CYBERINGENIØRSKOLEN

BACHELOROPPGAVE 2024

Kandidatens navn: Aleksander Firing Martinsen og Frode Molland Sanden

Oppgavens tittel: Interface for presentasjon av data fra mange kilder

Oppgavens tekst:

Mange virksomheter har i dag et driftsmiljø bestående av servere, klienter og Kubernetes Clustere som beskrives av konfigurasjonsfiler i Git. I dag presenteres tilstanden til serverene i en enkel CMDB (Configuration Management Database). Vi ønsker å få presentert konfigurasjon og faktisk tilstand på hele driftsmiljøet på en oversiktlig måte. Med det på plass kan avvik og problemer avdekkes raskere, og feilsøking vil forenkles.
Oppgaven går konkret ut på å utforske muligheter for presentasjon av utgitt datasett for tilstand og konfigurasjon av driftsmiljøet. Vi ser etter en løsning som helst bruker open-source verktøy og er enkelt å videreutvikle.
Oppgaven vil ta for seg følgende deler:

- Gjøre seg kjent med datasettet og forstå behovet for å få oversikt
- Undersøke løsninger for presentasjon av datasettet
- Lage en PoC (Proof of Concept) løsning

Oppgaven gitt: 2024-01-04
Besvarelsen levert: 2024-16-05
Omfang: 20 Studiepoeng
Utført ved: Forsvarets Høgskole/Cyberingeniørskolen

Jørstadmoen, 2023.12.24

Den som er faglærer (fra CISK)
Kapteinløytnant
Petter Fornebo

[Denne siden er blank med hensikt]

Sammendrag

I denne bacheloroppgaven har vi utforsket og analysert hvordan en bedrift kan samle inn data fra mange kilder og distribuere informasjonen om infrastrukturen, slik at brukerne enkelt kan få informasjonen om ressursene de bruker. I dagens IT-miljø deler ofte flere avdelinger samme fysiske infrastruktur for å utføre oppgaver. Dette gjør at kun IT-personell, som styrer sikkerheten, har full oversikt over infrastrukturen. Ved tilby en distribuert informasjon om systemer til brukerne kan man raskere oppdage problemer, slik at handlinger kan håndteres raskt, og man forebygger nedetid.

Vi har utviklet og testet en multi-tenant dashboard-løsning for å samle og presentere data fra mange kilder. Studiens hovedmål var å finne ut hvordan man kan samle inn data fra mange kilder og distribuere informasjonen om infrastrukturen, slik at brukerne enkelt kan få informasjonen om ressursene de bruker. Gjennom studien har vi vurdert og testet open-source verktøy som InfluxDB, Grafana og Zabbix, samt muligheten for en egenlaget løsning med rammeverket React.

Forskningen avdekket at Zabbix-applikasjonen tilbyr et robust grunnlag for å utvikle en fleksibel løsning som både er open-source, og støtter multi-tenancy. Vi beviste gjennom utvikling av en Proof-of-Concept (PoC) løsning at vi kunne håndtere og differensiere data på tvers av flere brukere (tenants). For å kunne implementere en integrasjon mot ”mange kilder” satte vi krav til at noen parametere måtte defineres av de som benytter løsningen.

Konklusjonene fra oppgaven viser at det er fullt mulig å bruke Zabbix, sammen med script for datainnsamling og tilføring av parametere, for presentasjon av data fra mange kilder, slik at brukerne enkelt kan få informasjon om ressursene de bruker. Løsningen vår kan prosessere data fra mange ulike kilder, ved at den støtter mottak av 12 ulike filformater. Den presenterer data i Zabbix på tekstformat. Løsningen vår tilbyr også en enkel måte å sette opp Kubernetes Cluster overvåking.

Oppgaven identifiserer videre arbeid og forbedringer som er nødvendige for å implementere PoC løsningen i en praktisk kontekst.

Forord

Oppgaven er skrevet av Aleksander Firing Martinsen og Frode Molland Sanden. Ved innlevering av denne bacheloroppgaven har vi fullført tre spennende og krevende år på Cyberingeniørskolen. Arbeidet med denne oppgaven har vært utfordrende og intensivt, men først og fremst veldig spennende og givende. Skolen har tilrettelagt for et godt samarbeid med oppdragsgiver som har vært svært verdifullt. Vi vil takke de ansatte på skolen som har gitt oss gode forutsetninger for å skrive oppgaven. Vi vil rette en spesiell takk til professor Siv Hilde Houmb ved skolen, som har hjulpet oss å finne et aktuelt intervjuobjekt og delt sine perspektiver. Temaet for oppgaven har vært helt nytt for oss, noe som har gitt oss mye motivasjon og læring.

Vi vil takke vår veileder som har hjulpet oss kontinuerlig gjennom hele oppgaven og gitt oss relevante tips og bidratt med sin kompetanse. Vi vil også takke Seniorrådgiver Eirik Skomakerstuen i Waveit for å ha delt sine erfaringer i et intervju, og for en veldig god samtale. Til slutt vil vi rette en takk til faglærer Petter Fornebo for tilbakemeldinger underveis i oppgaveskrivingen.

Ordforklaring

Anomalideteksjon	Oppdage mønstre som avviker fra det normale
API	Application Programming Interface
Autorisasjonsbærer	Autentiseringsmekanisme spesielt brukt for HTTP-baserte API kall
Back-end	Den delen av en applikasjon som ikke er synlig for brukeren
Cluster	Samling noder som jobber sammen for å levere applikasjoner og tjenester
CSS	Cascading Style Sheet, brukes til a beskrive presentasjonen av et dokument skrevet i HTML
Deployment	Prosessen med å utrulle eller distribuere en applikasjon eller programvare til et produksjonsmiljø
Front-end	Den delen av en applikasjon/tjeneste som sørger for interaksjon med brukeren
GUI	Grafisk brukergrensesnitt, den delen av programvaren som tillater brukeren a kommunisere ved bruk av grafiske elementer
HTML	HyperText Markup Language, et tekstformat for utforming av websider
IoT	Internet of Things
Item	Zabbix; en spesifikk enhet for datainnsamling og overvåking
JSON	JavaScript Object Notation, dataformat for semistrukturert hierarkiske datamodeller
JSON-RPC	JSON Remote Procedure Call, protokoll for utførelse av prosedyrekall fra klient til server ved hjelp av JSON-formaterte meldinger
Kontainere	Virtuelle miljøer som gjør det mulig å pakke, distribuere og kjøre applikasjoner på en konsistent måte uavhengig av infrastruktur
Metrics	Informasjon om dataobjekter
Micro-rammeverk	Lettvektig og enkelt rammeverk som ikke tilbyr all funksjonalitet ”out-of-the-box”
Open-source	Programvare eller prosjekter der kildekode er tilgjengelig for offentlig bruk
Pip	Package installer for Python
Plugins	Tileggsmoduler som utvider funksjonaliteten til programvarer eller tjenester
PoC	Proof of Concept
Pod	Kubernetes minste distribusjonsenhet, inneholder en eller flere kontainere
Private Cloud	Type skyinfrastruktur som er dedikert til bruk av en organisasjon
Properties	Egenskaper som beskriver et objekt
SaaS	Software-as-a-Service, leveransemodell der applikasjoner leveres over nettet som en tjeneste
Scraper	Automatisk hente ut og lese informasjon fra websider eller web-endepunkter
Services	Programprosesser på en enhet
Tenant	Bruker eller leietaker av et system/tjeneste/applikasjon
Test-data	Eksempeldata for bruk under testing
Widget	Et lite grafisk element som utfører en bestemt funksjon eller viser spesifikk informasjon

[Denne siden er blank med hensikt]

Innholdsfortegnelse

[Denne siden er blank med hensikt]

Liste over figurer

1 Introduksjon

1.1 Bakgrunn

Moderne militære styrker er helt avhengig av digital teknologi, og effektiv anvendelse av informasjons- og kommunikasjonsteknologi (IKT) vil være en av de største driverne for økt operativ evne og virksomhetsstyring i årene som kommer [1]. Riksrevisjonens undersøkelse av Forsvarets informasjonssystemer viser at det foreligger alvorlige mangler. Riksrevisjonen peker på at det er mangler i oversikt og dokumentasjon på IKT-området som påvirker muligheten for å ivareta sikkerheten i informasjonssystemene. Videre skriver de: «En grunnleggende forutsetning for at Forsvaret skal kunne foreta gode risikovurderinger og planlegge og gjennomføre effektive sikkerhetstiltak, er at virksomheten har god oversikt over informasjonssystemer og tilhørende kommunikasjonsinfrastruktur» [2]. Det foreligger altså et stort behov for god oversikt over informasjonsinfrastrukturen.

Det står beskrevet i Forsvarets fellesoperative doktrine at «det økende digitaliseringspresset i samfunnet gjelder også det operative området» [3]. Det er altså meget relevant å utforske og følge det sivile samfunnets IKT-utvikling og tilpasse dette til Forsvarets behov. I denne oppgaven velger vi å fokusere på erfaringer og perspektiver fra det sivile, og oversikt i deres IT-landskap.

Det finnes mange ulike verktøy en virksomhet kan benytte for å få hensiktsmessig oversikt over sitt IT-landskap. Mange av disse løsningene kan være særdeles kostbare. Det finnes ikke én universell standardløsning som passer for alle bedrifter. IT-landskapet varierer i stor grad mellom ulike virksomheter, dette betyr at de ulike virksomhetenes IT-landskap består av forskjellige ressurser og en rekke ulike kilder. Denne variasjonen er også innad i en virksomhet. Store IT selvskaper tilbyr IT-løsninger og tjenester som de kontrollerer. Når du distribuerer kontrollen over tjenesten til andre, kan de by på utfordringer innen tilpasningsmuligheter og fleksibilitet. Det foreligger derfor et behov for en egen allsidig og fleksibel løsning som kan presentere data fra mange kilder.

I dagens IT-miljø deler ofte flere avdelinger samme fysiske infrastruktur for å utføre oppgaver. Dette gjør at kun IT-personell, som styrer sikkerheten, har full oversikt over infrastrukturen, mens ansatte i de ulike avdelingene har begrenset innsikt i ressursene de bruker. Utfordringen er å tilby de ulike avdelingene og kundene en oversikt over deres ressurser, uten at det går ut over sikkerheten til infrastrukturen. Ved å distribuere informasjon om virksomhetens ressurser kan man oppnå en proaktiv problemhåndtering ved å tilby overvåkning av ressursstatus og gi tilgang til nødvendig konfigurasjonsdata for gruppens ressurser. Dette tillater gruppen og oppdage problemer og utføre handlinger raskt som bidrar til å forebygge nedetid og fokus på gruppens arbeid. Det er derfor et behov for en løsning som gir brukere tilgang til sin del av infrastrukturen. En såkalt ”multi-tenacy” dashboard-lønsing vil bidra til å adskille dataene logisk fra hverandre og presentere data for de ulike tenantsene i systemet [4].

1.2 Problemanalyse og -formulering

1.2.1 Hovedmål / Problemformulering

Hvordan kan man samle inn data fra flere kilder og distribuere informasjonen om infrastrukturen, slik at brukerne enkelt kan få informasjonen om ressursene de bruker?

1.2.2 Forskningsspørsmål / problemstilling

- Hvilke krav stilles til løsningen?

Forskningsspørsmålet skal belyse hva løsningen skal tilby og hvilken fleksibilitet løsningen skal ha.

- Hvordan håndtere data fra mange kilder?

Forskningsspørsmålet har som mål å finne ut hvordan man kan håndtere datainnsamling, prosessering og fremvisning av data fra mange kilder. Forskningsspørsmålet belyser også en praktisk betydning av multi-tenancy.

