Prosjektet går ut på å implementere, ytelses- og sikkerhetsvurdere et monitoreringssystem mest kjent som "TICK-stack", en komposisjon av fire programvarer utviklet av InfluxData, spesifikt lagd for overvåking og varsling for systemer og tjenester i drift. Stacken består av følgende programvare:

- **T**elegraf, en metrikk-samlende agent som kjører på hver maskin/node
- **I**nfluxDB, en database som tar imot all data fra samtlige agenter
- **C**hronograf, et visualiseringsverktøy for alt av data
- **K**apacitor, et varslingssystem for spesifisering av data

Basefarm AS er en europeisk driftsleverandør av virksomhetskritiske IT-løsninger. Med kontorer i flere europeiske land leverer de drift og håndtering av private og offentlige skytjenester fra egne datasentre i Norge, samt at de leverer rene datasentertjenester etter behov. Eksempelvis så drifter Basefarm servere og skytjenester for Norwegian, Flytoget og Lånekassen, hvor de har dedikerte team til enkelte.

De har i løpet av fjoråret gått over til å ta i bruk TICK-stack, hvor de i hovedsak bruker det til visualisering og integrering med deres alarm-system ved hjelp av to andre komponenter, derav Grafana (et alternativ til Chronograf og Kapacitor) og Alerta til å visualisere alarmer. Dette lar de bli varslet om eventuelle feil og avvik i systemene de er ansvarlige for, og gir de kraften til å sile ut overflødig informasjon og heller fokusere på det som er kritisk. De ønsker nå å benytte TICK-stack i større grad til innsamling av data fra deres infrastruktur.

## 1.2 Prosjektmål

Prosjektmålene er delt inn i tre kategorier: resultatmål, effektmål og læringsmål. Resultatmål representerer det som skal ferdigstilles ved prosjektets slutt. Effektmålene representerer det vi ønsker å se resultatet føre til. Læringsmålene representerer det vi som gruppe / studenter ønsker å lære av prosjektet.

### 1.2.1 Resultatmål

Det forventes at sluttrapporten skal inneholde følgende punkter:

- Demonstrasjon av forskjellige metoder for hvordan TICK kan implementeres.
    - En utgave implementert i en monolittisk arkitektur, og en i mikrotjeneste-arkitektur.
- En sikkerhetsevaluering av TICK-stack i begge utgaver.
    - Hvordan informasjon lagres, behandles, og sendes.

    - Hvilke anbefalte fremgangsmåter som finnes for TICK, og om disse er fornuftige.
- En ytelsestest og sammenligning av de forskjellige utgavene.
    - Hvordan de forskjellige utgavene skalerer.

### 1.2.2 Effektmål

- Det er ønskelig at sluttrapporten fungerer som et støttende dokument til fremtidig bruk av TICK.
- Bygge en konfigurasjon hvor den som leser kan basere, og bygge på, sin egen implementasjon.
- En sammenligning mellom forskjellige metoder for implementasjoner/utgaver av TICK-stack som kan fungere som et støttedokument til fremtidige beslutninger.
- Få økt ytelse- og sikkerhetsforståelse for TICK-stack og spesifikt forskjellene mellom de implementasjonene fra prosjektoppgaven.

### 1.2.3 Læringsmål

- Øke forståelsen av metrikk, loggføring, og overvåking i større systemer.
- Lære hvordan konfigurasjonsstyringssystemet Puppet fungerer med mikrotjenester.
- Lære å bruke en utviklingsmodell der utvikling ikke er sentralt som oppgave.
- Få innsikt i hvordan reelle bedrifter tar i bruk monitoreringssystemer.

## 1.3 Rammer

Denne delen beskriver hvilke rammer vi skal holde oss innenfor.

### Økonomiske Rammer

Reisekostnader vil bli dekket av oppdragsgiver i henhold til prosjektavtalen. Ressurser i form av datakraft (CPU, minne, lagring) er tilgjengelig via NTNU Gjøvik sin OpenStack implementasjon; SkyHigh. Hvis gruppen ender opp med å evaluere Enterprise- og/eller Cloud-versjonen av TICK-stack så vil en eventuell kostnad bli dekt av Basefarm. Reisekostnader vil og bli dekt av Basefarm, per prosjektavtalen. Det forventes ingen kostnader utover dette.

### Tidsmessige Rammer

- Prosjektet har oppstart 9. Januar, 2019.
- Prosjektplanen må leveres senest 1. februar, 2019.
- Hoveddokumentet skal leveres innen 20. mai, 2019.

### Kravspesifikke Rammer

- For konfigurasjonsstyring så skal det benyttes Puppet.
- Det er ikke nødvendig/ønsket å implementere noe relatert til Microsoft sine produkter.
- TICK skal implementeres som en monolittisk utgave og en mikrotjeneste utgave.
- TIG kan implementeres om tiden er til rådighet (hvor Grafana implementeres i mikrotjeneste-implementasjonen, og vil være med i de ytterlige to punktene).
- Det skal utføres en ytelsessammenligning mellom de forskjellige utgavene.
- Det skal foretas en sikkerhetsvurdering av implementasjonene.

# 2. Omfang

## 2.1 Problemstilling

Formålet med oppgaven er å implementere TICK i to forskjellige varianter, derav en monolittisk utgave og en mikrotjeneste-utgave. Det skal og utføres ytselsestester og sammenligne mellom variantene, og til slutt foreta en sikkerhetsevaluering. De to variantene vil fungere på veldig lik måte, den store forskjellen er da hvordan de skalerer og konfigureres. Om tiden er til rådighet vil TIG også implementeres og evalueres likt, og opp mot, de to andre variantene, men kun som mikrotjeneste-utgave.

## 2.2 Fagområde

Prosjektet vil i stor grad basere seg på Programmerbar Infrastruktur, som innebærer konfigurasjonsstyring, automatisert provisjonering av datakraft, og orkestrering av digital infrastruktur. Videre så skal gruppen utforske loggføring og overvåking av et større system, med mulighet for utvidelser mot andre relevante teknologier.

## 2.3 Avgrensning

Prosjektet skal utføres ved bruk av Puppet og Linux-baserte systemer. Gruppen skal ikke få komponentene i TICK til å fungere i, eller mot, et Windows-miljø. Oppdragsgiveren gir frihet til gruppen til å ta egne antagelser og valg underveis, da oppgaven er svært åpen i hvordan den utføres, med forbehold om at de blir informert og har mulighet til å godkjenne valgene som blir tatt.

# 3. Prosjektorganisering

## 3.1 Ansvarsforhold og roller

Da gruppen består av kun to medlemmer så inntar Adrian L Lange posisjonen som leder for prosjektet. De fleste beslutninger vil tas innad i gruppen, og eventuelle uenigheter kan oppklares ved innspill fra veileder (Eigil Obrestad v. NTNU), oppdragsgiver (Patrik Storm Olsen v. Basefarm), eller andre relevante ressurspersoner og kilder om det skulle trengs. Oppdragsgiver har fullmakt for eventuelle større beslutninger for prosjektet, så lenge det er innenfor avgrensningene for prosjektet (se kap 2.2).

## 3.2 Rutiner og regler i gruppen

Utover møter (se kap 4.2) og prosjektavtalen (se vedlegg) så jobber gruppemedlemmene enten hver for seg ihht. oppgaver tildelt i prosessrammeverket (se kap 4.1), eller ved parprogrammering med samme oppgavesett. Se vedlagte grupperegler for retningslinjer og regler.

# 4. Planlegging, oppfølging og rapportering

## 4.1 Hovedinndeling av prosjektet

Prosjektet går ut på å implementere TICK-stack i to forskjellige utgaver, samt å argumentere/utforske andre løsninger, sikkerheten og ytelse i de forskjellige implementasjonene.

Da det er flere deler av prosjektet, samt at TICK-stacken er oppdelt av fire forskjellige komponenter så trengs et prosessrammeverk som tillater å dele opp prosjektet og dets komponenter, delegere oppgaver og sette tidsfrister. Denne modellen må være godt oversiktlig og ha en smidig måte å justere oppgavene på. Prosjektgruppen er særdeles liten, men uansett så er god oversikt over oppgaver i alle stadier et krav.

Ett av valgene vi har er Scrum, hvor man har hyppig utvikling i intervaller, med møter imellom for å drøfte endringer. Denne hadde fungert for gruppen, men siden det er kun to medlemmer med hyppigere kommunikasjon enn det Scrum er designet for, så ser vi på andre alternativ.

Vi kom frem til at en modell basert på Kanban fungerer best for gruppen, da vi har mulighet til å dele oppgaver til enkelte gruppemedlem og sette tidsfrister hos disse. Vi vil bruke Trello da det ser ut som å være det beste tilgjengelige uten å tenke på kostnad. Vi så på "prosjekter" hos GitHub som et alternativ til Trello, da man kan integrere oppgaver direkte inn i arbeidet, men dette systemet hadde ikke støtte for tidsfrister. Oppgavetavlen utvikles og utfylles i fellesskap (gruppemøter), og hvert medlem er pliktig til å holde denne oppdatert i løpet av progresjonen av oppgaven (ihht. gruppereglene).

## 4.2 Plan for statusmøte og beslutningspunkter i perioden

Gruppen har fastsatte møter i plenum, som følger:

- Faste møter med veileder annenhver mandag kl 1200, disse fungerer som statusmøter hvor veileder kan gi tilbakemeldinger og innspill.
- Faste møter med veileder ved hver milepæl i prosjektet, som vil da erstatte et statusmøte hvis de faller på samme dato.
- Faste møter med gruppen hver mandag, kl. 1300, i etterkant av veileder-møtet.
- Møter med oppdragsgiver avtales etter behov eller ønske, typisk digitalt grunnet distanse.

Andre tidspunkter for møter kan avtales med de det gjelder, god tid i forkant.

Møtene brukes for å diskutere problemstillinger som de oppstår, komme frem til beslutninger, samt og oppdatere progresjonen gjennom prosjektet. I møtene vil et gruppemedlem fungere som referent som da roteres mellom møter.

Beslutningspunkter blir basert på milepæler i prosjektplanen (se kap 6), og ved hver milepæl så vil gruppen re-evaluere tidsskjema og oppdatere prosessrammeverket.

# 5. Organisering av kvalitetssikring

## 5.1 Dokumentasjon, standardbruk og kildekode

Alt av funn, design- og implementasjonsdetaljer dokumenteres forløpende slik at de ikke havner på avveie. Til dette bruker vi markeringsspråket *Markdown*, som versjonskontrolleres sammen med hoveddokumentet.

Konfigurasjonen skal følge gode standarder i forhold til syntaks og dokumentasjon, derav Puppets egen "Language Style Guide" tas i bruk.

Kilden til konfigurasjonsstyringen og hoveddokumentet vil være versjonskontrollert med Git og distribuert i reservoar hos GitHub. Konfigurasjonsstyringen vil være åpen til allmenheten, mens hoveddokumentet blir lukket med innsyn etter invitasjon inntil prosjektets slutt, hvor den deretter vil lisensieres og blir åpnet mot allmenheten, etter NTNUs prosedyrer for distribuering av fullførte bacheloroppgaver.