- Hvilke open-source verktøy med støtte for multi-tenancy egner seg for å presentere data fra mange kilder?

Med bakgrunn i de belyste kravene til løsningen, skal det presenteres ulike open-source verktøy som kan egne seg for å løse problemet. Spørsmålet sammenligner ulike verktøy og hvordan de egner seg som en løsning til kravene vi har satt til løsningen. Det gjøres en vurdering av ulike verktøy og perspektiver fra fagfolk for å velge det mest egnede verktøyet som det kan lages en Proof-of-Concept (PoC) løsning til.

1.2.3 Avgrensinger

Vi har avgrenset oppgaven til å kun rette seg mot tradisjonell infrastruktur som man finner i det sivile. Dette er fordi militær informasjonsinfrastruktur ofte har spesialløsninger og kompliserer mange aspekter ved oppgaven, samtidig ville oppgaven vært vanskelig å skrive på UGRADERT. Det er meget omfattende å implementere og vurdere en løsning som kan presentere data fra alle mulige kilder, derfor er oppgaven avgrenset til å utforske typiske kilder i en infrastruktur som består av servere, klienter og Kubernetes Clustere. Oppgaven er avgrenset til å utforske utelukkende open-source verktøy, da det ikke er satt av midler til eksterne kostnader i prosjektets økonomiske ramme, samt at det gir en fordel med hensyn til fleksibilitet.

1.3 Leserveiledning

Oppgaven retter seg mot tredjeårs kadetter ved Cyberingeniørskolen og antar at leseren besitter kompetanse og forståelse på et nivå som samsvarer med dette. Dokumentet er delt inn i 6 hoveddeler og har 1. vedlegg.

Teori

Dette kapitlet inneholder relevant teori for oppgaven og som går utenfor den forventende kompetansen til leseren.

Metode

Dette kapitlet gjennomgår metoden som ble benyttet for å besvare forskningsspørsmålene og det overordnede problemet. Kapitlet inneholder beskrivelse for hver del av prosessen.

Resultater

Dette kapitlet presenterer resultatene fra de ulike metodene som ble anvendt. Vi har definert kravene til løsningen og vurdert de ulike mulighetene som ble identifisert. Basert på denne gjennomgangen, har vi valgt den løsningen som oppgaven er basert på.

Design og utvikling

I dette kapitlet beskriver vi utviklingsprosessen til løsningen. Her legges den generelle ideen for løsningen fram, samt hvordan vi løste problemet i praksis. Kapitlet tar for seg forskningsspørsmål 3 og beskriver hvordan løsningen er laget. Grunnet omfanget til kodebasen for løsningen, beskrives utformingen av koden grovt i kapittel 5.6, mens koden i sin helhet kan ses i vedlegg A.

Diskusjon

Diskusjonen er delt inn i 5 deler, der hver del diskuteres og drøftes. kapitlet baserer seg i hovedsak på legitimiteten til løsningen og hvordan en slik løsning kan anvendes i praksis, samt om metodene som ble valgt er gode nok til å besvare forskningsspørsmålene.

Konklusjon

Dette kapitlet tar for seg hovedpunktene fra hoveddelen av oppgaven, hvilke resultater vi har og slutninger fra diskusjonen. Vi svarer på forskningsspørsmålene til oppgaven og presenterer forslag til videre arbeid.

[Denne siden er blank med hensikt]

2 Teori

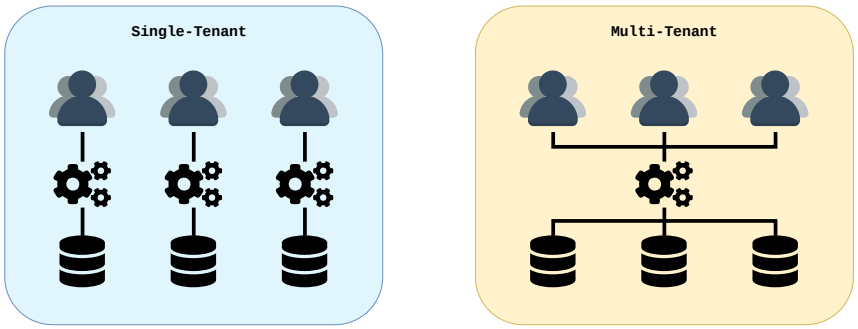
2.1 Multi-Tenant Software as a Service

Multi-tenant software as as service (SaaS) representerer en arkitekturmodell innenfor skytjenester hvor en enkelt programvareapplikasjon og dens underliggende infrastruktur tjener flere kunder eller tenants [5]. I den pågående "skyreisen" er det en økende tendens til å bevege seg bort fra modellen der hver avdeling eller organisasjon vedlikeholder sin egen dedikerte infrastruktur og servere. I stedet ønsker mange å dra nytte av tredjeparts skytjenesteleverandører som kan tilby skalerbare og skreddersydde løsninger. Denne overgangen fra egen infrastruktur til skybaserte tjenester gir organisasjoner betydelige fordeler [4].

En virksomhet som leverer tjenester, følger samme prinsipper som SaaS. I en "private cloud", vil behovet for sentralisert infrastruktur og sentraliserte databaser og tjenester sikre en leverende avdeling kontroll over brukere. Man kan tilpasse ressurstilgangen til brukere, og enkelt skalere opp eller ned tjenesten ved behov [6].

2.1.1 Grunnleggende Prinsipper

I en multi-tenant arkitektur deler tenants en felles kodebase og databaser, selv om dataene er logisk isolert på en slik måte at de ikke kan aksesseres av andre tenants. Basert på denne tilnærmingen krever det nøye utformede datamodeller og tilgangskontrollmekanismer for å sikre sikkerhet, konfidensialitet og tilgjengelighet [7][8].



Figur 1: Forskjellen mellom single- og multi-tenant arkitektur. I multi-tenant benytter brukerne samme applikasjon/tjeneste, bare deres data er logisk adskilt

Figur 1 viser prinsippet og forskjellen mellom single-tenant og multi-tenant. Som vist, i et multi-tenant miljø, benytter flere brukere samme applikasjon/tjeneste (vist som tannhjul i figur 1), mens deres data er logisk isolert fra hverandre. I en single-tenant arkitektur, vil alle brukerne benytte sine separate applikasjoner/tjenester, og deres data vil fysisk være plassert i forskjellige databaser.

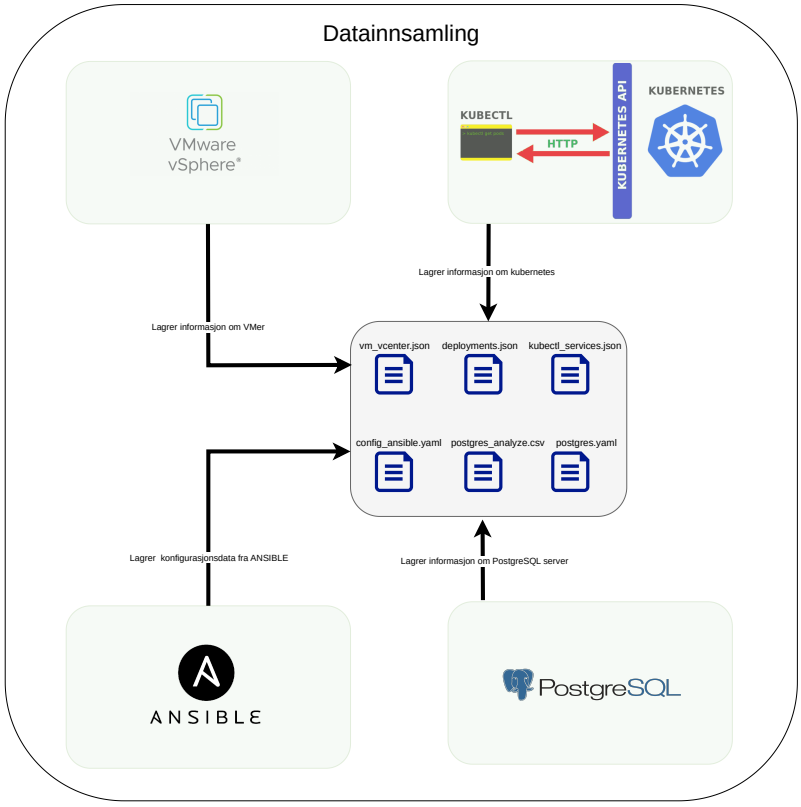
Kompleksiteten i en slik arkitektur vil avhenge av tjenesten. Å skille data mellom brukere, tilegne data til riktig brukere og sørge for sikkerhet og komplett isolering mellom brukere, er alle essensielle egenskaper til en slik arkitektur.

2.2 Datatyper

2.2.1 Statiske data

I denne oppgaven brukes uttrykket ”statiske data” for å beskrive data lagret i en lokal fil, som har en struktur som enten er strukturert eller semistrukturert. Dette inkluderer data som genereres når du ber en tjeneste om å lage en fil som for eksempel beskriver dens konfigurasjon eller tilstand. Statiske data kan også være tidsseriedata, basert på hvilket tidsintervall du henter data. Selv om innholdet i en fil kan endre seg når en ny versjon genereres, vil den opprinnelige filen forbli uendret. Disse statiske dataene brukes til å gi informasjon om IT-infrastrukturen.

Figuren under skal illustrere hva som menes med statiske data og er et eksempel på hvordan disse statiske dataene hentes.



Figur 2: Innsamling av data fra forskjellige kilder lagret som statiske filer for videre bruk.

Figur 2 viser hvordan vi genererer filer som inneholder informasjon om infrastrukturen. Disse filene inneholder den faktiske tilstanden eller konfigurasjonen til infrastrukturen da

de ble generert. Den statiske dataen som filene inneholder, kan brukes til presentasjon. Filene egner seg godt for å utvikle en integrasjon mot et grensesnitt, ettersom det muliggjør prøving og feiling.

2.2.2 Tidsseriedata

Tidsseriedata er sorterte sekvenser av verdier av en variabel målt ved et likt tidsintervall [9]. Et eksempel på tidsseriedata kan være temperatur målt fra en temperaturmåler. La oss si måleren måler temperaturen hvert 10. minutt, da vil resultatet fra disse målingene defineres som tidsseriedata. Dette er fordi det er en verdi som måles med lik antall tid mellom hver måling. På samme måte kan tidsstemplede data fra loggfiler hos endesystemer være tidsseriedata. Målingene forteller at verdiene er sortert i en tidslinje, og dermed kan historikken til målingene hentes i form av grafer, der x eller y aksen viser tid. Ofte vil endringer i rekkefølgen av tidsseriedata føre til at målingene ikke gir mening. Tidsseriedata gjør det mulig å oppdage trender og anomalier i målingene. Dersom man måler CPU temperatur, kan plutselige økninger i denne temperaturen antyde at et problem har oppstått.

2.2.3 Metrics

Metrics er målinger eller kvantitative data som brukes til å evaluere ytelse, effektivitet eller andre aspekter av et system, prosess eller tjeneste. Innenfor IT kan metrics omhandle nedetid, responstider, feilrate eller brukerengasjement. Ved å måle metrics vil man få direkte innblikk i et systems helse, samt kunne oppdage oppdukkende feil eller anomalier i systemets ytelse [10].

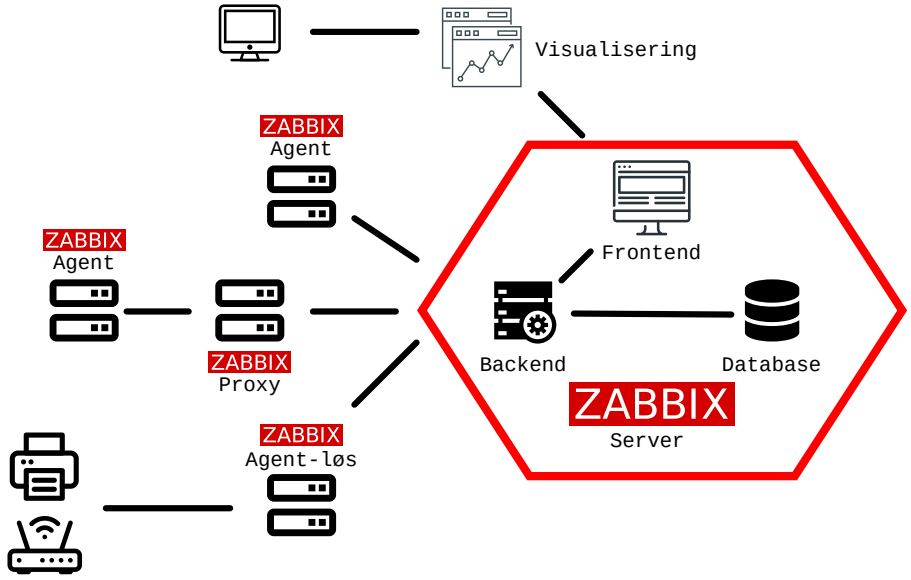
2.2.4 Strukturert- og semistrukturert data

Strukturert data er data som følger en bestemt struktur, vanligvis i form av tabeller med rader og kolonner. Denne organiseringen av dataene gjør det enkelt å lagre, søke, hente og analysere dataene. Den vanligste formen for strukturert data er lagret i relasjonelle databaser. Disse databasene organiserer dataene i tabeller, der hver rad representerer en forekomst av dataene og hver kolonne representerer en attributt eller egenskap av denne forekomsten [11].

Semistrukturert skiller seg fra strukturert data i den form at det ikke følger en forhåndsdefinert modell. Denne typen data organiseres på en mer fleksibel måte, ofte i form av hierarkiske strukturer som JSON og YAML [11].

2.3 Zabbix

Zabbix er et open-source overvåkningsverktøy for IT-infrastruktur utviklet og driftet av Zabbix LLC. Programvaren tilbyr en rekke funksjonaliteter og er anerkjent som et av de mest allsidige overvåkningsverktøyene på markedet. Hovedfunksjonen til Zabbix ligger i sanntidsovervåking av status til ulike nettverksenheter, servere, klienter, virtuelle maskiner og tjenester. Nøkkelfunksjoner inkluderer datainnsamling, problemoppdagelse, visualisering og varslingssystemer [12][13].



Figur 3: Zabbix arkitektur, bestående av server, datainnhenting-mekanismer og visualisering for brukeren.

Figur 3 viser oppsettet og samhandlingen mellom de ulike komponentene som kreves for overvåking via Zabbix. Som vist består arkitekturen av to hovedkomponenter, Zabbix-server og data-innsamlingsmekanismer.

2.3.1 Zabbix Server

Zabbix er et avansert innhentingsystem og krever derfor en rekke roller for å kunne driftes optimalt. For at Zabbix skal kjøre effektivt som et monitorerings-/overvåkingssystem, krever det en kombinasjon av flere nøkkelkomponenter som arbeider sammen. Disse komponentene er essensielle for å samle inn, prosessere, lagre, og presentere data [14].

- **Zabbix Server:** Serveren er hjertet i tjenesten, den har ansvaret for å motta innhentet data, for så å evaluere forholdene, identifisere hvor dataen hører til, generere varsler og lagre dataene i databasen. Serveren utfører all logikk som ligger bak for at tjenesten skal fungere [15]

- **Database:** Databasen lagrer konfigurasjonsinformasjon, historikk over datainnsamling og operasjonslogger. Zabbix støtter en rekke databasetyper som MySQL, PostgreSQL, SQLite, Oracle, og IBM DB2. Valg av databasetype avhenger av driftsmiljøets omfang, fremtidige behov og skaleringsmuligheter [16].
- **Zabbix Front-End:** Front-End er det webbaserte grensesnittet som tillater brukeren å konfigurere tjenesten, overvåke i form av visualisert data og grafer, karttjenester, definere terskler for varsling og administrere systemvarsler. Front-end kommuniserer med Zabbix-serveren og henter data fra databasen for å presentere informasjon til brukeren. Den er skrevet i PHP og krever en webserver som Apache eller Nginx for å fungere [17].
- **Zabbix Proxy:** Proxy er et mellomledd i datainnsamlingen. Den samler inn data fra en eller flere enheter og videresender dette til serveren. Bruk av proxy er nyttig i større driftsmiljøer for å minimere belasting på serveren samt minke nettverkstrafikk. For tjenester som Kubernetes-overvåking spiller også proxy en viktig rolle. Proxier kan også hjelpe med overvåking av nettverk som er segmentert, adskilt eller isolert da man kun trenger å åpne for kommunikasjon mellom proxy og server [18].

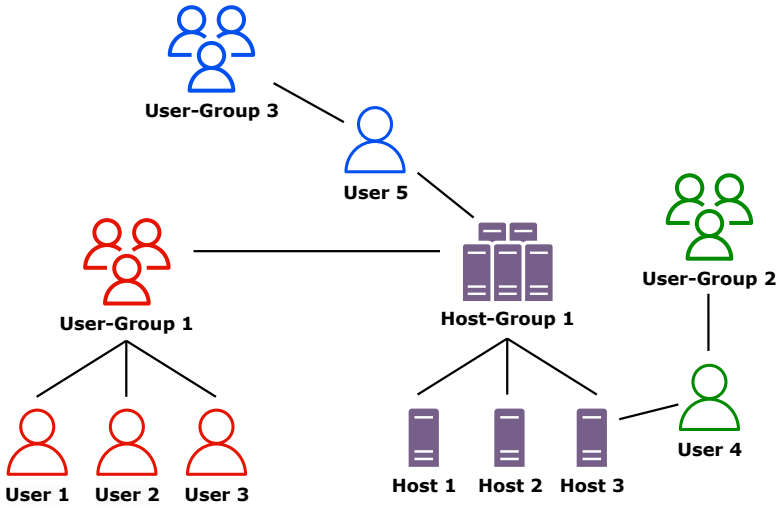
2.3.2 Datainnsamling

Zabbix tilbyr datainnsamling igjennom en rekke ulike metoder, noe som sikrer god fleksibilitet i ulike driftsmiljøer.

- **Agentbasert overvåkning:** Her er en Zabbix agent installert på enheten som skal overvåkes. Agenten samler inn detaljert informasjon om systemets operativsystem, programvareappliksjoner og systemressurser [19].
- **Agentløs overvåking:** For enheter der installasjon av Zabbix Agent ikke er mulig, kan Zabbix utfør agentløs innhenting ved å benytte standardprotokoller som SNMP, JMX og IPMI. Dette muliggjør overvåking av f.eks nettverksenheter og servere [20].
- **Brukerdefinert Script:** Administratorer kan utvikle egendefinerte script for datainnhenting. Dette utvider Zabbix funksjonalitet og gir enda større tilpasningsmuligheter.

2.3.3 Brukerhåndtering

Zabbix har en logisk inndeling av systemer og brukere. Tjenesten baseres på Hosts og Users. Hosts er endesystemer, eller enheter som skal overvåkes. Disse tilhører en Host-group, som er en samling Hosts. Users er sluttbrukere. Overordnet har også Users inndeling i grupper. Users og User-groups kan begge gis tilganger til Hosts og Host-groups [21].

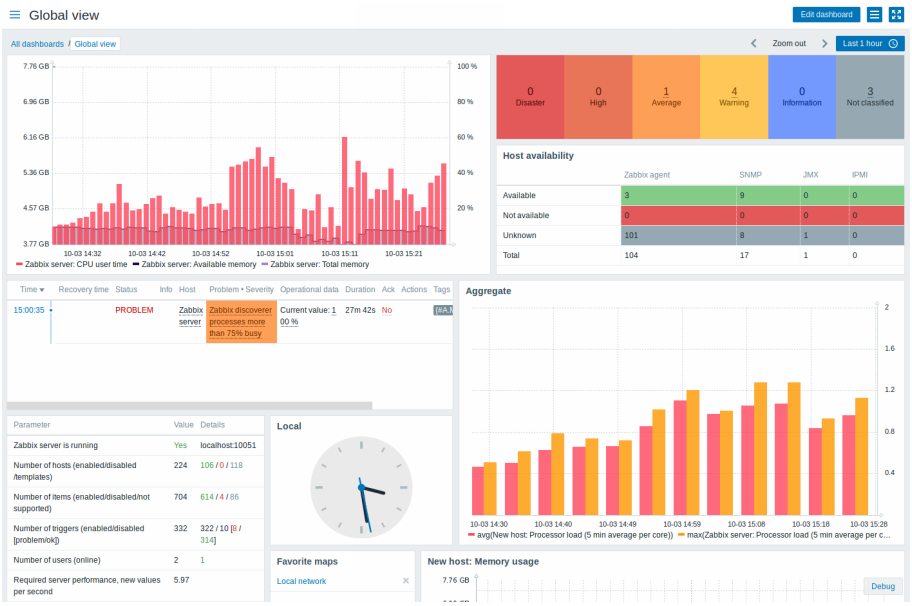


Figur 4: Users og Hosts i Zabbix. Brukere kan gis tilgang til objekter via gruppen de tilhører, eller spesifikke tilganger.

I figur 4 illustreres tilganger hos brukere og brukergrupper i Zabbix. Som vist kan både brukere og brukergrupper ha tilgang til Host-groups, samtidig som andre brukere kan ha tilgang til individuelle Hosts innad i Host-groupen. Basert på denne arkitekturen, er det få begrensinger på tilgangsstyring innad i tjenesten, og alle som har behov for tilgang, har mulighet til å få det.

2.3.4 Visualisering

Det webbaserte grensesnittet implementert i Zabbix tilbyr mye funksjonalitet for visualisering, som vist i figur 5. Forhåndsdefinerte templates kan hente informasjon fra databasen og opprette grafer og rapporter som kan vises i brukerdefinerte dashboard. På denne måten kan brukere og administratorer ha tilgang til sin data og lage egendefinerte dashboard slik de ønsker [22].



Figur 5: Eksempel på dashboard i Zabbix. Dashboardet demonstrerer mulighetene man har med visualisering i Zabbix, inkludert grafer, klokker og tabeller

Dashboard er også tilgangsstyrt, og både Users og User-groups kan gis tilgang til de ulike dashboardene.

2.3.5 Zabbix-API

Zabbix-API tilbyr en kraftig og omfattende måte å interagere med Zabbix-serveren for å hente informasjon, lage eller oppdatere konfigurasjoner, og automatisere ulike oppgaver. APIet benytter JSON-RPC (Remote Procedure Call) protokoll, som tillater utviklere å utføre API-kall ved å sende JSON-formaterte forespørsler over HTTP/HTTPS [23].

```
1  import requests
2
3  link = "http://192.168.0.2/api_jsonrpc.php"
4
5  data = {
6          "jsonrpc": "2.0",
7          "method": "user.login",
8          "params": {
9              "username": "Admin",
10             "password": "zabbix"
11         },
12         "id": 1
13         }
14
15 resp = requests.post(link, headers = {"content-type": "application/json
       -rpc"}, json=data)
```

Koden over er en JSON-RPC for å anskaffe en autorisasjonsbærer. Autorisasjonsbærer

kreves for hver kommando som skal utføres, og må dermed anskaffes på forhånd av hver RPC.