## 5.2 Risikoanalyse

Basert på NTNU ROS Veiledning, så har vi foretatt en initiell verdivurdering, og kommet frem til 6 verdier. Under er en oversikt med navn på verdi og totalvurdering basert på verdiens viktighet i form av KIT (Konfidensialitet, Integritet, Tilgjengelighet). Totalvurdering er da satt til den høyeste verdien av de tre punktene i KIT.

| **Verdier** | **Totalvurdering** |
|---|---|
| Rapporten | 4 |
| Dokumentasjon | 3 |
| Kommunikasjon mellom gruppe og oppdragsgiver | 3 |
| Kommunikasjon mellom gruppe og veileder | 2 |
| Kommunikasjon mellom gruppemedlemmer | 4 |
| Tidsrammene | 4 |

*Figur 1 – Verdier med totalvurdering ihht. KIT*

### Sannsynlighetsintervall

Sannsynlighet følger ROS sine standard intervall, men skalert ned til våre tidsrammer, altså fem måneder.

| Grad | Beskrivelse | Utdyping |
|---|---|---|
| **4** | Svært sannsynlig | Oftere enn én gang i uken |
| **3** | Sannsynlig | Opptil annenhver uke |
| **2** | Mindre sannsynlig | Opptil én gang i måneden |
| **1** | Usannsynlig | Sjeldnere enn månedlig |

*Figur 2* – Sannsynlighetsintervall

### Konsekvensintervaller

Konsekvensintervallene baserer seg på tid, da dette er den eneste reelle kostnaden i vårt prosjekt.

| Grad | Beskrivelse | Kostnad (tid) |
|---|---|---|
| **4** | Kritisk | 7 dagsverk |
| **3** | Alvorlig | 3-7 dagsverk |
| **2** | Liten | 1-3 dagsverk |
| **1** | Ubetydelig | < 1 dagsverk |

*Figur 3 – Konsekvensintervall*

### Risikovurdering før tiltak

Detaljert hver verdi, dens risiko og trussel. Sannsynlighet og konsekvens er basert på NTNU ROS Veiledning, og har 4 nivåer.

| # | Beskrivelse | Påvirker | Sann. | Kons. | Initiell vurdering (S x K) |
|---|---|---|---|---|---|
| **A** | Langvarig fravær | Rapporten | 1 | 4 | 4 |
| **B** | Kortvarig fravær | Kommunikasjon mellom gruppe og veileder | 1 | 2 | 2 |
| **C** | Uoverensstemmelser internt i gruppen | Kommunikasjon mellom gruppemedlemmer | 2 | 1 | 2 |
| **D** | Uoverensstemmelser mellom gruppe og oppdragsgiver | Kommunikasjon mellom gruppe og oppdragsgiver | 1 | 3 | 3 |
| **E** | Tap av tilgjengelighet | Rapporten, Dokumentasjon | 2 | 3 | 6 |
| **F** | Lav motivasjon | Tidsrammene | 2 | 2 | 4 |

*Figur 4 – Risikoer*

Risikoene blir så plassert i en risikomatrise ihht. ROS, som da er fargekodet etter alvorlighet. Horisontalt i matrisen er konsekvensen rangert fra lav til høy, vertikalt er sannsynligheten, fra lav til høy.

| | Ubetydelig (1) | Liten (2) | Alvorlig (3) | Kritisk (4) |
|---|---|---|---|---|
| Usannsynlig (1) | | **B** | **D** | **A** |
| Mindre sannsynlig (2) | **C** | **F** | **E** | |
| Sannsynlig (3) | | | | |
| Svært sannsynlig (4) | | | | |

*Figur 5 – Risikomatrise før tiltak*

## Risikovurdering etter tiltak

Vi vurderer kun tiltak for de risikoene som faller i gul kategori eller høyere.

| # | Tiltak | Sluttvurdering |
|---|---|---|
| **A** | Justering av tildelte oppgaver for å passe på at arbeid blir utført | Konsekvensen blir redusert til nivå 3, da det fortsatt kan være en del arbeid som må gjøres |
| **E** | Backup-rutiner av arbeid slik at vi kan jobbe lokalt og ikke være avhengig av verktøy som befinner seg på nett | Konsekvensen blir redusert til nivå 1, da vi har full mulighet til å jobbe med oppgaver, bare ikke dele det i sanntid |
| **F** | Sosial aktivitet med gruppemedlemmene | Konsekvensen blir redusert til nivå 1, da det går med en dag/kveld på den sosiale aktiviteten |

*Figur 6 – Tiltak*

| | Ubetydelig (1) | Liten (2) | Alvorlig (3) | Kritisk (4) |
|---|---|---|---|---|
| Usannsynlig (1) | | **B** | **A, D** | |
| Mindre sannsynlig (2) | **C, E, F** | | | |
| Sannsynlig (3) | | | | |
| Svært sannsynlig (4) | | | | |

*Figur 7 – Risikomatrise etter tiltak*

Da samtlige risikoer er innen grønn sone ved tiltak så ser vi på risikoene som overkommelige.

# 6. Plan for gjennomføring

## 6.1 Hovedinndeling av prosjektet

Vi har valgt å dele prosjektet opp i fire hoveddeler. Første del vil omhandle datainnsamling. Andre del er hvor vi designer og implementerer de to utgavene og gjør klart for tredje del hvor vi risikovurderer og ytelsestester systemene, samt utfører en sammenligning mellom versjonene vi har implementert. For hver del så vil vi skrive ned alt som er relevant i rapporten. I fjerde del skal vi gå gjennom rapporten og fjerne irrelevant eller redundant informasjon, og gi den en finpuss før innlevering den 20. Mai.

### Del 1 - Datainnsamling.

Datainnsamling vil gå med til innhøsting av informasjon om TICK og dets tekniske detaljer som vil være til bruk senere. Vi vil utføre en uformell meningsmåling, hvor vi skal samle inn data om hva IT profesjonelle synes og mener om TICK. Målingen vil være i form av datainnhøsting fra diverse forum, tekniske blogger, o.l.

### Del 2 - Design og implementasjon

Vi kommer til å bruke litt av tiden satt av til å oppfriske kunnskap om Puppet, og gjøre klar en infrastruktur som vi kan bruke til å rulle ut TICK i forskjellig utgaver. Resten av denne delen går til planlegging, design, og implementasjon av TICK i både monolittisk og mikrotjeneste utgave.

### Del 3 - Ytelse og sikkerhetsvurdering

Da vi går IT-Drift og Informasjonssikkerhet føler vi at det er naturlig å inkludere en risikovurdering av systemet vi har implementert. Det skal også foretas en ytelsesvurdering av begge utgavene, samt en sammenligning mellom disse.

## 6.2 Milepæler

Under finner man en tabell over de største milepælene i prosjektet, og hvilke datoer vi antar at delen starter på, og når den slutter. Vi estimerer mer tid enn det vi tror er nødvendig, så justeringer vil forekomme i løpet av prosjektperioden.

Ved hver milepæl så re-evaluerer vi statusen og fremdriften i prosjektet og utbedrer planen videre. Det er ønskelig med tilbakemelding og innspill fra oppdragsgiver og veileder ved disse milepælene.

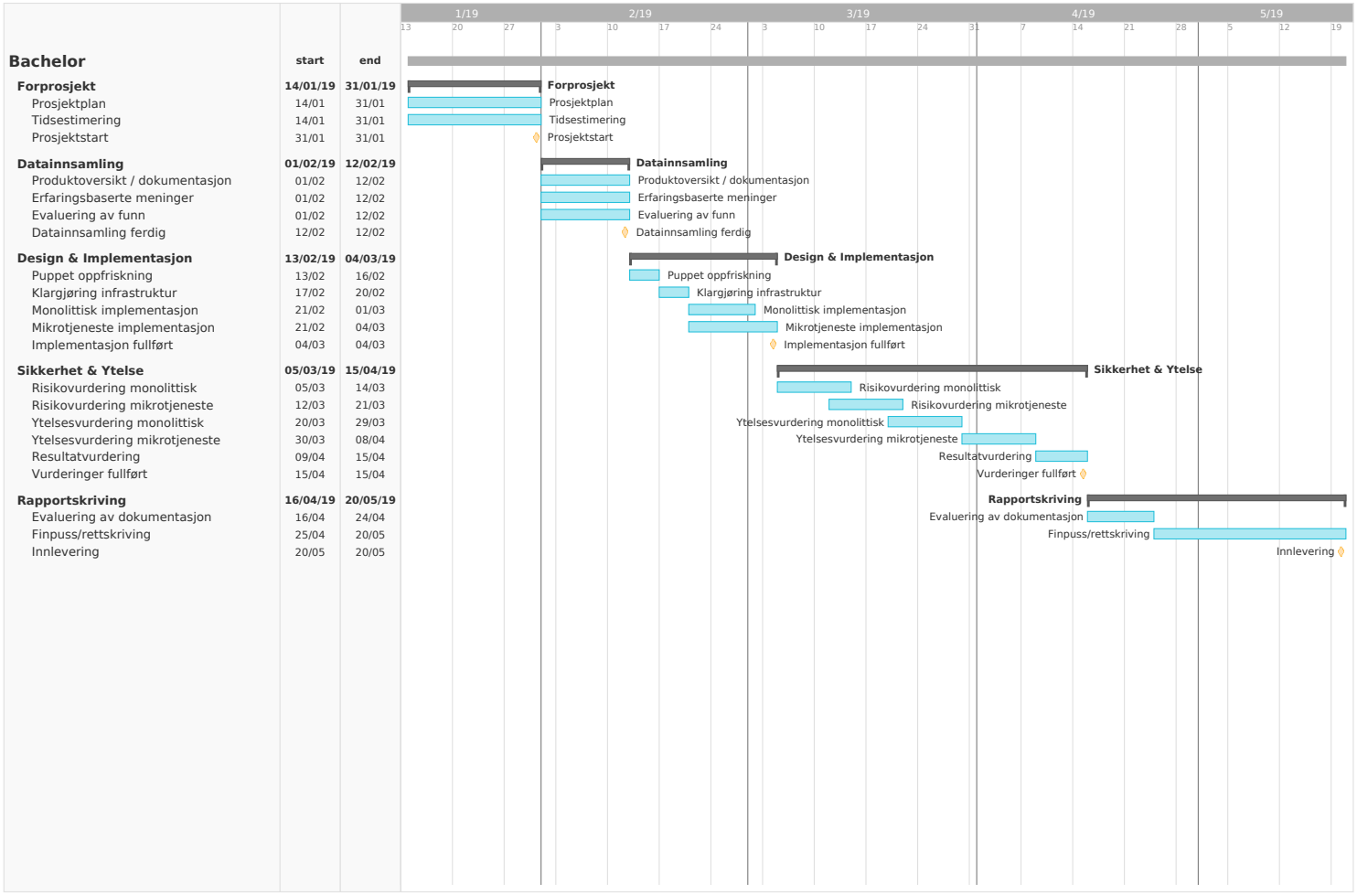
Rapportskriving er representert som oppgaver/punkter i tidstabellen, men de er kontinuerlige prosesser gjennom hele tidsperioden, er bare vanskelig å visualisere dette i en tabell.

| | | |
|---|---|---|
| **Del 1 - Datainnsamling** | Start: 1.2.2019 | Slutt: 12.2.2019 |
| Produktoversikt | | |
| Meningsmåling | | |
| Evaluering av funn | | |
| Rapportskriving | | |
| **Del 2 - Design og implementasjon** | Start: 13.2.2019 | Slutt: 4.3.2019 |
| Oppfriskning | | |
| Klargjøring av infrastruktur | | |
| Planlegge, designe, implementere monolittisk utgave | | |
| Planlegge, designe, implementere mikrotjeneste utgave | | |
| Rapportskriving | | |
| **Del 3 - Ytelse og Sikkerhetsvurdering** | Start: 5.3.2019 | Slutt: 15.4.2019 |
| Risikovurdering av hver utgave | | |
| Ytelsesvurdering av hver utgave | | |
| Sammenligning av resultater | | |
| Rapportskriving | | |
| **Del 4 - Rapportskriving** | Start: 16.4.2019 | Slutt: 20.5.2019 |
| Evaluering av dokumentasjon, designdokumenter, rapport | | |
| Finpuss og finskriving | | |

*Figur 8 – Milepæler*

Vedlagt er et Gantt-skjema som viser tidsplanen m/ milepælene.