2.3.6 Trapper Items

Zabbix Trapper Item er en type monitoreringselement innad i Zabbix som tillater asynkron sporadisk innsamling av data. I motsetning til de fleste andre datainnsamling typer i Zabbix, venter Trapper Item på at data skal bli sendt til den. Dette gjør Trapper Item spesielt nyttig for sporadiske eller hendelsesdrevne data, som ikke nødvendigvis er tidsbestemt eller regelmessig. Trapper Item fungerer ved å lytte til innkommende data til Zabbix Serveren. Når data mottas, blir de behandlet og lagret av Zabbix, akkurat som data samlet inn via den konvensjonelle metoden [24].

```
zabbix_sender -z <server-ip> -p <port> -s "host-navn" -k <nøkkel> -o "Informasjon"
```

Kommandoen over benytter Zabbix_sender til å sende informasjon til Zabbix Trapper Item [25]. Dette er metoden som må benyttes for å sende informasjon til Trapper Item.

2.4 InfluxDB

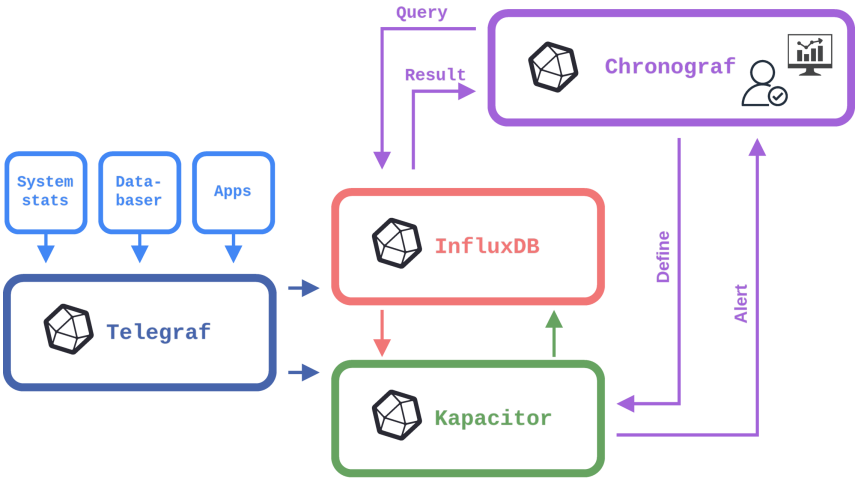
InfluxDB er en open-source tidsdatabasessystem designet for å håndtere tidsstemplet data. Et tidsdatabasesystem er optimalisert for høy lese- og skriveytelse [9], noe som gjør den ideell for applikasjoner som involverer store mengder tidsbestemte data. InfluxDB tilbyr effektiv lagring og spørring av store volum tidsdatabasert informasjon. Spørringen baserer seg på InfluxQL, som ligner SQL, og som sørger for lett dataanalyse og henting.

2.4.1 InfluxDATA Tick Stack

InfluxDB er en komponent i utvikleren InfluxData, sin såkalte Tick Stack. Stacken tilbyr en komplett løsning for innsamling, lagring, visualisering og overvåking av tidsbasert informasjon og systemer [26].

- **Telegraf:** Telegraf er InfluxData sin egne datainnsamlingsagent og sørger for innhenting og videresending av alle målinger og hendelser. Telegraf kan installeres og kjøres på det meste av databaser, systemer og IoT-sensorer [27].
- **InfluxDB:** Tidsseriedatabase for lagring av data.
- **Chronograf:** Chronograf er InfluxData sitt visualiseringsverktøy med et tilhørende grensesnitt for brukeren. Tjenesten sørger for oppsett av dashboard, administrasjon og håndtering av alarmer innad i hele stacken [28].

- **Kapacitor:** Kapacitor er hjernen i stacken. Den er hendelsesorientert og sørger for alarmering og anaomalideteksjon. Kapacitor analyserer resultater og informerer videre til brukeren [29].



Figur 6: Relasjoner i InfluxDB stacken. Telegraf henter informasjon og sender videre til Kapacitor og InfluxDB, Chronograf sørger for visualisering og alarmering.

Figur 6 viser hvordan de ulike komponentene i stacken samarbeider, som vist er det Telegraf som henter og sender videre informasjon fra det som overvåkes. Dette lagres deretter i InfluxDB og behandles av Kapacitor. InfluxDB og Kapacitor sender ikke automatisk det de har blitt tilsendt, men heller venter på forespørsler fra Chronograf som deretter fremlegger resultatet på brukerens ønskede måte [30].

2.4.2 Datainnsamling

InfluxDB tilbyr en rekke metoder for datainnsamling.

- **Telegraf:** Telegraf er en plugin-basert server som overvåker og sender informasjon videre til InfluxDB og Kapacitor. I hovedsak benyttes Telegraf på servere, større systemer og ulike IoT-sensorer. Telegraf kompileres til binært slik at den kan kjøre på alle systemer uten eksterne avhengigheter. Ettersom prosjektet er open-source finnes det over 300+ offentlige plugins til tjenesten, og tilbyr overvåking av det meste man kan tenke seg [27].
- **Client Libraries:** InfluxDB tilbyr egenskrevne biblioteker for innhenting av data fra selvdefinerte kilder. Tjenesten benytter InfluxAPI til å sikre at databasen mottar det som sendes til den. Ved å benytte Client Libraries vil man kunne selv definere hva som skal hentes og overvåkes, i tillegg til hva som skal sendes, og hvor mye [31].
- **Scrapers:** Scrapers er en innhentingsmetode som henter data tilgjengelig via åpne HTTP endepunkter [32].

- **Ecosystem:** Mottar data og hendelser fra tredjepartsapplikasjoner direkte inn i InfluxDB [33].

2.4.3 Brukerhåndtering

I InfluxDB styres brukerhåndtering av såkalte organisasjoner. Alt av data, brukere, dashboard, oppgaver, hendelser og overvåkede systemer hører til en organisasjon. Hver organisasjon utgjør en logisk adskilt enhet, og alt som tilhører den er isolert fra andre organisasjoner. Dette gjør det mulig å administrere tilgang til data ved å tilordne brukere til organisasjonen som dataene tilhører [34]. Brukere i InfluxDB kan ikke tildeles individuelle tilganger til spesifikke ressurser eller data. I stedet tildeles de overordnede tilganger til enten alle ressurser innenfor organisasjonen de tilhører, eller de kan ekskluderes fra organisasjonen helt.

2.4.4 Visualisering

For visualisering og brukergrensesnitt benyttes Chronograf. Dashboard kan tilpasses etter brukerens behov for å vise både sanntidsdata og data som er lagret i InfluxDB. Chronograf er designet for visualisering av tidsdatabaser og tilbyr et web-basert grensesnitt [28][35].

2.4.5 API

InfluxDB tilbyr et eget API som muliggjør interaksjon direkte med databasen. APIet er HTTP basert, og svarene kommer i JSON-format. APIet er tilknyttet databasen og dermed består funksjonene av */query*, */ping*, */get*, */write* og */health* [36].

```
curl -XPOST "localhost:8086/api/v2/write?bucket=db/rp&precision=s" \
    -H 'Authorization: Token <username>:<password>' \
    --data-raw "mem,host=host1 used_percent=23.43234543 1556896326"
```

koden over viser et eksempel på en spørring i form av en endring i minnebruk hos et endesystem. Man kan se at linken */api/v2/write?bucket* velger hva som skal utføres, og videre benyttes *-H* til å velge headers i API-kallet [37].

2.5 Grafana med Prometheus

Grafana, utviklet av Raintank, er et open-source verktøy for data visualisering og overvåking. Grafana er spesielt kjent for sin evne til å lage detaljerte interaktive dashboard som visuelt representerer sanntidsdata fra ulike kilder [38]. I hovedsak baserer tjenesten seg på å visualisere data fra tidsdatabaser, SQL-databaser og ulike skytjenester. Grafana muliggjør høy grad av tilpassing og fleksibilitet for brukeren, noe som har gjort det til et populært verktøy de siste årene [39][40].

2.5.1 Grafana Loki

Grafana Labs er utviklerne bak Grafana, og tilbyr en rekke verktøy med ulike bruksområder. Grafana er primært et verktøy for visualisering og overvåking, men det finnes også andre verktøy Grafana Labs har som retter seg mot andre bruksområder. Et av disse er Grafana Loki. Grafana Loki er et høyt tilgjengelig aggregeringssystem for logger. Loki tilbyr en multi-tenant arkitektur for kostnadseffektiv lagring av logger. I motsetning til andre loggsystemer, lagrer Loki bare etiketter av informasjonen og ikke all teksten mottatt i loggen. Denne metoden å lagre logger på gjør tjenesten effektiv, både i form av tidsbruk og kostnader. Tjenesten er i hovedsak designet for å tilby en mer strømlinjeformet og kostnadseffektiv løsning for loggbehandling, som er lett å integrere med Grafana for enkel anomalideteksjon og oppdagelse av problemer [41].

Som nevnt støtter Grafana Loki multi-tenancy. I Loki defineres bruker som tenant og har ingen øvrige inndelinger. Skille mellom tenants skjer i spørringene til Loki, der et eget *tenant id* felt definerer hvilke tenant spørringen hører til. Spørringene og dataen fra tenant A vil ikke være aksesserbart for Tenant B, altså det oppstår et logisk skille mellom dem [42].

2.5.2 Datainnsamling

Ettersom hovedfokuset til Grafana ligger i visualisering og dashboardløsninger, tilbyr Grafana i seg selv få metoder for datainnhenting. Grafana Agent er Grafanas egne datainnhentings verktøy, og tilbyr innsamling, aggregering og sending av metrics og loggdata til diverse tjenester. Selv om Grafana kan motta data direkte fra Grafana Agent, er det vanlig praksis å benytte et tredjeparts program for mottak av data. Grunnen til dette, er Grafanas mangel på overvåkningsfunksjonalitet og langtidslagring av tidsseriedata. Når det gjelder loggdata og logger, er Grafana Agent konfigurert til å sende disse direkte til Grafana Loki [43].

Grafana kombineres gjerne med Prometheus, et overvåknings- og tidsdatabasesystem [44], som gir en mer komplett leveranse av overvåking og analyse. Prometheus er spesialisert innen innsamling og lagring av metrics i sanntid, og selv om det kan virke som det er overlapp i funksjonalitet mellom de to, er de designet for å jobbe sammen [45]. Ettersom Grafana er open-source, finnes det mange flere metoder og programmer som tilbyr en komplett overvåking og visualisering i kombinasjon med Grafana [39].

2.6 React

React er et open-source JavaScript-bibliotek for utforming og bygging av grensesnitt, da spesielt dynamiske webapplikasjoner. Biblioteket er utviklet og vedlikeholdes av META og har med tiden blitt ett av de mer populære verktøyene for front-end utvikling. Hovedfunksjonen til React er dens mulighet til å endre deler av siden, enten det er data eller ut-

forming, uten å måtte laste hele siden på nytt. Dette bidrar til en mer flytende og interaktiv opplevelse for brukeren. Kjernen i React sin popularitet ligger i dens komponentbaserte arkitektur. Tilnærmingen basere seg på å bygge komponenter som gjenbrukes, noe som gjør det enklere å håndtere den økene kompleksiteten i moderne webapplikasjoner [46].

2.6.1 Kjernekonsepter i React

React har introdusert en rekke innovative konsepter og prinsipper. De mest sentrale konseptene er den komponentbaserte arkitekturen, JavaScript XML og måten React håndterer data [47].

- **Komponentbasert Arkitektur:** React basere hele sin arkitektur på ideen om komponenter. En komponent defineres i React som et uavhengig, gjenbrukbart stykke brukergrensesnitt som tilsvarer en del av et større visuelt oppsett. Komponenter i seg selv kan være enkle, som en knapp eller et felt for tilførsel av tekst, eller komplekse som en hel applikasjonsform. Denne tilnærmingen gjør vedlikehold og oppdatering enklere, da brukergrensesnittet deles inn i isolerte deler [48].
- **JSX (Javascript XML):** JSX er en syntax utvidelse for JavaScript som tillater brukeren å skrive HTML-elementer direkte i JavaScript koden. JSX er ikke obligatorisk å benytte, men forbedrer lesbarheten og simplifiserer koden ved å kombinere de to språkene sammen. JSX-kode kompileres til vanlig JavaScript, noe som gjør det mulig for nettleseren å tolke og vise elementene [47].
- **State og Props:** State og props er to konsepter som bidrar til å håndtere dataflyten innad i en React-applikasjon. State representerer deler av komponenten som kan endre seg over tid, altså en slags tilstand. Et eksempel på state kan være byggeår på et bil-objekt. Endringer i State fører til at komponenten lastes på nytt for å reflektere de nye verdiene [49]. Props, eller Properties, er en måte å sende data fra en overordnet komponent til en underordnet, og muliggjør gjenbruk av komponenter med ulikt innhold [50][47].

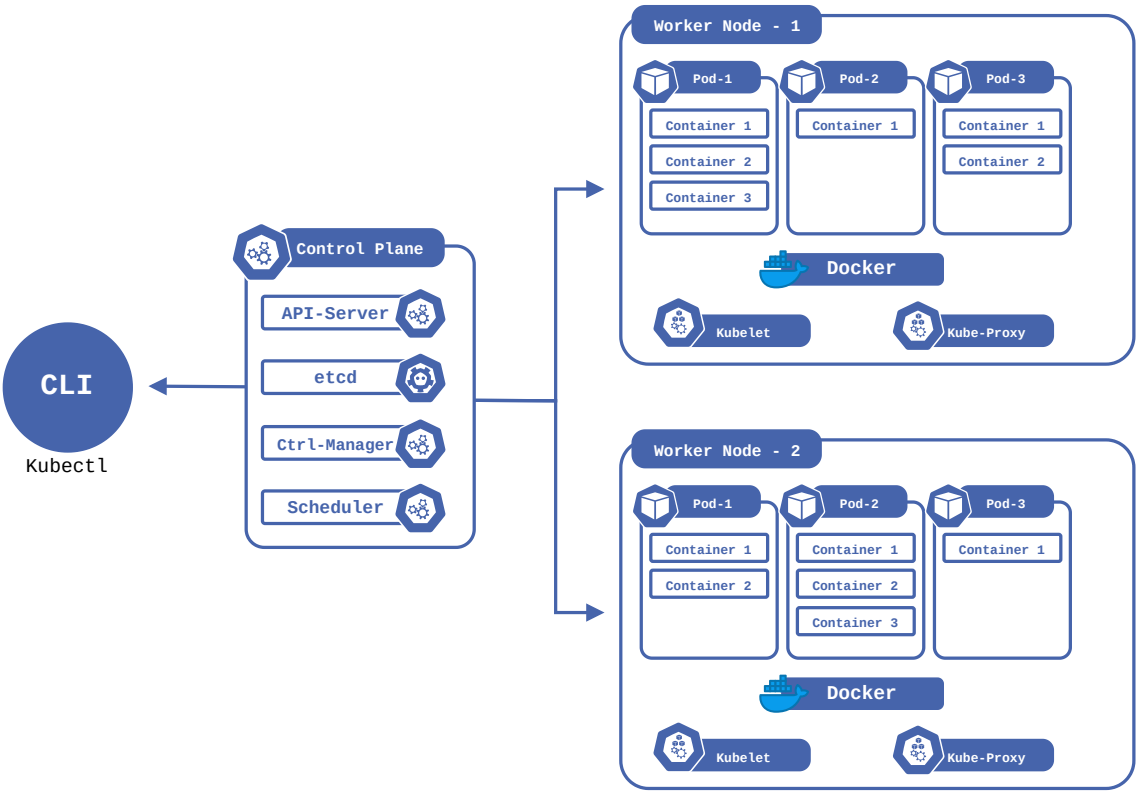
2.6.2 React-økosystemet

Ettersom React er open-source, har det blitt utarbeidet et stort og variert økosystem av tilleggsverktøy, biblioteker og praksiser. Dette økosystemet utvider funksjonaliteten til det grunnleggende biblioteket ved å implementere ny funksjonalitet som tilstadsforvaltning, ruting, API, testing, utviklerverktøy og forhånds konfigurerte UI-komponenter. Dette gir utviklere fleksibilitet til å velge løsninger som passer best for seg, samtidig som det drar nytte av React sitt kraftfulle fundament for bygging av webapplikasjoner [51] [52].

2.7 Kubernetes

Kubernetes er en open-source orkistreringsplattform for kontainere. Plattformen er utviklet av Google, eies av Cloud Native Computer Foundation (CNCF), og muliggjør automatisert distribusjon, skalering og administrasjon av conteiniserte applikasjoner og tjenester. I Kubernetes betegnes Clustere som en samling av noder (fysiske eller virtuelle maskiner som utgjør infrastrukturen) som jobber sammen for å levere applikasjoner og tjenester [53].

Kubernetes følger en Clusterbasert arkitektur. Et Kubernetes Cluster består av minst en master-node (Control-plane), som styrer og koordinerer Clusteret, og flere worker-noder, hvor applikasjoner og tjenester kjøres [53].



Figur 7: Kubernetes Arkitektur bestående av noder, control plane og CLI. Virtualiseringen innenfor nodene gjøres med Docker.

Figur 7 illusterer et typisk Kubernetes Cluster og komponentene det inneholder. Sentralt i Clusteret ligger Control-Plane (Kontrollplanet), som er ansvarlig for å styre og koordinere hele Clusteret. Kontrollplanet består av flere nøkkelkomponenter:

- **API-server:** Denne komponenten er ansvarlig for å motta og håndtere forespørsler fra brukeren og andre systemer.
- **etcd:** Dette er en database som brukes av Kubernetes til å lagre viktig informasjon om Clusteret.

- **Scheduler:** Denne komponenten er ansvarlig for å plassere podder på de riktige nodene basert på ressurskrav og andre faktorer.
- **Ctrl-Manager:** Overvåker Clusteret og sikrer at det opprettholder ønsket tilstand. Den administrerer forskjellige aspekter av Clusteret for å sikre at alt fungere som det skal.

Alle disse rollene jobber sammen for å orkestrere både noder og podder i Clusteret. En node kan inneholde flere podder, og hver pod kan inneholde en eller flere kontainere som kjører forskjellige applikasjoner [54]. Ofte benyttes Docker for virtualisering av kontainerene, da dette gir gode muligheter for isolasjon og ressursadministrasjon mellom dem [55].

2.7.1 Helm

Helm er en pakkebehandler for automatisk deployering og håndtering av Kubernetes applikasjoner. Tjenesten lar brukeren definere installere og oppgradere selv de mest kompliserte Kubernetes applikasjoner. Prosjektet er open-source og driftes av Helm Community etter sin uteksaminering fra CNCF i 2020 [56][57].

2.7.2 Overvåking

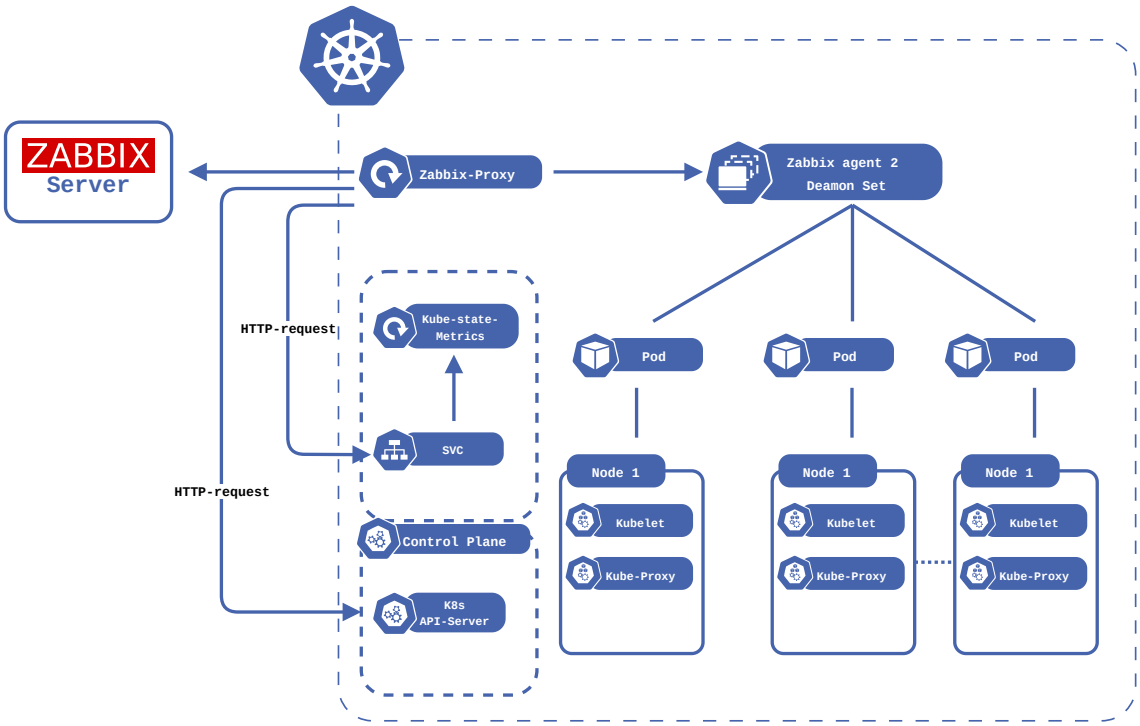
Monitorering av Kubernetes Clustere er avgjørende for å sikre applikasjoners helse, opprettholde forventet ytelse, tilgjengelighet og sikkerhet. Effektiv overvåking gir utviklere og driftspersonell innsikt i Clusterets helse og bidrar til rask identifikasjon om potensielle problemer. Hva som skal overvåkes avhenger av Clusterets omfang, behov og intensjon [58][53].

- **Cluster-ressurser:** Overvåking av CPU, lagring, minne og nettverksressurser for både master og worker-noder gir informasjon om Clusterets generelle helse og ressursbruk/ytelse.
- **Podder og kontainere:** Monitorering av individuelle podder og kontainere hjelper driftspersonell å indentifisere oppdukkende problemer i applikasjoner.
- **Tjenester og ingress:** Overvåking over nettverkstrafikk, responstid, feilrater og nettverksbruk for tjenester eksponert utenfor Clusteret.

2.7.2.1 Zabbix

Kubernetes-overvåking i Zabbix baserer seg på å depolyere agenter på hver node i Clusteret. Agentene benytter cAdvisor, som er forhåndsinstallert i hver node, til å hente metrics om

nodens tilstand og ressursbruk. I tillegg til cAdvisor, utnytter Zabbix Kube-State-Metrics som er en sideliggende-applikasjon som lytter til Kube-API og genererer metrics om objektstatuser [59].