# Vedlegg - Gantt-skjema




| Bachelor | start | end |
|---|---|---|
| **Forprosjekt** | **14/01/19** | **31/01/19** |
| Prosjektplan | 14/01 | 31/01 |
| Tidsestimering | 14/01 | 31/01 |
| Prosjektstart | 31/01 | 31/01 |
| **Datainnsamling** | **01/02/19** | **12/02/19** |
| Produktoversikt / dokumentasjon | 01/02 | 12/02 |
| Erfaringsbaserte meninger | 01/02 | 12/02 |
| Evaluering av funn | 01/02 | 12/02 |
| Datainnsamling ferdig | 12/02 | 12/02 |
| **Design & Implementasjon** | **13/02/19** | **04/03/19** |
| Puppet oppfriskning | 13/02 | 16/02 |
| Klargjøring infrastruktur | 17/02 | 20/02 |
| Monolittisk implementasjon | 21/02 | 01/03 |
| Mikrotjeneste implementasjon | 21/02 | 04/03 |
| Implementasjon fullført | 04/03 | 04/03 |
| **Sikkerhet & Ytelse** | **05/03/19** | **15/04/19** |
| Risikovurdering monolittisk | 05/03 | 14/03 |
| Risikovurdering mikrotjeneste | 12/03 | 21/03 |
| Ytelsesvurdering monolittisk | 20/03 | 29/03 |
| Ytelsesvurdering mikrotjeneste | 30/03 | 08/04 |
| Resultatvurdering | 09/04 | 15/04 |
| Vurderinger fullført | 15/04 | 15/04 |
| **Rapportskriving** | **16/04/19** | **20/05/19** |
| Evaluering av dokumentasjon | 16/04 | 24/04 |
| Finpuss/rettskriving | 25/04 | 20/05 |
| Innlevering | 20/05 | 20/05 |

# C Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

# Prosjektavtale

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

BASEFARM AS

(oppdragsgiver), og

ADRIAN LUND-LANGE

VETLE TANGEN HOEN

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 9/1-19 til 20/5-19.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Eigil Obrestad

Oppdragsgivers kontaktperson (navn): Patrik Storm Olsen

Student(er) (signatur): Vetle Moen dato 14/1-19

Adrian Monge dato 14/1-19

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato 16/01/19

*Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.*
*Godkjennes digitalt av instituttleder/faggruppeleder.*

*Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.*
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato 25/1/19

# D Gruppeavtale

# Grupperegler

Adrian L Lange, Vetle T Moen

14. Januar 2019

## 1. Introduksjon

Grupperegler for utføring av Bachelor prosjekt, med tilhørende utvikling og rapportskriving.

## 2. Arbeidskrav

Det forventes minimum 30 arbeidstimer per person i uken under hele prosjektperioden. Det skal allikevel være av høy prioritet å fullføre tidspressende arbeidsoppgaver som burde bli gjort. Helg kan til nød tas i bruk. Ved planlagte arbeidstimer er gruppen pålagt å møte i tidsrommet, ellers kan man jobbe til alle døgnets tider. Hovedsakelig vil arbeidstimer være hverdager 08:00 til 16:00. Arbeid utover dette kan ikke forventes av alle gruppemedlemmer (jfr. Noen har familie/jobbmessige/andre plikter). Dersom det skulle skje at vi havner langt bak med arbeidsoppgaver kan gruppen i fellesskap bli enige om å jobbe overtid (for eksempel 10 timer i løpet av en helg).

Gruppemedlemmene har møteplikt på prosjekt- og statusmøter hver uke. Er et gruppemedlem utilgjengelig i tidsrommet må gruppemedlemmet gi beskjed i god tid slik at møtet evt. kan flyttes. Dato for møter er fastsatt til mandager kl 13.00, rett etter veiledningsmøte kl 12.00. Vi tilstreber møtetider hvor alle gruppemedlemmer er tilgjengelig. Det forventes aktiv deltagelse i arbeidet fra samtlige medlemmer. Gruppen har et felles mål om å nå frister (gruppen ønsker gode marginer, slik at utfordringer kan overkommes).

Ved føring av referat på statusmøter skal disse inneholde:

- Fattede avgjørelser
- Hva som er utført/ferdig
- Fremdriftsplan mot neste uke
- Ting som må gjøres blir oppdatert og frister settes/oppdateres.

Standardformat på dokumenter:

- PDF (innleveringer)
- Microsoft Word (møtereferat og dokumentnotater)
- LaTeX (bacheloroppgaven, prosjektplan)

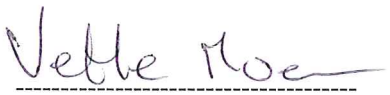
## 3. Uforutsette problemer

Å være forberedt/gjort oppgaver til avtalt tid, gi beskjed om man "henger etter", og evt. enighet om at "dette er greit"/kan hentes inn. Ved faglige uenigheter skal gruppen forsøke å komme til enighet, evt. finne ekspertvurderinger. Gruppemedlemmene må tidlig si ifra om de ikke er fornøyd med innsatsen til enkelte gruppemedlemmer. Dersom det skulle oppstå et større problem; samtale(r) mellom alle gruppemedlemmer og evt. veileder, der regelbrudd blir påpekt. Konkret (tidsfrister, forventet arbeidsinnsats, m.m.) om hva som kreves for å "bøte på skaden". Sette inn tiltak mot regelbruddet. Det er ikke lov å ekskludere et gruppemedlem.

Ved sykdom eller fravær under 7 dager må enkelte gruppemedlem ta igjen tapt arbeid påfølgende uke. Dersom fraværet går over 7 dager, skal veileder kobles inn i beslutningen og dialog med instituttet vil avgjøre om personen skal gjennomføre bacheloroppgaven.

## 4. Signaturer

Adrian Lund-Lange



Vetle Tangen Moen

# E Puppet-modul

## E.1 Toppklasse og repo for installasjon

Listing E.1: init.pp

```
1  # == Class: tick
2  #
3  class tick_stack {}
```

Listing E.2: repo.pp

```
1  # == Class: tick_stack::repo
2  #
3  class tick_stack::repo (
4    $osfamily          = $::facts['os']['family'],
5    ){
6
7    if $osfamily == 'Debian' {
8      $operatingsystem  = downcase($::facts['os']['name'])
9      $oscodename       = downcase($::facts['os']['distro']['codename'])
10     apt::source { 'influx':
11       location => "https://repos.influxdata.com/${operatingsystem}",
12       repos    => 'stable',
13       release  => $oscodename,
14       key      => {
15         'id'     => '05CE15085FC09D18E99EFB22684A14CF2582E0C5',
16         'source' => 'https://repos.influxdata.com/influxdb.key',
17       }
18     }
19   } elsif $osfamily == 'RedHat' {
20     yumrepo { 'influx':
21       baseurl  => 'https://repos.influxdata.com/rhel/$releasever/$basearch/ ↙
           stable',
22       descr    => 'InfluxDB Repository - RHEL \$releasever',
23       enabled  => '1',
24       gpgcheck => '1',
25       gpgkey   => 'https://repos.influxdata.com/influxdb.key'
26     }
27   }
28 }
```

## E.2 Telegraf

Listing E.3: telegraf.pp

```
# == Class: tick_stack::telegraf
#
class tick_stack::telegraf (
  # For Install-class
  $ensure      = $tick_stack::telegraf::params::ensure,

  # For Config-class
  $conf_path   = $tick_stack::telegraf::params::conf_path,
  $template    = $tick_stack::telegraf::params::template,

  # For Service-class
  $service     = $tick_stack::telegraf::params::service,
  $enable      = $tick_stack::telegraf::params::enable,
  $hasrestart  = $tick_stack::telegraf::params::hasrestart,
  $hasstatus   = $tick_stack::telegraf::params::hasstatus,

  # Specific configuration hashes:
  Hash $global_tags = $tick_stack::telegraf::params::global_tags,
  Hash $agent       = $tick_stack::telegraf::params::agent,
  Hash $outputs     = $tick_stack::telegraf::params::outputs,
  Hash $inputs      = $tick_stack::telegraf::params::inputs,

  ) inherits tick_stack::telegraf::params {

  include tick_stack::repo

  contain tick_stack::telegraf::install
  contain tick_stack::telegraf::config
  contain tick_stack::telegraf::service

  Class['tick_stack::telegraf::install']
  -> Class['tick_stack::telegraf::config']
  ~> Class['tick_stack::telegraf::service']

}
```

Listing E.4: telegraf config

```
# == Class: tick_stack::telegraf::config
#
class tick_stack::telegraf::config {
  assert_private()

  $conf_path = $tick_stack::telegraf::conf_path
  $template  = $tick_stack::telegraf::template

  file { $conf_path:
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template($template)
  }
}
```

Listing E.5: telegraf install

```
# == Class: tick_stack::telegraf::install
#
class tick_stack::telegraf::install {
  assert_private()

  $ensure = $tick_stack::telegraf::ensure

  package { 'telegraf':
    ensure => $ensure,
  }
}
```

Listing E.6: telegraf params

```
# == Class: tick_stack::telegraf::params
#
class tick_stack::telegraf::params {
  # Install options
  $ensure     = 'present'

  # Config options
  $conf_path  = '/etc/telegraf/telegraf.conf'
  $template   = 'tick_stack/telegraf.conf.erb'

  # Service options
  $service    = 'running'
  $enable     = true
  $hasrestart = true
  $hasstatus  = true

  # Specific hashes for configuration of the agent
  $global_tags = {
    'dc' => 'home',
  }

  $agent       = {
    'interval' => '10s',
  }

  $outputs     = {
    'influxdb' => {
      'urls'      => [ 'http://localhost:8086' ],
      'database'  => '"telegraf"',
      'precision' => '"s"',
      },
  }

  $inputs      = {
    'cpu' => {
      'percpu'   => true,
      'totalcpu' => true,
    },
    'system' => {}
  }
}
```

Listing E.7: telegraf service

```
# == Class: tick_stack::telegraf::service
```

```
2  #
3  class tick_stack::telegraf::service {
4    assert_private()
5
6    $service    = $tick_stack::telegraf::service
7    $enable     = $tick_stack::telegraf::enable
8    $hasrestart = $tick_stack::telegraf::hasrestart
9    $hasstatus  = $tick_stack::telegraf::hasstatus
10
11   service { 'telegraf':
12     ensure     => $service,
13     enable     => $enable,
14     hasrestart => $hasrestart,
15     hasstatus  => $hasstatus,
16   }
17 }
```

Listing E.8: telegraf template

```
1  # Telegraf Configuration
2  #
3  # THIS FILE IS MANAGED BY PUPPET
4  #
5  <% if scope['tick_stack::telegraf::global_tags'] -%>
6  [global_tags]
7  <%   scope['tick_stack::telegraf::global_tags'].each do |key, value| -%>
8    <%= key %> = "<%= value %>"
9  <%   end -%>
10 <% end -%>
11
12 <% if scope['tick_stack::telegraf::agent'] -%>
13 [agent]
14 <%   scope['tick_stack::telegraf::agent'].each do |key, value| -%>
15   <%= key %> = "<%= value %>"
16 <%   end -%>
17 <% end -%>
18
19 <% if scope['tick_stack::telegraf::outputs'] -%>
20 # OUTPUTS
21 <% scope['tick_stack::telegraf::outputs'].each do |key,hash| -%>
22 [[outputs.<%= key %>]]
23 <% hash.each do |key,value| -%>
24 <%= key %> = <%= value %>
25 <%   end -%>
26 <%  end -%>
27 <% end -%>
28
29 <% if scope['tick_stack::telegraf::inputs'] -%>
30 # INPUTS
31 <% scope['tick_stack::telegraf::inputs'].each do |key,hash| -%>
32 [[inputs.<%= key %>]]
33 <% hash.each do |key,value| -%>
34 <%= key %> = <%= value %>
35 <%   end -%>
36
37 <%  end -%>
38 <% end -%>
```