Figur 8: Zabbix Kubernetes-overvåking. Zabbix Server kommuniserer med en Zabbix Proxy innenfor noden. Proxyen mottar informasjon fra agenter i hver pod.

Figur 8 viser overordnet hvordan overvåking av Kubernetes Clustere gjennomføres i Zabbix. All kommuikasjon til Zabbix-Server vil skje igjennom en Zabbix-Proxy innad i Clusteret. Proxyen mottar informasjon fra agentene og videresender dette til serveren. I tillegg vil proxyen være konfigurert til å samle metrics fra både Kube-State-Metrics og Clusterets API-server. Metoden krever en servicekonto innad i Clusteret for at Zabbix-modulene skal tilegnes riktige tilganger. [60].

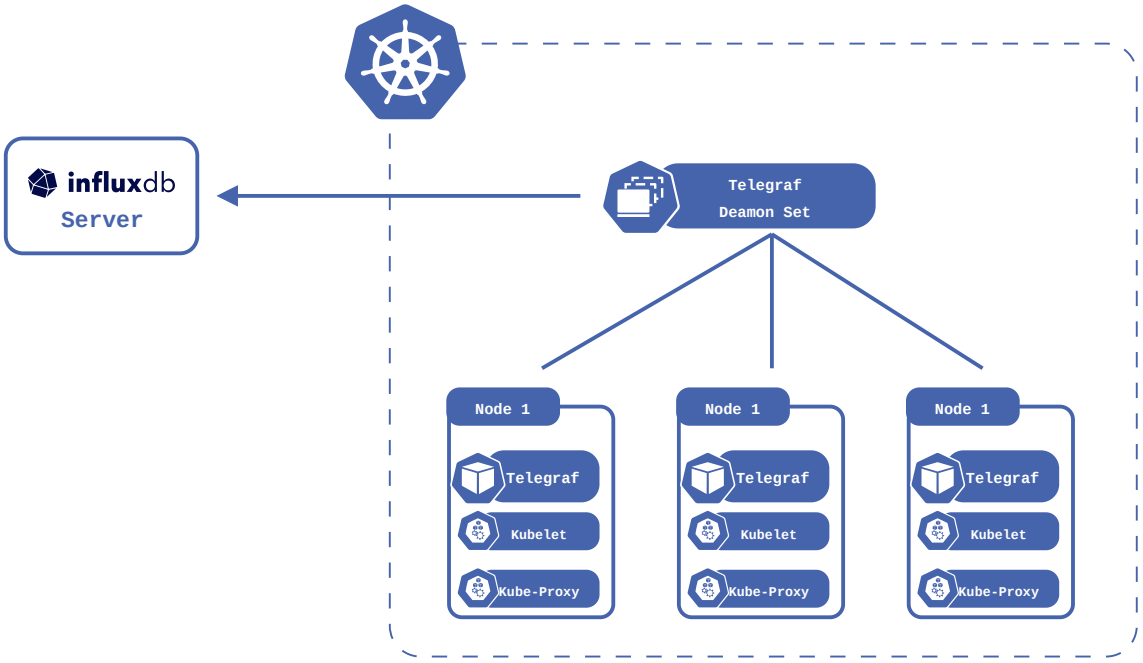
Selv om arkitekturen for overvåkningen kan virke intrikat og komplisert, finnes det løsninger som forenkler oppsettet. Ved å benytte Zabbix-Helm-chart vil man ved en kommando kunne deployere agenter, proxy, servicekonto og alt nødvendig for å fullstendig overvåke Clusteret [61]. Innad i Zabbix, finnes det templates for overvåkingen som muliggjør enkel og automatisert oppsett av Cluster-overvåkingen.

2.7.2.2 InfluxDB

Kubernetes-overvåking i InfluxDB kan gjøres på forskjellige måter basert på hva som er ønskelig å overvåke og omfanget av overvåkingen [62].

Telegraf deployert i hver node: Som metoden benyttet i Zabbix kan agenter deployeres i

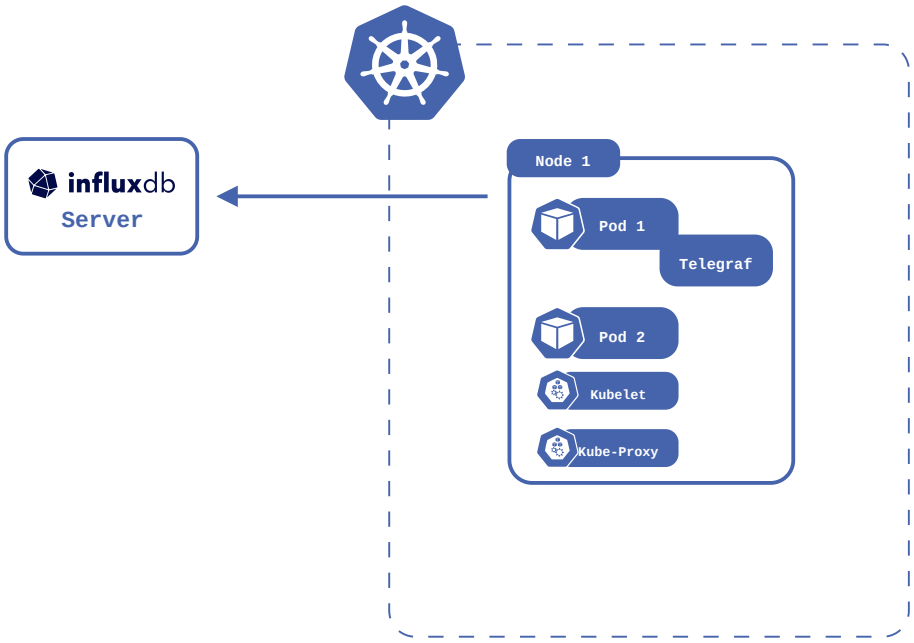
hver Kubernetes node. På denne måten kan Telegraf hente data fra noden, podder i noden, kontainere og applikasjoner via push-pull mekanismer [62].



Figur 9: Telegraf agent deployert i hver node. Her kjører Telegraf i sin egen pod i noden.

Som figur 9 viser, har man ved å benytte denne metoden deployert en Telegraf instans i hver node. Denne metoden blir ofte benyttet der det er behov for enkle overvåking av verdier innad i hver pod og node.

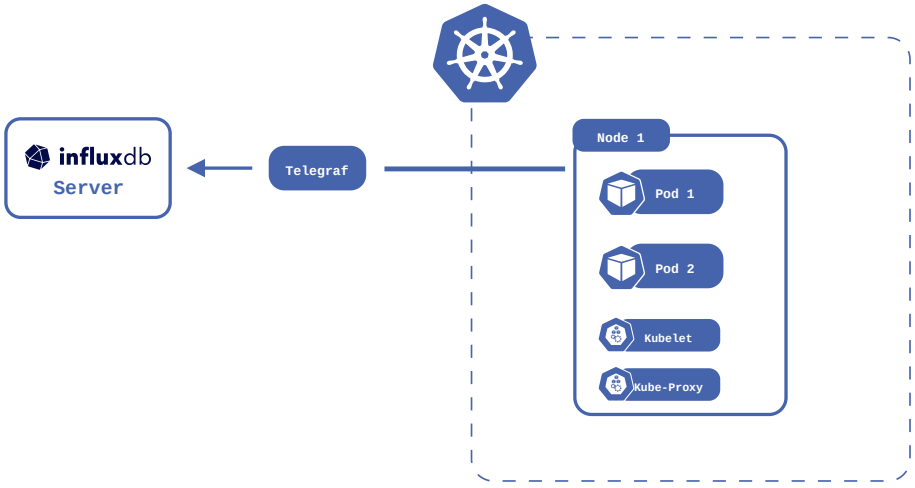
Telegraf deployert som sidecar: Ved instanser der det er ønskelig å monitorere poder, ved å isolere omfanget av overvåkingen kan man deployere Telegraf som en sidecar til poden [62]. Med sidecar menes at kontaineren som kjører Telegraf, kjører sammen med den opprinnelige kontaineren i samme [63].



Figur 10: Telegraf deployert som sidecar. Her kjører Telegraf i samme pod som den ønsker å monitorere

Figur 10 viser at Telegraf vil være sidestilt med kontaineren, og dermed befinne seg på samme nettverk og i samme pod. På denne måten kan Telegraf hente informasjon om tilstanden og brukeren har mer kontroll over hvilke metrics som er ønskelig å overvåke [62].

Telegraf deployert som sentral agent: Telegraf støtter oppdagelse av tjenester ved å identifisere endepunkter for Kubernetes metrics innenfor et Cluster. På denne måten vil Telegraf søke igjennom podden etter de åpne endepunkten, scrape disse og sende videre til InfluxDB.



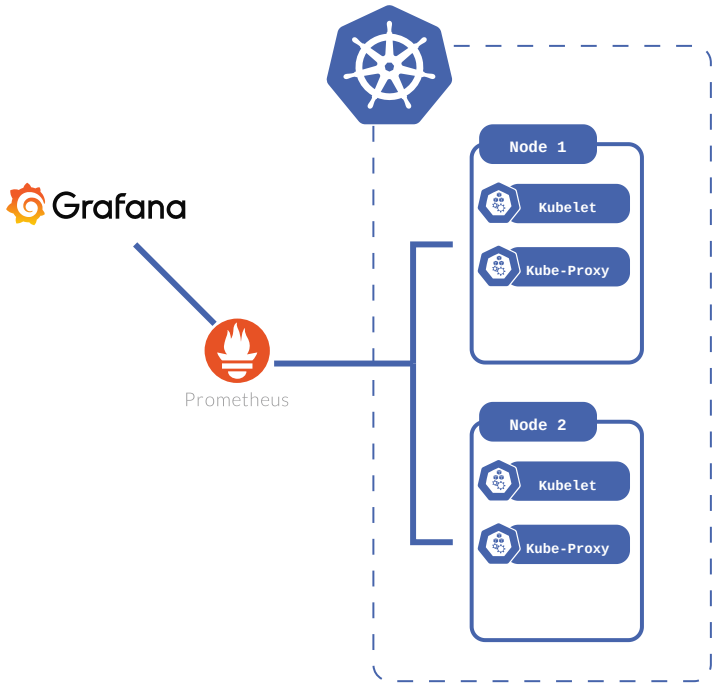
Figur 11: Telegraf deployert som sentral agent. Her kjører Telegraf utenfor Clusteret og med det har ikke interne tilganger.

Som vist i figur 11 vil Telegraf her være utenfor Clusteret, og heller lytte på endepunkter på innsiden. Ved å benytte denne metoden vil man kunne monitorere alle applikasjoner med et åpent */metrics* endepunkt. Telegraf henter data fra endepunktet og sender resultatet tilbake til InfluxDB.

På samme måte som for Zabbix, finnes det Helm-Charts for installasjon av de to øvrige metodene, mens den siste ikke krever konfigurasjon innad i Kubernetes, utenom åpning av */metrics* endepunktet for de modulene man ønsker å overvåke [64].

2.7.2.3 Grafana

For Grafana blir det sjelden benyttet Kubernetes overvåking uten tredjeparts applikasjoner som gjennomfører innhentingen. Den vanligste metoden er å benytte Prometheus som agent, for deretter visualisere resultatet i Grafana. Prometheus i seg selv er svært kompatibel med Kubernetes i den form at det er spesielt designet for dynamiske konteinermiljøer [65]. Prometheus samler inn metrics via de åpne endepunktene i applikasjonene, for deretter å lagre disse og tilby dataene til Grafana [66][45].



Figur 12: Utformingen til Kubernetes-overvåkning i Grafana. Prometheus scraper nodenes åpne endepunkter for metrics, for så å videresende disse til Grafana for visualisering

Figur 12 viser hvordan Grafana har minimal tilgang til innhenting av data og metrics, men heller etterlater dette til Prometheus. Her vil Prometheus skrape etter åpne */metrics* endepunkter på samme måte som telegraf. På denne måten kan man monitorer de meste av tilstanden til Clusteret, så lenge endepunktene er åpne [67].

2.8 Python-pakker

Python pakker er moduler eller pakker som utvider og forsterker funksjonaliteten til kodespråket. Pakkene er forhåndsskrevet kode som kan gjenbrukes i applikasjoner. Disse pakkene tilbyr en rekke verktøy og funksjoner som kan benyttes til løse ulike oppgaver fra maskinlæring, webutvikling, dataanalyse og systemadministrasjon. PyPI, Pyhton Package Index, er databasen for publiserte Python-pakker og inneholder over 500 tusen ulike verktøy, fordelt over 10 millioner filer. PyPI muliggjør enkel installasjon og implementering av pakker via tredjeparts programvare som Pip [68].

2.8.1 Flask

Flask er et micro-rammeverk for webutvikling i Python. Flask er designet for å være lettvekts, fleksibelt og skalerbart i form av både enkle websider, og større prosjekter. På grunn av sin enkelhet og minimalistisk design har Flask blitt et populært valg for utvikling av webapplikasjoner og APIer. Flask tilbyr kjernen i websideutvikling, og selv om micro-rammeverket ikke tilbyr alt av funksjonalitet, kan dette tilbys av andre pakker, noe som gjør rammeverket enkelt for utviklere å tilpasse etter sitt behov [69].

2.8.2 Pandas

Pandas, også kjent som Panel Data, er et Python-bibliotek designet for effektiv datamanipulasjon og analyse. Biblioteket bruker avanserte datastrukturer for å støtte innlasting, forberedelse, manipulasjon, modellering og analyse av data i diverse filformater.

Pandas ble initialt utviklet for finansmarkedet, da funksjonalitet i hovedsak omhandlet tidsseriedata og prosessering av aksjepriser og informasjon. For å kunne håndtere dette, har Pandas utviklet en rekke funksjoner som går utover finansmarkeder, slik at det også kan anvendes i mer generelle sammenhenger. På bakgrunn av denne rike funksjonaliteten kan pakken benyttes til en rekke områder, som å transformere en fil fra en filtype til en annen [70].

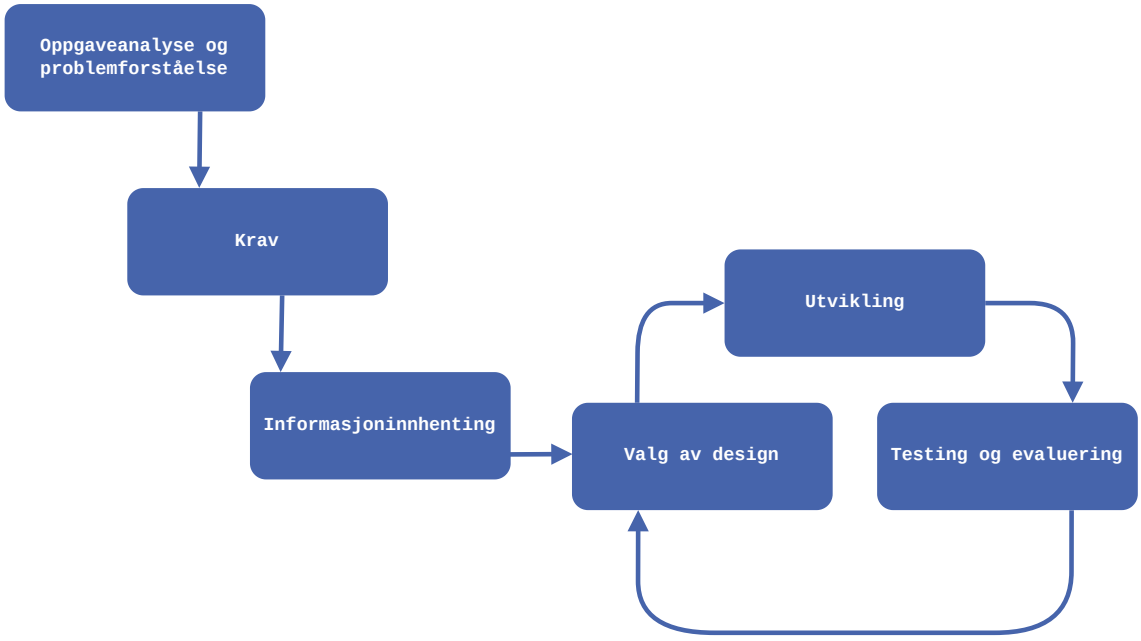
3 Metode

3.1 Metodevalg

I denne oppgaven baserer vi store deler av hoveddesignet på metoder innenfor teknologiforskning. Solheim og Stølen definerer teknologiforskning som ”forskning for å frambringe nye og forbedrede artefakter” [71]. Med artefakter menes menneskeskapte objekter. I denne oppgaven skal vi lage en PoC løsning der vi vil benytte teknlogiforskningsmetoder for å besvare spørsmålet om ”Hvordan frambringe et forbedret artefakt?”. Teknologiforskning har sin tyngde innenfor anvendt forskning, som handler om forskning for å finne løsninger på praktiske problemer [71]. I vårt metodevalg har vi tatt utgangspunkt i stegene i en teknologiforskningsmetode, som er en iterativ prosess. Solheim og Stølen beskriver hovedstegene i denne prosessen [71]:

1. *Problemanalyse* - Forskeren kartlegger et potensielt behov for et nytt eller forbedret artefakt ved å interagere med mulige brukere og andre interessenter.
2. *Innovasjon* - Forskeren forsøker å konstruere et artefakt som tilfredsstiller det potensielle behovet. Den overordnede hypotesen er at artefaktet tilfredsstiller behovet.
3. *Etterprøving* - Forskeren tar utgangspunkt i det potensielle behovet og formulerer prediksjoner som artefaktet. Så undersøker forskeren om prediksjonene slår til. Dersom resultatene er positive, kan forskeren argumentere for at artefaktet tilfredsstiller behovet.

Stegene i denne prosessen har som mål enten å styrke eller svekke hypotesene som tilhører artefaktet. Hypotesene til denne oppgaven samsvarer med kravene som er presentert i resultatkapitlet. Vår overordnede hypotese er: *Løsningen tilfredsstiller behovet*. Negative testresultater som ikke tilfredsstiller kravene/hypotesene stimulerer til nye iterasjoner i hovedstegene; problemanalyse - innovasjon - etterprøving [71]. Vi har valgt en metode som tar utgangspunkt i denne iterative utviklingsprosessen. Vi har delt hoveddesignet inn i seks faser. Metoden er illustrert i figur 13.



Figur 13: Metodevalg

For å kunne besvare forskningspørsmålene best mulig var det nødvendig med omfattende informasjonsinnhenting både fra litteratur og intervjuer. I vår tilnærming ønsket vi å danne det beste grunnlaget for valg av design, både for å bedre empirien til konklusjonen, men også for å kunne sikre at konseptet som utvikles oppfyller oppdragsgivers behov.

3.2 Fase 1 - Oppgaveanalyse og problemforståelse

I denne fasen skulle vi forstå hva slags problem som ble presentert for oss i oppgaveteksten. Fasen gikk ut på å bryte ned oppgaveteksten og analysere den. Her jobbet vi tett med veilederen og hadde flere møter for å sette oss inn i den praktiske konteksten av oppgaven. Intervju ble brukt for å få erfaring fra fagpersoner med lang erfaring innen IT til å hjelpe oss med å forstå hvilke metoder og verktøy som tradisjonelt blir brukt for å få oversikt over infrastruktur. Vi ble gitt et datasett fra veilederen, som vi skulle sette oss inn i, og forstå behovet brukere har for å skaffe oversikt. Målet med denne fasen var å identifisere hva slags artefakt som skulle utvikles, og hvorfor. Det ble utviklet problemformulering og tilhørende forskningspørsmål.

3.3 Fase 2 - Krav

Fasen har som mål å besvare forskningsspørsmål 1 ved å sette krav til løsningen. Bakgrunn for kravene var å ha rettede mål å jobbe mot, og på den måten skape progresjon og systematisk utvikling. Kravene ble aktivt brukt gjennom den iterative utviklingsprosessen. Hvert krav ble nøye utforsket og utviklet for å møte oppdragsgivers spesifikke

behov, samt basert på innsikt fra intervju. Deretter ble de testet og evaluert for å sikre at de oppfylte de ønskede standardene. På grunnlag av de fastsatte kravene ble innsamlingen av informasjon utført på en mer systematisk måte, ettersom disse kravene spesifiserte hvilke komponenter løsningen skulle omfatte. Kravanalysen startet med å utforme prosjektbehov ut ifra oppdragsgivers behov. I dialog med veileder kunne krav endres eller fjernes basert på gjennomførbarhet underveis.

Arbeidet med kravene bestod hovedsakelig i å identifisere og analysere disse. I samråd med oppdragsgiver kunne vi til slutt fastslå de endelige kravene til løsningen.

3.4 Fase 3 - Informasjonsinnhenting

Fase 3 gikk ut på å utforske hvilke eksisterende open-source programmer som kan modifiseres og brukes for løsningen. Det ble gjennomført tradisjonelt litteratursøk for å undersøke relevante programmer. Tradisjonelle litteratursøk følger narrativet til forfatteren, og med det baserer seg på helhetsvurderinger istedenfor grundig og omfattende studier. Vi får dermed et bredere fokus og mulighet til å sammenlikne og drøfte litteraturen basert på et målbevisst utvalg [72][73]. I tillegg til å utforske mulige programmer, ble det i denne fasen utforsket muligheter rundt hvordan vi skal sikre en allsidig og brukervennlig løsning. Målet med denne fasen var å gjøre kompetanseheving for å svare på forskningsspørsmål 2 og 3. Det var i denne fasen hensiktsmessig å samle informasjon og perspektiver fra fagspesialister. Dette ble gjort ved å snakke med spesialister ved avdeling, intervju og samtaler med professor ved skolen. Intervju skulle også belyse utfordringer og muligheter med ulike løsninger. Veiledning fra professoren ble gjort over e-post, mens samtaler med avdeling og intervju ble gjort fysisk og over web. Dette gjorde at vi kunne få presisjon og kvalitet i svarene slik at de ble treffende for får oppgave. Tilsammen gav litteraturstudien og fagsamtalene grunnlaget for å starte med valg av design i fase 4.

3.5 Fase 4 - Valg av Design

Designvalg var første steg i den iterative prosessen der vi skal utforme et PoC. På bakgrunn av de gjennomførte intervjuene og kompetansehevingen gjennomført i fase 3, var målet i denne fasen å planlegge en løsning som oppfyller kravene. Valg av hvilke program som skulle benyttes var et essensielt steg for å sikre gjennomførbarheten til prosjektet. Denne delen av oppgaven besvarer forskningspørsmål 3, om hvilke open-source verktøy som egner seg. Første steg var å utforske hvilke mulige løsninger vi stod ovenfor. Vi satt av en tidsramme på en uke med testing og evaluering for deretter å velge hvilken løsning som skulle benyttes.

Grunnet oppgavens natur, og ønske om å utforske ulike programmer som kan benyttes, var det ønskelig å ha en egen fase for valg av design. Ved å sammenligne kravene med informasjonen fra informasjonsinnhentingen kunne vi sette et godt utgangspunkt for valg

av design og starte designutviklingen. Etter valg av programvare, gikk arbeidet ut på å forstå ulike komponenter i programvaren, utforske muligheter og planlegge testmiljøet.