## E.3 Influxdb

Listing E.9: influxdb.pp

```
1  # == Class: tick_stack::influxdb
2  #
3  class tick_stack::influxdb (
4    # For Install-class
5    $ensure                              = $tick_stack::influxdb::params:: ↙
         ensure,
6
7    # For Config-class
8    $conf_path                           = $tick_stack::influxdb::params:: ↙
         conf_path,
9    $template                            = $tick_stack::influxdb::params:: ↙
         template,
10
11   # For Service-class
12   $service                             = $tick_stack::influxdb::params:: ↙
         service,
13   $enable                              = $tick_stack::influxdb::params:: ↙
         enable,
14   $hasrestart                          = $tick_stack::influxdb::params:: ↙
         hasrestart,
15   $hasstatus                           = $tick_stack::influxdb::params:: ↙
         hasstatus,
16
17   # Specific settings in config
18   $bind_addr                           = $tick_stack::influxdb::params:: ↙
         bind_addr,
19   $meta_dir                            = $tick_stack::influxdb::params:: ↙
         meta_dir,
20   $data_dir                            = $tick_stack::influxdb::params:: ↙
         data_dir,
21   $wal_dir                             = $tick_stack::influxdb::params:: ↙
         wal_dir,
22
23   # Optionals for InfluxDB Data config
24   $trace_logging_enabled               = $tick_stack::influxdb::params:: ↙
         trace_logging_enabled,
25   $query_log_enabled                   = $tick_stack::influxdb::params:: ↙
         query_log_enabled,
26   $cache_max_memory_size               = $tick_stack::influxdb::params:: ↙
         cache_max_memory_size,
27   $cache_snapshot_memory_size          = $tick_stack::influxdb::params:: ↙
         cache_snapshot_memory_size,
28   $cache_snapshot_write_cold_duration = $tick_stack::influxdb::params:: ↙
         cache_snapshot_write_cold_duration,
29   $compact_full_write_cold_duration    = $tick_stack::influxdb::params:: ↙
         compact_full_write_cold_duration,
30   $max_series_per_database             = $tick_stack::influxdb::params:: ↙
         max_series_per_database,
31   $max_values_per_tag                  = $tick_stack::influxdb::params:: ↙
         max_values_per_tag,
32
33   # Hashes for the config template
34   Hash $http                           = $tick_stack::influxdb::params::http,
35   Hash $coordinator                    = $tick_stack::influxdb::params:: ↙
         coordinator,
```

```
36    Hash $retention                      = $tick_stack::influxdb::params:: ↙
          retention,
37    Hash $shardprecreation               = $tick_stack::influxdb::params:: ↙
          shardprecreation,
38    Hash $monitor                        = $tick_stack::influxdb::params:: ↙
          monitor,
39    Hash $logging                        = $tick_stack::influxdb::params:: ↙
          logging,
40    Hash $subscriber                     = $tick_stack::influxdb::params:: ↙
          subscriber,
41    Hash $tls                            = $tick_stack::influxdb::params::tls,
42    ) inherits tick_stack::influxdb::params  {
43
44    include tick_stack::repo
45
46    contain tick_stack::influxdb::install
47    contain tick_stack::influxdb::config
48    contain tick_stack::influxdb::service
49
50    Class['tick_stack::influxdb::install']
51    -> Class['tick_stack::influxdb::config']
52    ~> Class['tick_stack::influxdb::service']
53
54  }
```

Listing E.10: influxdb config

```
1  # == Class: tick_stack::influxdb::config
2  #
3  class tick_stack::influxdb::config {
4    assert_private()
5
6    $conf_path = $tick_stack::influxdb::conf_path
7    $template  = $tick_stack::influxdb::template
8
9    file { $conf_path:
10     ensure  => file,
11     owner   => 'root',
12     group   => 'root',
13     mode    => '0644',
14     content => template($template)
15   }
16 }
```

Listing E.11: influxdb install

```
1  # == Class: tick_stack::influxdb::install
2  #
3  class tick_stack::influxdb::install {
4    assert_private()
5
6    $ensure = $tick_stack::influxdb::ensure
7
8    package { 'influxdb':
9      ensure => $ensure,
10   }
11 }
```

Listing E.12: influxdb params

```
# == Class: tick_stack::influxdb::params
#
class tick_stack::influxdb::params {
  # Install options
  $ensure     = 'present'

  # Config options
  $conf_path  = '/etc/influxdb/influxdb.conf'
  $template   = 'tick_stack/influxdb.conf.erb'

  # Service options
  $service    = 'running'
  $enable     = true
  $hasrestart = true
  $hasstatus  = true

  # Specific options for InfluxDB config
  $bind_addr = '127.0.0.1:8088'
  $meta_dir = '/var/lib/influxdb/meta'
  $data_dir = '/var/lib/influxdb/data'
  $wal_dir = '/var/lib/influxdb/wal'

  # Optionals for InfluxDB data config, defaults are InfluxDBs recommended settings
  $trace_logging_enabled              = false
  $query_log_enabled                  = true
  $cache_max_memory_size              = 1048576000
  $cache_snapshot_memory_size         = 26214400
  $cache_snapshot_write_cold_duration = '10m'
  $compact_full_write_cold_duration   = '4h'
  $max_series_per_database            = 1000000
  $max_values_per_tag                 = 100000

  # HTTP config hash
  $http             = {
    'http_enable' => true,
    'http_bind'   => '":8086"',
    'http_auth'   => false,
  }

  # Undefined hashes for the config template
  $coordinator      = {}
  $retention        = {}
  $shardprecreation = {}
  $monitor          = {}
  $logging          = {}
  $subscriber       = {}
  $tls              = {}

}
```

Listing E.13: influxdb service

```
# == Class: tick_stack::influxdb::service
#
class tick_stack::influxdb::service {
  assert_private()

```

```
6     $service     = $tick_stack::influxdb::service
7     $enable      = $tick_stack::influxdb::enable
8     $hasrestart  = $tick_stack::influxdb::hasrestart
9     $hasstatus   = $tick_stack::influxdb::hasstatus
10
11    service { 'influxdb':
12      ensure     => $service,
13      enable     => $enable,
14      hasrestart => $hasrestart,
15      hasstatus  => $hasstatus,
16    }
17  }
```

Listing E.14: influxdb template

```
1  # Influxdb Configuration
2  #
3  # THIS FILE IS MANAGED BY PUPPET
4  #
5  # Global settings
6  [global]
7  bind-address = "<%= scope['tick_stack::influxdb::bind_addr'] %>"
8
9  # Metastore settings
10 [meta]
11 dir = "<%= scope['tick_stack::influxdb::meta_dir'] %>"
12
13 # Data settings
14 [data]
15 dir = "<%= scope['tick_stack::influxdb::data_dir'] %>"
16 wal-dir = "<%= scope['tick_stack::influxdb::wal_dir'] %>"
17 trace_logging_enabled = <%= scope['tick_stack::influxdb:: ↙
       trace_logging_enabled'] %>
18 query_log_enabled = <%= scope['tick_stack::influxdb::query_log_enabled'] %>
19 cache_max_memory_size = <%= scope['tick_stack::influxdb:: ↙
       cache_max_memory_size'] %>
20 cache_snapshot_memory_size = <%= scope['tick_stack::influxdb:: ↙
       cache_snapshot_memory_size'] %>
21 cache_snapshot_write_cold_duration = "<%= scope['tick_stack::influxdb:: ↙
       cache_snapshot_write_cold_duration'] %>"
22 compact_full_write_cold_duration = "<%= scope['tick_stack::influxdb:: ↙
       compact_full_write_cold_duration'] %>"
23 max_series_per_database = <%= scope['tick_stack::influxdb:: ↙
       max_series_per_database'] %>
24 max_values_per_tag = <%= scope['tick_stack::influxdb::max_values_per_tag'] ↙
       %>
25
26 <%# The rest are optional hashes that can be defined if needed, they are ↙
       undef in params %>
27 <% if scope['tick_stack::influxdb::http'] -%>
28 [http]
29 <%   scope['tick_stack::influxdb::http'].each do |key, value| -%>
30   <%= key %> = <%= value %>
31 <%   end -%>
32 <% end -%>
33
34 <% if scope['tick_stack::influxdb::coordinator'] -%>
35 [coordinator]
36 <%   scope['tick_stack::influxdb::coordinator'].each do |key, value| -%>
```

```
37    <%= key %> = <%= value %>
38  <%   end -%>
39  <% end -%>
40
41  <% if scope['tick_stack::influxdb::retention'] -%>
42  [retention]
43  <%   scope['tick_stack::influxdb::retention'].each do |key, value| -%>
44    <%= key %> = <%= value %>
45  <%   end -%>
46  <% end -%>
47
48  <% if scope['tick_stack::influxdb::shardprecreation'] -%>
49  [shard-precreation]
50  <%   scope['tick_stack::influxdb::shardprecreation'].each do |key, value|  ↙
        -%>
51    <%= key %> = <%= value %>
52  <%   end -%>
53  <% end -%>
54
55  <% if scope['tick_stack::influxdb::monitor'] -%>
56  [monitor]
57  <%   scope['tick_stack::influxdb::monitor'].each do |key, value| -%>
58    <%= key %> = <%= value %>
59  <%   end -%>
60  <% end -%>
61
62  <% if scope['tick_stack::influxdb::logging'] -%>
63  [logging]
64  <%   scope['tick_stack::influxdb::logging'].each do |key, value| -%>
65    <%= key %> = <%= value %>
66  <%   end -%>
67  <% end -%>
68
69  <% if scope['tick_stack::influxdb::subscriber'] -%>
70  [subscriber]
71  <%   scope['tick_stack::influxdb::subscriber'].each do |key, value| -%>
72    <%= key %> = <%= value %>
73  <%   end -%>
74  <% end -%>
75
76  <% if scope['tick_stack::influxdb::tls'] -%>
77  [tls]
78  <%   scope['tick_stack::influxdb::tls'].each do |key, value| -%>
79    <%= key %> = <%= value %>
80  <%   end -%>
81  <% end -%>
```

## E.4 Chronograf

Listing E.15: chronograf.pp

```
# == Class: tick_stack::chronograf
#
class tick_stack::chronograf (
  # For Install-class
  $ensure              = $tick_stack::chronograf::params::ensure,

  # For Config-class
  $conf_path           = $tick_stack::chronograf::params::conf_path,
  $template            = $tick_stack::chronograf::params::template,

  # For Service-class
  $service             = $tick_stack::chronograf::params::service,
  $enable              = $tick_stack::chronograf::params::enable,
  $hasrestart          = $tick_stack::chronograf::params::hasrestart,
  $hasstatus           = $tick_stack::chronograf::params::hasstatus,

  # Specific configuration settings:
  $host                = $tick_stack::chronograf::params::host,
  $port                = $tick_stack::chronograf::params::port,
  $influxdb_url        = $tick_stack::chronograf::params::influxdb_url,
  $influxdb_username   = $tick_stack::chronograf::params::influxdb_username,
  $influxdb_password   = $tick_stack::chronograf::params::influxdb_password,
  $kapacitor_url       = $tick_stack::chronograf::params::kapacitor_url,
  $kapacitor_username  = $tick_stack::chronograf::params::kapacitor_username,
  $kapacitor_password  = $tick_stack::chronograf::params::kapacitor_password,
  $cert                = $tick_stack::chronograf::params::cert,
  $key                 = $tick_stack::chronograf::params::key,

  ) inherits tick_stack::chronograf::params {

  include tick_stack::repo

  contain tick_stack::chronograf::install
  contain tick_stack::chronograf::config
  contain tick_stack::chronograf::service

  Class['tick_stack::chronograf::install']
  -> Class['tick_stack::chronograf::config']
  ~> Class['tick_stack::chronograf::service']

}
```

Listing E.16: chronograf config

```
# == Class: tick_stack::chronograf::config
#
class tick_stack::chronograf::config {
  assert_private()

  $conf_path = $tick_stack::chronograf::conf_path
  $template  = $tick_stack::chronograf::template

  file { $conf_path:
    ensure  => file,
    owner   => 'root',
    group   => 'root',
```

```
13      mode    => '0644',
14      content => template($template),
15    }
16  }
```

Listing E.17: chronograf install

```
1  # == Class: tick_stack::chronograf::install
2  #
3  class tick_stack::chronograf::install {
4    assert_private()
5
6    $ensure = $tick_stack::chronograf::ensure
7
8    package { 'chronograf':
9      ensure => $ensure,
10   }
11 }
```

Listing E.18: chronograf params

```
1  # == Class: tick_stack::chronograf::params
2  #
3  class tick_stack::chronograf::params {
4    $ensure   = 'present'
5
6    # Service options
7    $service            = 'running'
8    $enable             = true
9    $hasrestart         = true
10   $hasstatus          = true
11
12   $conf_path          = '/etc/default/chronograf'
13   $template           = 'tick_stack/chronograf.erb'
14   # General options #
15   $host               = '0.0.0.0'
16   $port               = 8888
17   # InfluxDB options #
18   $influxdb_url       = 'http://127.0.0.1:8086'
19   $influxdb_username  = undef
20   $influxdb_password  = undef
21   # Kapacitor options #
22   $kapacitor_url      = undef
23   $kapacitor_username = undef
24   $kapacitor_password = undef
25   # TLS options #
26   $cert               = undef
27   $key                = undef
28 }
```

Listing E.19: chronograf service

```
1  # == Class: tick_stack::chronograf::service
2  #
3  class tick_stack::chronograf::service {
4    assert_private()
5
6    $service    = $tick_stack::chronograf::service
7    $enable     = $tick_stack::chronograf::enable
8    $hasrestart = $tick_stack::chronograf::hasrestart
```

```
 9    $hasstatus  = $tick_stack::chronograf::hasstatus
10
11    service { 'chronograf':
12      ensure     => $service,
13      enable     => $enable,
14      hasrestart => $hasrestart,
15      hasstatus  => $hasstatus,
16    }
17  }
```

Listing E.20: chronograf template

```
 1  HOST=<%= scope['tick_stack::chronograf::host'] %>
 2  PORT=<%= scope['tick_stack::chronograf::port'] %>
 3  <% if scope['tick_stack::chronograf::influxdb_url'] %>
 4  INFLUXDB_URL=<%= scope['tick_stack::chronograf::influxdb_url'] %>
 5  <% end %>
 6  <% if scope['tick_stack::chronograf::influxdb_username'] %>
 7  INFLUXDB_USERNAME=<%= scope['tick_stack::chronograf::influxdb_username'] %>
 8  <% end %>
 9  <% if scope['tick_stack::chronograf::influxdb_password'] %>
10  INFLUXDB_PASSWORD=<%= scope['tick_stack::chronograf::influxdb_password'] %>
11  <% end %>
12  <% if scope['tick_stack::chronograf::kapacitor_url'] %>
13  KAPACITOR_URL=<%= scope['tick_stack::chronograf::kapacitor_url'] %>
14  <% end %>
15  <% if scope['tick_stack::chronograf::kapacitor_username'] %>
16  KAPACITOR_USERNAME=<%= scope['tick_stack::chronograf::kapacitor_username'] ↙
        %>
17   <% end %>
18  <% if scope['tick_stack::chronograf::kapacitor_password'] %>
19  KAPACITOR_PASSWORD=<%= scope['tick_stack::chronograf::kapacitor_password'] ↙
        %>
20  <% end %>
21  <% if scope['tick_stack::chronograf::cert'] %>
22  TLS_CERTIFICATE=<%= scope['tick_stack::chronograf::cert'] %>
23  TLS_PRIVATE_KEY=<%= scope['tick_stack::chronograf::key'] %>
24  <% end %>
```

## E.5 Kapacitor

Listing E.21: kapacitor.pp

```
# == Class: tick_stack::kapacitor
#
class tick_stack::kapacitor (
  # For Install-class
  $ensure     = $tick_stack::kapacitor::params::ensure,

  # For Config-class
  $conf_path  = $tick_stack::kapacitor::params::conf_path,
  $template   = $tick_stack::kapacitor::params::template,

  # For Service-class
  $service    = $tick_stack::kapacitor::params::service,
  $enable     = $tick_stack::kapacitor::params::enable,
  $hasrestart = $tick_stack::kapacitor::params::hasrestart,
  $hasstatus  = $tick_stack::kapacitor::params::hasstatus,

  # Specific configuration settings:
  String $data_dir = $tick_stack::kapacitor::params::data_dir,
  Hash $http       = $tick_stack::kapacitor::params::http,
  Hash $influxdb   = $tick_stack::kapacitor::params::influxdb,
  Hash $replay     = $tick_stack::kapacitor::params::replay,
  Hash $storage    = $tick_stack::kapacitor::params::storage,
  Hash $task       = $tick_stack::kapacitor::params::task,
  Hash $load       = $tick_stack::kapacitor::params::load,
  Hash $logging    = $tick_stack::kapacitor::params::logging,

  ) inherits tick_stack::kapacitor::params {

  include tick_stack::repo

  contain tick_stack::kapacitor::install
  contain tick_stack::kapacitor::config
  contain tick_stack::kapacitor::service

  Class['tick_stack::kapacitor::install']
  -> Class['tick_stack::kapacitor::config']
  ~> Class['tick_stack::kapacitor::service']

}
```

Listing E.22: kapacitor config

```
# == Class: tick_stack::kapacitor::config
#
class tick_stack::kapacitor::config {
  assert_private()

  $conf_path = $tick_stack::kapacitor::conf_path
  $template  = $tick_stack::kapacitor::template

  file { $conf_path:
    ensure  => file,
    owner   => 'root',
    group   => 'root',
    mode    => '0644',
    content => template($template)
```

```
15    }
16  }
```

Listing E.23: kapacitor install

```
1  # == Class: tick_stack::kapacitor::install
2  #
3  class tick_stack::kapacitor::install {
4    assert_private()
5
6    $ensure = $tick_stack::kapacitor::ensure
7
8    package { 'kapacitor':
9      ensure => $ensure,
10   }
11 }
```

Listing E.24: kapacitor params

```
1  # == Class: tick_stack::telegraf::params
2  #
3  class tick_stack::kapacitor::params {
4    # Install options
5    $ensure     = 'present'
6
7    # Config options
8    $conf_path  = '/etc/kapacitor/kapacitor.conf'
9    $template   = 'tick_stack/kapacitor.conf.erb'
10
11   # Service options
12   $service    = 'running'
13   $enable     = true
14   $hasrestart = true
15   $hasstatus  = true
16
17   # Meta
18   $hostname = $facts['hostname']
19   $data_dir = '/var/lib/kapacitor'
20
21   # HTTP Settings:
22   $http = {
23     'bind_address'      => '":9092"',
24     'log_enabled'       => true,
25     'write_tracing'     => false,
26     'pprof_enabled'     => false,
27     'https_enabled'     => false,
28   }
29
30   # InfluxDB connection options:
31   $influxdb = {
32     'enabled'                    => true,
33     'name'                       => '"default"',
34     'default'                    => false,
35     'urls'                       => '["http://localhost:8086"]',
36     'username'                   => '""',
37     'password'                   => '""',
38     'ssl-ca'                     => '""',
39     'ssl-cert'                   => '""',
40     'ssl-key'                    => '""',
```

```
41      'insecure-skip-verify'         => false,
42      'timeout'                      => '"0s"',
43      'disable-subscriptions'        => false,
44      'subscription-protocol'        => '"http"',
45      'subscription-mode'            => '"server"',
46      'kapacitor-hostname'           => '""',
47      'http-port'                    => 0,
48      'udp-bind'                     => '""',
49      'udp-buffer'                   => 1000,
50      'udp-read-buffer'              => 0,
51      'startup-timeout'              => '"5m0s"',
52      'subscriptions-sync-interval' => '"1m0s"',
53    }
54
55    # The rest are suggested defaults:
56    $replay = {
57      'dir' => '"/var/lib/kapacitor/replay"'
58    }
59
60    $storage = {
61      'boltdb' => '"/var/lib/kapacitor/kapacitor.db"'
62    }
63
64    $task = {
65      'dir'                => '"/var/lib/kapacitor/tasks"',
66      'snapshot-interval' => '"1m0s"'
67    }
68
69    $load = {
70      'enabled' => 'false',
71      'dir'     => '"/var/lib/kapacitor/load"'
72    }
73
74    $logging = {
75      'file'  => '"STDERR"',
76      'level' => '"DEBUG"'
77    }
78
79  }
```

Listing E.25: kapacitor service

```
1  # == Class: tick_stack::kapacitor::service
2  #
3  class tick_stack::kapacitor::service {
4    assert_private()
5
6    $service    = $tick_stack::kapacitor::service
7    $enable     = $tick_stack::kapacitor::enable
8    $hasrestart = $tick_stack::kapacitor::hasrestart
9    $hasstatus  = $tick_stack::kapacitor::hasstatus
10
11   service { 'kapacitor':
12     ensure     => $service,
13     enable     => $enable,
14     hasrestart => $hasrestart,
15     hasstatus  => $hasstatus,
16   }
17 }
```

Listing E.26: kapacitor template

```
# kapacitor Configuration
#
# THIS FILE IS MANAGED BY PUPPET
#

hostname = "<%= scope['tick_stack::kapacitor::hostname'] %>"
data_dir = "<%= scope['tick_stack::kapacitor::data_dir'] %>"

<% if scope['tick_stack::kapacitor::http'] -%>
[http]
<%   scope['tick_stack::kapacitor::http'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::replay'] -%>
[replay]
<%   scope['tick_stack::kapacitor::replay'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::storage'] -%>
[storage]
<%   scope['tick_stack::kapacitor::storage'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::task'] -%>
[task]
<%   scope['tick_stack::kapacitor::task'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::load'] -%>
[load]
<%   scope['tick_stack::kapacitor::load'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::logging'] -%>
[logging]
<%   scope['tick_stack::kapacitor::logging'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>

<% if scope['tick_stack::kapacitor::influxdb'] -%>
[[influxdb]]
<%   scope['tick_stack::kapacitor::influxdb'].each do |key, value| -%>
  <%= key %> = <%= value %>
<%   end -%>
<% end -%>
```

# F Orkestreringskode

## F.1 Heat maler

Listing F.1: Heat mal for distribuert infrastruktur

```
1  # https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html#hot-spec-template-version
2  heat_template_version: queens
3
4  description: >
5    HOT template to create compute instances for a distributed implementation
6    of the TICK stack.
7
8  parameters:
9    image:
10     type: string
11     description: ID or name of image used for the compute instance(s).
12     default: CentOS 7.5 x86_64
13   flavor:
14     type: string
15     description: ID or name of flavor used for the compute instance(s).
16     default: m1.medium
17   key_name:
18     type: string
19     description: Name of keypair used to access the compute instance(s).
20   public_net_id:
21     type: string
22     description: ID or name of the public network floating IP addresses are
           allocated from.
23   mgmt_net:
24     type: string
25     description: Name of management network to be use.
26     default: tick_mgmt_net
27   mgmt_subnet:
28     type: string
29     description: Name of management subnet to be use.
30     default: tick_mgmt_net
31   manager_ip:
32     type: string
33     description: IP address of the manager instance on the management
           network.
34   dns_ip:
35     type: string
36     description: IP address of the DNS server on the management network.
37   influx_sec_group:
38     type: string
39     description: InfluxDB and Chronograf security group
40
41 resources:
42   master:
43     type: lib/generic_node.yaml
44     properties:
```

```
45        hostname: master
46        image:          { get_param: image }
47        flavor:         m1.medium
48        key_name:       { get_param: key_name }
49        sec_groups:
50          - default
51          - { get_param: influx_sec_group }
52        public_net_id: { get_param: public_net_id }
53        mgmt_net:       { get_param: mgmt_net }
54        mgmt_subnet:    { get_param: mgmt_subnet }
55        manager_ip:     { get_param: manager_ip }
56        dns_ip:         { get_param: dns_ip }
57
58    worker:
59      type: OS::Heat::ResourceGroup
60      properties:
61        count: 2
62        resource_def:
63          type: lib/generic_node.yaml
64          properties:
65            hostname: worker-%index%
66            image:          { get_param: image }
67            flavor:         m1.medium
68            key_name:       { get_param: key_name }
69            sec_groups:
70              - default
71              - { get_param: influx_sec_group }
72            public_net_id: { get_param: public_net_id }
73            mgmt_net:       { get_param: mgmt_net }
74            mgmt_subnet:    { get_param: mgmt_subnet }
75            manager_ip:     { get_param: manager_ip }
76            dns_ip:         { get_param: dns_ip }
```

Listing F.2: Heat mal for manager node og nettverk

```
1  # https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html#hot- ↙
       spec-template-version
2  heat_template_version: queens
3
4  description: >
5    HOT template to create a new network and router to the public network, and
6    for deploying a manager compute instance into the network. The instance is
7    assigned a floating IP address. This creates the tick.lab infrastructure,
8    with the manager working as a DNS server and Puppet master.
9
10 parameters:
11   image:
12     type: string
13     description: ID or name of image used for the compute instance(s).
14     default: CentOS 7.5 x86_64
15   flavor:
16     type: string
17     description: ID or name of flavor used for the compute instance(s).
18     default: m1.medium
19   key_name:
20     type: string
21     description: Name of keypair used to access the compute instance(s).
22   public_net_id:
23     type: string
```

```
24      description: ID or name of the public network floating IP addresses are ↙
            allocated from.
25    mgmt_net_name:
26      type: string
27      description: Name of management network to be created.
28      default: tick_mgmt_net
29    mgmt_net_cidr:
30      type: string
31      description: Management network address (CIDR notation).
32      default: 192.168.100.0/24
33    mgmt_net_gateway:
34      type: string
35      description: Management network gateway address.
36      default: 192.168.100.1
37    mgmt_net_pool_start:
38      type: string
39      description: Start of the management network IP address allocation pool.
40      default: 192.168.100.100
41    mgmt_net_pool_end:
42      type: string
43      description: End of the management network IP address allocation pool.
44      default: 192.168.100.199
45
46  resources:
47    mgmt_net:
48      type: OS::Neutron::Net
49      properties:
50        name: { get_param: mgmt_net_name }
51
52    mgmt_subnet:
53      type: OS::Neutron::Subnet
54      properties:
55        network_id: { get_resource: mgmt_net }
56        cidr:       { get_param: mgmt_net_cidr }
57        gateway_ip: { get_param: mgmt_net_gateway }
58        allocation_pools:
59          - start:  { get_param: mgmt_net_pool_start }
60            end:    { get_param: mgmt_net_pool_end }
61
62    router:
63      type: OS::Neutron::Router
64      properties:
65        external_gateway_info:
66          network: { get_param: public_net_id }
67
68    router_interface_mgmt:
69      type: OS::Neutron::RouterInterface
70      properties:
71        router_id: { get_resource: router }
72        subnet_id: { get_resource: mgmt_subnet }
73
74    manager:
75      type: OS::Nova::Server
76      properties:
77        name: manager
78        image:     { get_param: image }
79        flavor:    { get_param: flavor }
80        key_name:  { get_param: key_name }
```

```
81        networks:
82          - port:  { get_resource: manager_port }
83        user_data: { get_file: lib/manager.bash }
84
85    manager_port:
86      type: OS::Neutron::Port
87      properties:
88        network_id:      { get_resource: mgmt_net }
89        security_groups:
90          - default
91          - { get_resource: manager_ssh_sec_group }
92        fixed_ips:
93          - subnet_id:   { get_resource: mgmt_subnet }
94
95    manager_public_ip:
96      type: OS::Neutron::FloatingIP
97      properties:
98        floating_network: { get_param: public_net_id }
99        port_id:          { get_resource: manager_port }
100
101   manager_ssh_sec_group:
102     type: OS::Neutron::SecurityGroup
103     properties:
104       description: Public SSH access
105       name: ssh
106       rules:
107         - remote_ip_prefix: 0.0.0.0/0
108           protocol: tcp
109           port_range_min: 22
110           port_range_max: 22
111
112 outputs:
113   manager_mgmt_ip:
114     description: IP address of the manager instance on the management ↙
          network.
115     value: { get_attr: [manager, networks, get_param: mgmt_net_name, 0] }
116   mgmt_net:
117     value: { get_resource: mgmt_net }
118   mgmt_subnet:
119     value: { get_resource: mgmt_subnet }
```

Listing F.3: Heat mal for monolittisk node

```
1  # https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html#hot- ↙
     spec-template-version
2  heat_template_version: queens
3
4  description: >
5    HOT template to create compute instances for a distributed implementation
6    of the TICK stack.
7
8  parameters:
9    image:
10     type: string
11     description: ID or name of image used for the compute instance(s).
12     default: CentOS 7.5 x86_64
13   flavor:
14     type: string
15     description: ID or name of flavor used for the compute instance(s).
```

```
16      default: m1.medium
17    key_name:
18      type: string
19      description: Name of keypair used to access the compute instance(s).
20    public_net_id:
21      type: string
22      description: ID or name of the public network floating IP addresses are ↙
            allocated from.
23    mgmt_net:
24      type: string
25      description: Name of management network to be use.
26      default: tick_mgmt_net
27    mgmt_subnet:
28      type: string
29      description: Name of management subnet to be use.
30      default: tick_mgmt_net
31    manager_ip:
32      type: string
33      description: IP address of the manager instance on the management ↙
            network.
34    dns_ip:
35      type: string
36      description: IP address of the DNS server on the management network.
37    influx_sec_group:
38      type: string
39      description: InfluxDB and Chronograf security group
40
41  resources:
42    monostack:
43      type: lib/generic_node.yaml
44      properties:
45        hostname: monostack
46        image:          { get_param: image }
47        flavor:         m1.medium
48        key_name:       { get_param: key_name }
49        sec_groups:
50          - default
51          - { get_param: influx_sec_group }
52        public_net_id: { get_param: public_net_id }
53        mgmt_net:       { get_param: mgmt_net }
54        mgmt_subnet:    { get_param: mgmt_subnet }
55        manager_ip:     { get_param: manager_ip }
56        dns_ip:         { get_param: dns_ip }
```

Listing F.4: Heat grunnleggende mal for generisk node

```
1  # https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html#hot- ↙
       spec-template-version
2  heat_template_version: queens
3
4  description: >
5    HOT template to create a compute instance in the management network.
6
7  parameters:
8    image:
9      type: string
10     description: ID or name of image used for the compute instance(s).
11     default: CentOS 7.5 x86_64
12   flavor:
```

```
13      type: string
14      description: ID or name of flavor used for the compute instance(s).
15    hostname:
16      type: string
17      description: Name of the compute instance.
18    key_name:
19      type: string
20      description: Name of keypair used to access the compute instance(s).
21    sec_groups:
22      type: comma_delimited_list
23      description: Security groups the instances will use.
24    public_net_id:
25      type: string
26      description: ID or name of the public network floating IP addresses are ↙
            allocated from.
27    mgmt_net:
28      type: string
29      description: Name of management network to be use.
30      default: tick_mgmt_net
31    mgmt_subnet:
32      type: string
33      description: Name of management subnet to be use.
34      default: tick_mgmt_net
35    manager_ip:
36      type: string
37      description: IP address of the manager instance on the management ↙
            network.
38    dns_ip:
39      type: string
40      description: IP address of the DNS server on the management network.
41
42  resources:
43    instance:
44      type: OS::Nova::Server
45      properties:
46        name:       { get_param: hostname }
47        image:      { get_param: image }
48        flavor:     { get_param: flavor }
49        key_name:   { get_param: key_name }
50        networks:
51          - port:   { get_resource: instance_port }
52        user_data_format: RAW
53        user_data:
54          str_replace:
55            template: { get_file: node.bash }
56            params:
57              manager_ip_address: { get_param: manager_ip }
58              dns_ip_address:     { get_param: dns_ip }
59
60    instance_port:
61      type: OS::Neutron::Port
62      properties:
63        network_id:       { get_param: mgmt_net }
64        security_groups: { get_param: sec_groups }
65        fixed_ips:
66          - subnet_id:    { get_param: mgmt_subnet }
67
68    instance_public_ip:
```

```
69      type: OS::Neutron::FloatingIP
70      properties:
71        floating_network: { get_param: public_net_id }
72        port_id:          { get_resource: instance_port }
```

Listing F.5: Heat rot mal

```
1  # https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html#hot- ↙
       spec-template-version
2  heat_template_version: queens
3
4  description: >
5    HOT template that invokes subtemplates.
6
7  parameters:
8    key_name:
9      type: string
10     description: Name of keypair used to access the compute instance(s).
11   public_net_id:
12     type: string
13     description: ID or name of the public network floating IP addresses are  ↙
           allocated from.
14     default: ntnu-internal
15
16 resources:
17   influxdata_sec_group:
18     type: OS::Neutron::SecurityGroup
19     properties:
20       description: InfluxDB and Chronograf
21       name: influx
22       rules:
23         - remote_ip_prefix: 192.168.0.0/0
24           protocol: tcp
25           port_range_min: 8086
26           port_range_max: 8086
27         - remote_ip_prefix: 0.0.0.0/0
28           protocol: tcp
29           port_range_min: 8888
30           port_range_max: 8888
31
32   manager:
33     type: manager.yaml
34     properties:
35       key_name:      { get_param: key_name }
36       public_net_id: { get_param: public_net_id }
37
38   monolithic:
39     type: monolithic.yaml
40     properties:
41       key_name:      { get_param: key_name }
42       public_net_id: { get_param: public_net_id }
43       mgmt_net:         { get_attr: [manager, mgmt_net] }
44       mgmt_subnet:      { get_attr: [manager, mgmt_subnet] }
45       manager_ip:       { get_attr: [manager, manager_mgmt_ip] }
46       dns_ip:           { get_attr: [manager, manager_mgmt_ip] }
47       influx_sec_group: { get_resource: influxdata_sec_group }
48
49   distributed:
50     type: distributed.yaml
```

```
51      properties:
52        key_name:          { get_param: key_name }
53        public_net_id:     { get_param: public_net_id }
54        mgmt_net:          { get_attr: [manager, mgmt_net] }
55        mgmt_subnet:       { get_attr: [manager, mgmt_subnet] }
56        manager_ip:        { get_attr: [manager, manager_mgmt_ip] }
57        dns_ip:            { get_attr: [manager, manager_mgmt_ip] }
58        influx_sec_group: { get_resource: influxdata_sec_group }
59
60  outputs:
61    manager_public_ip:
62      description: Public IP address of the manager instance.
63      value: { get_attr: [manager, manager_floating_ip] }
```

Listing F.6: Heat variabler

```
1  parameters:
2    key_name:
```

## F.2 Cloud-init

Listing F.7: Cloud-init skript for manager node

```
#!/bin/bash -v

# correct the hostname and FQDN
echo "$(hostname -I) $(hostname -s).lab $(hostname -s)" >> /etc/hosts
hostnamectl set-hostname $(hostname -s).lab
systemctl restart systemd-hostnamed
systemctl restart network

# install puppetserver
rpm -Uvh https://yum.puppet.com/puppet5/puppet5-release-el-7.noarch.rpm
yum install -y puppetserver

# update path (this will be fixed after a reboot)
PATH=$PATH:/opt/puppetlabs/bin

# stop puppet and puppetserver in case they're already running
puppet resource service puppet ensure=stopped enable=true
puppet resource service puppetserver ensure=stopped enable=true

# configure the agent's interval and master, and enable auto-signing
puppet config set server manager.lab --section main
puppet config set runinterval 300 --section main
puppet config set autosign true --section main

# install, configure and apply r10k
puppet module install puppet-r10k
cat <<EOF > /var/tmp/r10k.pp
class { 'r10k':
        version => latest,
        sources => {
                'puppet' => {
                        'remote'  => 'https://github.com/p3lim/tick-stack-conf.git',
                        'basedir' => '/etc/puppetlabs/code/environments',
                        'prefix'  => false,
                },
        },
}
EOF
puppet apply /var/tmp/r10k.pp

# run r10k
r10k deploy environment -p

# the default puppet.service doesn't have a restart clause, we'll add it
mkdir -p /etc/systemd/system/puppet.service.d/
cat << EOF > /etc/systemd/system/puppet.service.d/restart.conf
[Service]
Restart=on-failure
RestartSec=30s
EOF
systemctl daemon-reload

# start the Puppet server and bootstrap the Puppet client
puppet resource service puppetserver ensure=running enable=true
```

```
55 puppet agent -t # request certificate
56 puppet agent -t # configure manager
57 puppet agent -t # once more to update exported resources
58 puppet resource service puppet ensure=running enable=true hasrestart=true
59
60 # manually control resolvd, setting ourselves as the nameserver
61 echo "PEERDNS=no" >> /etc/sysconfig/network-scripts/ifcfg-eth0
62 echo "search lab" > /etc/resolv.conf
63 echo "nameserver 127.0.0.1" >> /etc/resolv.conf
64 systemctl restart network
65
66 #wc notify --data-binary '{"status": "SUCCESS"}'
```

Listing F.8: Cloud-init skript for generiske noder

```
1  #!/bin/bash -v
2
3  # add manager to our hosts file ('manager_ip_address' is a template variable ↙
       for Heat)
4  echo "manager_ip_address manager.lab manager" >> /etc/hosts
5
6  # correct the hostname and FQDN
7  echo "$(hostname -I) $(hostname -s).lab $(hostname -s)" >> /etc/hosts
8  hostnamectl set-hostname $(hostname -s).lab
9  systemctl restart systemd-hostnamed
10 systemctl restart network
11
12 # install puppet
13 rpm -Uvh https://yum.puppet.com/puppet6/puppet6-release-el-7.noarch.rpm
14 yum install -y puppet-agent
15
16 # update path (this will be fixed after a reboot)
17 PATH=$PATH:/opt/puppetlabs/bin
18
19 # stop puppet in case it's already running
20 puppet resource service puppet ensure=stopped enable=true
21
22 # configure the puppet's interval and master, then run it
23 puppet config set server manager.lab --section main
24 puppet config set runinterval 300 --section main
25 puppet agent -t # request certificate
26 puppet agent -t # configure
27
28 # the default puppet.service doesn't have a restart clause, we'll add it
29 mkdir -p /etc/systemd/system/puppet.service.d/
30 cat << EOF > /etc/systemd/system/puppet.service.d/restart.conf
31 [Service]
32 Restart=on-failure
33 RestartSec=30s
34 EOF
35 systemctl daemon-reload
36
37 # enable and start puppet
38 puppet resource service puppet ensure=running enable=true
39
40 # manually control resolvd, setting the correct nameserver
41 echo "PEERDNS=no" >> /etc/sysconfig/network-scripts/ifcfg-eth0
42 echo "search lab" > /etc/resolv.conf
43 echo "nameserver dns_ip_address" >> /etc/resolv.conf
```

```
44 systemctl restart network
45
46 #wc notify --data-binary '{"status": "SUCCESS"}'
```

# G Provisjoneringskode

## G.1 Puppet kontroll-repo roller

Listing G.1: Site manifest tildeling av roller til noder

```
# Assign roles to nodes based on their FQDN.

node default {
        include profile::base
}

node 'manager.lab' {
        include ::role::manager
}

node 'master.lab' {
        include ::role::master
}

node /^worker-\d+\.lab$/ {
        include ::role::worker
}

node 'monostack.lab' {
        include ::role::monolithic
}
```

Listing G.2: Rolle manifest for manager node

```
# Manager role, for the node acting as the manager for the entire infrastructure.
# Uses the base profile, runs PuppetDB and the DNS server, and gets monitored by Telegraf.

class role::manager {
        include profile::base
        include profile::puppetdb
        include profile::dns::server
        include profile::influx::telegraf
        include profile::r10k
}
```

Listing G.3: Rolle manifest for swarm master node

```
# Master role, for nodes participating as masters in Docker Swarm.
# Uses the base profile, then starts the swarm, creates the (T)ICK stack and gets monitored by Telegraf.

class role::master {
        include profile::base
        include profile::swarm::manager
        include profile::swarm::stack
        include profile::influx::telegraf
}
```

Listing G.4: Rolle manifest for monolithic node

```
1 # Monolithic role, for the node running TICK entirely.
2 # Uses the base profile, then sets up the entire TICK stack.
3
4 class role::monolithic {
5   include profile::base
6   include profile::monotick
7 }
```

Listing G.5: Rolle manifest for swarm worker node

```
1 # Worker role, for nodes participating as workers in Docker Swarm.
2 # Uses the base profile, then joins the swarm as a worker and gets monitored ↙
      by Telegraf.
3
4 class role::worker {
5         include profile::base
6         include profile::swarm::worker
7         include profile::influx::telegraf
8 }
```

## G.2 Puppet kontroll-repo profiler

Listing G.6: Profil for grunnleggende konfigurasjon

```
# This class sets up base configuration for all nodes, including NTP and time, SSH keys and DNS

class profile::base {
        # Configure time syncronization
        class { 'ntp':
                servers  => ['ntp.ntnu.no'],
                restrict => [
                        'default kod nomodify notrap nopeer noquery',
                        '-6 default kod nomodify notrap nopeer noquery',
                ],
        }

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

        if $::hostname == 'manager' {
                # Create new SSH key so manager can act as a bastion
                sshkeys::create_ssh_key { 'centos': }

                # Add our own SSH keys so we can access the machine
                ssh_authorized_key { 'vetle key':
                        user => 'centos',
                        type => 'ssh-rsa',
                        key  => 'AAA...ZaQ==',
                }
                ssh_authorized_key { 'adrian key 1':
                        user => 'centos',
                        type => 'ssh-rsa',
                        key  => 'AAA...kWQ==',
                }
                ssh_authorized_key { 'adrian key 2':
                        user => 'centos',
                        type => 'ssh-rsa',
                        key  => 'AAA...86Q==',
                }
        } else {
                # This will add the SSH key of the manager node to every other node
                @@sshkeys::set_authorized_key { "centos@manager to centos@${::hostname}":
                        local_user  => 'centos',
                        remote_user => "centos@manager.lab",
                }

                Sshkeys::Set_authorized_key <<||>>
        }

        # Add FQDN to DNS
        include ::profile::dns::client
}
```

Listing G.7: Profil for DNS klient

```
# This class notifies the DNS server that the client is available at the given IP address.
# https://github.com/ajjahn/puppet-dns#exported-resource-patterns

class profile::dns::client {
        # Export A record for the node's hostname in the private zone
        @@dns::record::a { $::hostname:
                zone => 'lab',
                data => $::ipaddress,
        }
}
```

Listing G.8: Profil for DNS server

```
# This class sets up the DNS server for the management network.
# https://github.com/ajjahn/puppet-dns#usage
# https://github.com/ajjahn/puppet-dns#exported-resource-patterns

class profile::dns::server {
        # Install the DNS server
        include dns::server

        # Set forwarding addresses (CloudFlare DNS)
        dns::server::options { '/etc/named/named.conf.options':
                forwarders => [
                        '1.1.1.1',
                        '1.0.0.1'
                ],
        }

        # Set forward zone
        dns::zone { 'lab':
                soa         => $::fqdn,
                soa_email   => "admin.${::domain}",
                nameservers => [$::hostname],
        }

        # Collect A records from other nodes
        Dns::Record::A <<||>>

        # Open up the firewall for DNS traffic
        ::profile::firewall::management { 'DNS TCP':
                port     => 53,
                protocol => 'tcp',
        }

        ::profile::firewall::management { 'DNS UDP':
                port     => 53,
                protocol => 'udp',
        }
}
```

Listing G.9: Profil for brannmur på management nettverket

```
# This class is used to open up the firewall for the given service on the management network,
# used by other profiles to define their own rules.

```

```
4  define profile::firewall::management (
5          Variant[Integer, Array[Integer], String] $port,
6          String                                   $protocol,
7  ){
8          # Make sure the base profile is loaded
9          require ::profile::firewall
10
11         # Set up the firewall policy
12         firewall { "5 Accept service ${name} from ${net}":
13                 proto  => $protocol,
14                 dport  => $port,
15                 action => 'accept',
16                 source => '192.168.0.0/16',
17         }
18 }
```

Listing G.10: Profil for brannmur på åpent nettverk

```
1  # this class is used to open up the firewall for the given service on all networks,
2  # used by other profiles to define their own rules.
3
4  define profile::firewall::public (
5          Variant[Integer, Array[Integer], String] $port,
6          String                                   $protocol,
7  ){
8          # Make sure the base profile is loaded
9          require ::profile::firewall
10
11         # Set up the firewall policy
12         firewall { "5 Accept service ${name} from ${net}":
13                 proto  => $protocol,
14                 dport  => $port,
15                 action => 'accept',
16         }
17 }
```

Listing G.11: Profil for brannmur post

```
1  # This class finalizes the firewall rules.
2
3  class profile::firewall::post {
4          # Drop all traffic not already defined by a rule
5          firewall { '999 drop all':
6                  proto  => 'all',
7                  action => 'drop',
8                  before => undef,
9          }
10 }
```

Listing G.12: Profil for brannmur pre

```
1  # This class sets up the basic firewall rules.
2
3  class profile::firewall::pre {
4          # Wipe existing firewall rules
5          Firewall {
6                  require => undef,
7          }
8
```

```
9           # Set default firewall rules
10          firewall { '000 accept all icmp':
11                  proto  => 'icmp',
12                  action => 'accept',
13          }
14
15          firewall { '001 accept all to lo interface':
16                  proto   => 'all',
17                  iniface => 'lo',
18                  action  => 'accept',
19          }
20
21          firewall { '002 reject local traffic not on loopback interface':
22                  proto       => 'all',
23                  iniface     => '! lo',
24                  destination => '127.0.0.1/8',
25                  action      => 'reject',
26          }
27
28          firewall { '003 accept related established rules':
29                  proto  => 'all',
30                  state  => ['RELATED', 'ESTABLISHED'],
31                  action => 'accept',
32          }
33
34          firewall { '004 accept incoming SSH':
35                  proto  => 'tcp',
36                  dport  => 22,
37                  action => 'accept',
38          }
39  }
```

Listing G.13: Profil for brannmur

```
1  # This class sets up the firewall using sub-classes for pre/post  ↙
       configuration.
2  class profile::firewall {
3          # Define the before and after sub-classes
4          Firewall {
5                  before  => Class['::profile::firewall::post'],
6                  require => Class['::profile::firewall::pre'],
7          }
8
9          # Include the sub-classes
10         class { ['::profile::firewall::pre', '::profile::firewall::post']: }
11
12         # Set up the firewall
13         include firewall
14 }
```

Listing G.14: Profil for Telegraf på distribuert infrastruktur

```
1  # This class installs, configures and runs Telegraf on the given node
2
3  class profile::influx::telegraf {
4          # Get Telegraf parameters from Hiera
5          $influx_user = lookup('influx::telegraf_user')
6          $influx_pass = lookup('influx::telegraf_pass')
7
```

```
8           # Install Telegraf with custom inputs and report to InfluxDB locally
9           class { 'tick_stack::telegraf':
10                  inputs => {
11                          'cpu' => {
12                                  'percpu'   => true,
13                                  'totalcpu' => true,
14                          },
15                          'system'    => {},
16                          'mem'       => {},
17                          'net'       => {},
18                          'netstat'   => {},
19                          'processes' => {},
20                          'procstat'  => {
21                                  'pattern' => '"influx|kapa*"',
22                          },
23                  },
24                  outputs => {
25                          'influxdb' => {
26                                  'urls'      => "[\"http://master.lab:8086\"] ↙
                                        ",
27                                  'database'  => '"telegraf"',
28                                  'precision' => '"s"',
29                                  'username'  => "\"${influx_user}\"",
30                                  'password'  => "\"${influx_pass}\"",
31                          }
32                  }
33          }
34  }
```

Listing G.15: Profil for monolittisk TICK

```
1  # This class sets up TICK on a single machine.
2
3  class profile::monotick {
4    # Get Telegraf parameters from Hiera
5    $telegraf_user = lookup('influx::telegraf_user')
6    $telegraf_pass = lookup('influx::telegraf_pass')
7
8    # Install InfluxDB, Chronograf and Kapacitor using default configuration ↙
         values
9    include tick_stack::influxdb
10   include tick_stack::chronograf
11   include tick_stack::kapacitor
12
13   # Install Telegraf with custom inputs and report to InfluxDB locally
14   class { 'tick_stack::telegraf':
15     inputs => {
16       'cpu' => {
17         'percpu'   => true,
18         'totalcpu' => true,
19       },
20       'system'    => {},
21       'mem'       => {},
22       'net'       => {},
23       'netstat'   => {},
24       'processes' => {},
25       'procstat'  => {
26         'pattern' => '"influx|kapa*"',
27       },
```

```
28      },
29      outputs => {
30        'influxdb' => {
31          'urls'      => "[\"http://127.0.0.1:8086\"]",
32          'database'  => '"telegraf"',
33          'precision' => '"s"',
34          'username'  => "\"${telegraf_user}\"",
35          'password'  => "\"${telegraf_pass}\"",
36        }
37      }
38    }
39  }
```

Listing G.16: Profil for PuppetDB

```
1  # This class sets up the PuppetDB instance.
2  # https://forge.puppet.com/puppetlabs/puppetdb
3
4  class profile::puppetdb {
5          # Configure PuppetDB using the default parameters.
6          include puppetdb
7          include puppetdb::master::config
8
9          # Accept communication on TCP port 8140 for PuppetDB
10         ::profile::firewall::management { 'PuppetDB TCP':
11                 port     => 8140,
12                 protocol => 'tcp',
13         }
14 }
```

Listing G.17: Profil for r10k

```
1  # This class sets up the r10k service, which pulls the configuration from
       the public repository
2  # on GitHub.
3
4  class profile::r10k {
5          # Configure r10k to pull from GitHub to the local environments
               directory
6          class { 'r10k':
7                  version => latest,
8                  sources => {
9                          'puppet' => {
10                                 'remote'  => 'https://github.com/p3lim/tick-
                                       stack-conf.git',
11                                 'basedir' => '/etc/puppetlabs/code/
                                       environments',
12                                 'prefix'  => false,
13                         },
14                 },
15         }
16
17         # Configure r10k to pull changes every 30 minutes
18         cron { 'r10k':
19                 command => '/bin/r10k deploy environment -p',
20                 user    => 'root',
21                 minute  => '*/30',
22         }
23 }
```

Listing G.18: Profil for Docker Swarm mester

```
# This class initiates a Docker Swarm cluster.

class profile::swarm::manager {
        # Install Docker first
        include 'docker'

        # Create the swarm
        docker::swarm { 'cluster':
                init           => true,
                advertise_addr => $::ipaddress,
                listen_addr    => $::ipaddress,
                before         => File['/etc/puppetlabs/facter/facts.d/ ↙
                    swarm_token.sh'],
        }

        # Create the parent directory for Facter facts
        file { ['/etc/puppetlabs/facter', '/etc/puppetlabs/facter/facts.d']:
                ensure => 'directory',
                owner  => 'root',
                group  => 'root',
        }

        # Define script that will act as a custom Facter fact, containing  ↙
           the
        # worker token for the swarm
        $fact_content = '#!/bin/bash
echo "swarm_token=$(docker swarm join-token worker -q)"
'

        # Create script
        file { '/etc/puppetlabs/facter/facts.d/swarm_token.sh':
                owner   => 'root',
                group   => 'root',
                content => $fact_content,
                mode    => '0755',
                require => File['/etc/puppetlabs/facter/facts.d'],
        }

        # Configure firewall to accept Docker Swarm TCP and UDP ports
        ::profile::firewall::management { 'Docker Swarm Manager TCP':
                port     => [2376, 2377, 7946],
                protocol => 'tcp',
        }

        ::profile::firewall::management { 'Docker Swarm Manager UDP':
                port     => [7946, 4789],
                protocol => 'udp',
        }
}
```

Listing G.19: Profil for Docker Swarm TICK stack

```
# This class creates and maintains the ICK portion of the TICK stack
# in a Docker Swarm.

class profile::swarm::stack {
        # Copy the templated stack configuration to the node
        file { '/tmp/ick-stack.yaml':
```

```
7                 ensure  => file,
8                 content => epp('profile/ick-stack.yaml.epp'),
9         }
10
11        # Create/maintain the stack from the configuration file
12        docker::stack { 'ick':
13                stack_name    => 'ick',
14                compose_files => ['/tmp/ick-stack.yaml'],
15                require       => File['/tmp/ick-stack.yaml'],
16        }
17
18        # Open up the firewall for all ports required by ICK
19        ::profile::firewall::management { 'InfluxDB TCP':
20                port     => 8086,
21                protocol => 'tcp',
22        }
23
24        ::profile::firewall::management { 'Kapacitor TCP':
25                port     => 9092,
26                protocol => 'tcp',
27        }
28
29        ::profile::firewall::public { 'Chronograf TCP':
30                port     => 8888,
31                protocol => 'tcp',
32        }
33 }
```

Listing G.20: Profil for Docker Swarm arbeider

```
1  # This class joins workers into the Docker Swarm.
2
3  class profile::swarm::worker {
4         # Install Docker first
5         include 'docker'
6
7         # Get facts about the swarm master
8         $master = puppetdb_query('inventory[facts] {facts.trusted.certname ~ ↙
              "master"}')
9
10        # Join the swarm using the master's IP and swarm worker token
11        docker::swarm { 'cluster':
12                join           => true,
13                advertise_addr => $::ipaddress,
14                listen_addr    => $::ipaddress,
15                manager_ip     => $master[0]['facts']['ipaddress'],
16                token          => $master[0]['facts']['swarm_token'],
17        }
18
19        # Configure firewall to accept Docker Swarm TCP and UDP ports
20        ::profile::firewall::management { 'Docker Swarm Worker TCP':
21                port     => [2376, 7946],
22                protocol => 'tcp',
23        }
24
25        ::profile::firewall::management { 'Docker Swarm Worker UDP':
26                port     => [7946, 4789],
27                protocol => 'udp',
28        }
```

```
}
```

Listing G.21: Mal for Docker Swarm stack

```
version: '3'

services:
  influxdb:
    image: influxdb:1.7-alpine
    hostname: influxdb
    volumes:
      - influxdb-data:/var/lib/influxdb
    ports:
      - 8086:8086 # HTTP API
    environment:
      INFLUXDB_DB: <%= lookup('influx::db_name') %>
      INFLUXDB_HTTP_AUTH_ENABLED: 'true'
      INFLUXDB_ADMIN_USER: <%= lookup('influx::admin_user') %>
      INFLUXDB_ADMIN_PASSWORD: <%= lookup('influx::admin_pass') %>
      INFLUXDB_USER: <%= lookup('influx::telegraf_user') %>
      INFLUXDB_USER_PASSWORD: <%= lookup('influx::telegraf_pass') %>
    networks:
      - influx
    deploy:
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role != manager]

  kapacitor:
    image: kapacitor:1.5-alpine
    hostname: kapacitor
    volumes:
      - kapacitor-data:/var/lib/kapacitor
    ports:
      - 9092:9092 # HTTP API
    environment:
      KAPACITOR_INFLUXDB_0_URLS_0: http://influxdb:8086
      KAPACITOR_INFLUXDB_0_USERNAME: <%= lookup('influx::admin_user') %>
      KAPACITOR_INFLUXDB_0_PASSWORD: <%= lookup('influx::admin_pass') %>
    networks:
      - influx
    deploy:
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role != manager]
    depends_on:
      - influxdb

  chronograf:
    image: chronograf:1.7-alpine
    volumes:
      - chronograf-data:/var/lib/chronograf
    ports:
      - 8888:8888 # Web UI
    environment:
      INFLUXDB_URL: http://influxdb:8086
      INFLUXDB_USERNAME: <%= lookup('influx::admin_user') %>
```

```
56        INFLUXDB_PASSWORD: <%= lookup('influx::admin_pass') %>
57        KAPACITOR_URL: http://kapacitor:9092
58        KAPACITOR_USERNAME: <%= lookup('influx::admin_user') %>
59        KAPACITOR_PASSWORD: <%= lookup('influx::admin_pass') %>
60      networks:
61        - influx
62      deploy:
63        restart_policy:
64          condition: on-failure
65        placement:
66          constraints: [node.role != manager]
67      depends_on:
68        - influxdb
69        - kapacitor
70
71  volumes:
72    influxdb-data:
73    kapacitor-data:
74    chronograf-data:
75
76  networks:
77    influx:
```

## G.3 Puppet kontroll-repo diverse

Listing G.22: Hiera konfigurasjon

```
---
version: 5

defaults:
  datadir: hieradata   # path relative to this file
  data_hash: yaml_data # use the built-in YAML backend

hierarchy:
  - name: "Per-node data"
    path: "nodes/%{trusted.certname}.yaml" # file path, relative to datadir

  - name: "Common data"
    path: "common.yaml"
```

Listing G.23: Hiera eksempeldata

```
---
influx::db_name: telegraf
influx::admin_user: admin
influx::admin_pass: adminadminadminadmin
influx::telegraf_user: telegraf
influx::telegraf_pass: metricsmetricsmetricsmetrics
```

Listing G.24: Puppetfil som definerer avhengigheter

```
forge 'http://forge.puppet.com'

mod 'puppet-r10k', '6.8.0'
mod 'puppetlabs-stdlib', '5.2.0'
mod 'puppetlabs-translate', '1.2.0'
mod 'puppetlabs-apt', '6.3.0'

mod 'saz-timezone', '5.1.1'
mod 'stm-debconf', '2.3.0'

mod 'puppetlabs-ntp', '7.4.0'
mod 'puppetlabs-puppetdb', '7.1.0'
mod 'puppetlabs-inifile', '2.5.0'
mod 'puppetlabs-postgresql', '5.12.1'
mod 'puppetlabs-firewall', '1.15.1'

mod 'ajjahn-dns',
  :git    => 'https://github.com/jearls/puppet-dns',
  :branch => 'fix_rfc1912_zone_handling'
mod 'puppetlabs-concat', '5.2.0'

mod 'puppet-sshkeys',
  :git    => 'https://github.com/jtopjian/puppet-sshkeys',
  :commit => 'fc5273a'
mod 'dalen-puppetdbquery', '3.0.1'

mod 'puppetlabs-docker', '3.3.0'
mod 'puppetlabs-powershell', '2.2.0'
mod 'puppetlabs-reboot', '2.1.2'

mod 'tick_stack',
```

```
32    :git    => 'https://github.com/vetletm/tick_stack',
33    :commit => 'de5e50a'
```

# H Scripts

Listing H.1: Script for å kjøre stress-test

```
#!/bin/bash

# run 5 tests for 1 hour each with an hour spacing
for i in 1 2 3 4 5; do
        # run the test as fast as possible for an hour
    /usr/local/src/go/bin/influx-stress insert \
        -f -r 1h --host "http://${DB_URL}:8086" \
        --user "$DB_USER" --pass "$DB_PASS"

    # wait an hour between each full run, let resources normalize
    sleep 3600
done
```

Listing H.2: Script for å kjøre spike-test

```
#!/bin/bash

# run 5 tests for 1 hour each with an hour spacing
for i in 1 2 3 4 5; do
    # set runtime for each test to 1 hour
    runtime=3600

    # as long as the runtime is a valid value
    while ((runtime > 0)); do
        # set the spike interval to a random number between 1 and 10
        spike=$(((RANDOM % 10) + 10))
        runtime=$((runtime - spike))

        # run the test as fast as possible for the specified length
        /usr/local/src/go/bin/influx-stress insert \
            -f -r "${spike}s" --host "http://${DB_URL}:8086" \
            --user "$DB_USER" --pass "$DB_PASS"

        # set a random waiting time between each run
        waittime=$((RANDOM % 10) + 1)
        runtime=$((runtime - vent))

        sleep $waittime
    done

    # wait an hour between each full run, let resources normalize
    sleep 3600
done
```