Oppsett av testmiljø var bidragsytende for å danne en grunnleggende forståelse, og minske sjansen for feil og unødvendig tidsbruk under utvikling. Ved å praktisk utforske mulighetene og funksjonalitetene innenfor programvaren kunne vi sikre god progresjon i overgangen til neste fase av prosjektet.

Ettersom løsningen skal være designet for å motta data fra mange kilder, og dermed vil være meget generell, ble det også i denne fasen definert krav til hvordan løsningen skulle brukes for et felles standpunkt for alle kildene. Det var viktig å definere hvordan innkommende data skulle se ut, samt hvilke data som tillates. Under denne prosessen ble avgrensinger gjennomført og kravene evaluert på nytt for å sikre gjennomførbarheten til oppgaven basert på de gitte tidsrammene.

3.6 Fase 5 - Utvikling

Utvikling er andre fasen i den iterative utviklingsprosessen og baserer seg på å realisere designet fra fase 4, gitt kravene og informasjonen fra fase 2 og 3. For best mulig resultat var det viktig å sikre et godt design i tråd oppdragsgiver ønsker, før begynnelsen av dette steget. Utviklingen av PoC startet etter at designet var besluttet og utviklingsmiljøet var klargjort.

I den iterative modellen skjer testing underveis i utviklingsprosessen. Basert på dette, ble det gjennomført en rekke mindre tester for å sikre at konseptet treffer rett design. Fasen startet med å sikre funksjonalitetskrav for løsningen å sjekke disse opp mot det valgte programvareverktøyet. Ved å sørge for den grunnleggende funksjonaliteten kunne vi sikre at konseptet var mulig å fremstille, og dermed gå videre til logikken i løsningen.

Kravene gitt i fase 2 og designets premisser satt i fase 4, ga klare retningslinjer og prioriteringer for hvordan utviklingsprosessen skulle gå for seg. I hovedsak baserte utviklingsprosessen seg på å utvikle konseptet til å støtte ett av kravene, for deretter teste og analysere resultatet. Ved å benytte denne metoden sikret vi at hver komponent i løsningen fungerte, før vi gikk videre til neste krav.

Samtidig som utviklingsarbeidet foregikk, ble det holdt regelmessige møter og samtaler med veileder for å diskutere mulige tillegg eller ny funksjonalitet som kunne være fordelaktig å integrere i løsningen. Formålet med disse møtene var å fremme framdrift og justere løsningens omfang underveis. Et konkret eksempel på dette er implementasjon av Kubernetes-overvåking, da dette var noe veileder ønsket i løsningen.

3.7 Fase 6 - Testing og evaluering

Testing og evaluering skjedde kontinuerlig under utviklingen av PoC. Både under fase 3, 4 og 5 ble det testet og evaluert hvordan programvaren og APIet fungerer, og hvilke kapabiliteter de har. Dette var helt nødvendig for å få en forståelse for hvordan løsningen skulle bli utformet. I den iterative modellen testes konseptet underveis i utviklingsprosessen. Dette medførte en rekke mindre tester for å sikre at konseptet ikke avviker fra opprinnelig design. Fasen startet med å sikre funksjonalitetskrav for løsningen og sjekke disse opp mot det valgte programvareverktøyet. Ved å først sørge for den grunnleggende funksjonaliteten, kunne vi sikre at konseptet var mulig å fremstille, og dermed gå videre til logikken i løsningen.

[Denne siden er blank med hensikt]

4 Resultater

4.1 Krav til løsningen

I samråd med oppdragsgiver ble følgende krav til løsningen utformet:

- Løsningen skal kunne visuelt presentere statisk data fra vilkårlige kilder
- Løsningen skal støtte multi-tenancy, å skille mellom brukere, og tilegne enheter til sin respektive tenants
- Løsningen skal være enkel å implementere
- Løsningen skal tilby overvåking av Kubernetes Clustere
- Løsningen skal støtte JSON og YAML filer

Løsningen skal ikke basere seg utelukkende på overvåking, men heller på muligheter rundt presentasjon av vilkårlige datafiler med skille mellom tenants. Denne løsningen skal tilby overvåkning og datavisning for flere brukergrupper eller kunder i en bedrift. Løsningen skal kunne motta rådata og strukturere og visualisere dette i et dashboard. Løsningen skal som et minimum kunne støtte JSON og YAML filer. Løsningen skal automatisere alt som er mulig å automatisere for at det skal være enkelt å implementere for en virksomhet. Løsningen skal også kunne tilby overvåkning av Kubernetes Clustere.

4.2 Intervju

I dette kapitlet legger vi fram resultater fra et kvalitativt intervju, samtaler med en professor ved Cyberingeiørskolen og samtaler med veileder. Med intervju og samtaler ønsket vi å finne ut hvilke mulige programvare-løsninger vi kan bruke for å løse problemet. Vi ønsket også å finne ut om hvordan virksomheter i dag får en tilfredsstillende oversikt over driftsmiljøets tilstand og konfigurasjon. Det var ønskelig å få perspektiver rundt en multi-tenant løsning, samt hvordan fagpersoner ser for seg en løsning på problemstillingen.

4.2.1 Mulige løsninger

Fra veiledningssamtaler og intervju fikk vi følgende forslag:

1. **Zabbix**

   Zabbix brukes vanligvis til overvåkning, men det burde være mulig å bruke APIet til å vise informasjon fra JSON-filer. Man kan gjøre dette ved å ha et Python-script som leser inn alle JSON-filene og grupperer basert på ressurs og tenant-informasjon. Når dataene er prosessert kan de sendes til Zabbix gjennom API slik

at brukerne kan se det. Zabbix har et dashboard for presentasjon av data. Tenant-informasjonen kan brukes til å gi forskjellige grupper tilgang til de forskjellige ressursene.

2. **Grafana**

   Å bruke Grafana for å visualisere data er også en mulighet. Man kan bruke Prometheus for å hente data. Så et Python script til å transformere dataene, altså gruppere og gjøre klar for presentasjon, og så presentere det i Grafana. Velger man Grafana så må man finne ut av multi-tenancy i Grafana og sørge for at dataene transformeres slik at de kan importeres og vises riktig.

3. **Lage et verktøy i React**

   I likhet med den mulige løsningen med Grafana, kan man transformere dataene med et Python script for deretter å sende til en egenlaget løsning i React. En egenlaget løsning vil gi stor handlefrihet.

4. **InfluxDB**

   InfluxDB er primært en tids-seriedatabase, men den skal kunne håndtere statisk informasjon også. Den har også et eget dashboard. Datasettet som vi har må kunne sendes inn i InfluxDB og kunne presenteres til rett tenant.

4.2.2 Multi-tenancy perspektiver

I intervjuet belyste respondenten noen perspektiver knyttet til multi-tenancy. Respondenten tar for seg perspektiver i et leverandørs perspektiv for IT-bedrifter. Respondentens erfaring innen multi-tenancy er hovedsakelig for ticket system. Ticket system brukes av leverandører for å håndtere brukerhenvendelser, problemer og oppgaver med deres leveranser. Respondenten forteller at multi-tenancy egner seg godt for å samle oversikt og oppnå god ressursdeling. Med ticket system, som OTRS (originally Open-Source Ticket Request System), får man en samlet plattform for å eksempelvis sette rettigheter for andre applikasjoner, som forenkler denne prosessen. Utfordringene med multi-tenancy ligger i at utformingen av løsningen krever grundig planlegging om hvordan den skal fungere, samt kontroll på rettigheter og personvern. Multi-tenancy vil komplisere strukturen til løsningen, men til gjengjeld samler man plattformen [74].

4.2.3 Hvordan ville en virksomhet løst problemet i dag

Dette kapitlet tar for seg intervju-svar på spørsmålet;

Hvilke verktøy og løsninger bruker virksomheter for å holde oversikt over driftsmiljøet i dag?

Det kommer fram at det ikke finnes en standardisert løsning og at det er stor variasjon i tjenester fra leverandør til leverandør. Microsoft sine tjenester dekker oftest hele spekteret

av hva som trengs for IT-drift, men kan bli veldig kostbart. Respondenten peker på at disse løsningene kan koste opp mot 70 000 kroner i måneden [74].

HaloPSA er et eksempel på en løsning man ser hos bedrifter. HaloPSA er et britisk webbasert helhetlig system for integrasjon av klienthåndtering med tilhørende overvåkning. Tjenesten støtter multi-tenancy.

For detaljert oversikt over nettverket nevner respondenten Netbox som en god løsning. Netbox er open-source konfigurasjonssystem som brukes mot nettverksløsninger. Løsningen egner seg for oversikt over konfigurasjon, hvilke ressurser bedriften har og hvor ressursene er. Løsningen brukes gjerne for nettverkstopologi. Respondenten mener Netbox egner seg godt for å vise nettverksrelasjoner, men kan også brukes til VMer og deres relasjoner. Løsningen har mulighet for automatisering. Respondenten har gode erfaringer med Netbox og forteller at det brukes mye av virksomheter i dag. En annen tjeneste respondenten peker på som blir brukt mye til overvåking og oversikt, er Zabbix. Respondenten forteller at Helse-Øst eksempelvis benytter Zabbix for å overvåke alarmsentraler, men ser også utbredt bruk i resten av Helse-Norge. Zabbix er open-source og et veldig godt verktøy for å håndtere feil, samt gi gode visualiseringer [74]. Respondenten ser på Zabbix som et av de beste verktøyene for overvåkning og kontroll over driftsmiljø. Zabbix viser en god historikk, har god støtte for multi-tenancy, og er fleksibelt.

Andre verktøy som blir nevnt er N-able og Solarwinds, som ikke er open-source.

4.3 Data fra mange kilder

Med en datainnsamling av statiske data slik som presentert i illustrasjonen 2 i kapittel 2.2.1, viser det seg at den begrensende faktoren for håndtering av data fra mange kilder er hvor mange filformater løsningen kan støtte. Implementasjon for støtte av flere filformater, og dermed flere datakilder, ble utforsket i form av en litteraturstudie. Vi resonerte oss fram til at det ville være lønnsomt å transformere all data til samme format for å danne et felles standpunkt. Vi fant et Python bibliotek som heter Pandas som muliggjorde enkel manipulasjon og transformasjon av ulike dataformater. Pandas tilbød muligheten til å omforme en rekke datafiler til ett format. Etter funn av et tilfredsstillende verktøy med litteraturstudie utforsket vi Pandas i en praktisk ramme, der testing av ulike funksjoner viste oss hvordan de kan anvendes i løsningen. Vi valgte JSON som et felles standpunkt for alle filformater. Alle filer som sendes til løsningen blir transformert til JSON med bruk av Pandas før de blir prosessert og sendt til Zabbix for presentasjon. Resultatet fra implementasjonen av Pandas vises i tabell 2, i kapittel 5.7.

4.4 Designvalg

Basert på de definerte kravene, samt gjennomførte intervju og samtaler med veileder, sto vi ovenfor et kritisk valg angående hvilke open-source verktøy som var mest hensiktsmessige

å benytte til å utvikle et PoC for oppgaven. I de innledende fasene av prosjektet ble det diskutert en rekke programmer som alle hadde egenskaper som kunne bidratt positivt til løsningen. Ved å definere hvilke funksjonaliteter det var ønskelig at applikasjonen skulle ha, kunne vi sammenligne de ulike applikasjonene.

- **Open-source:** Applikasjonen skal ha åpen kildekode, da dette er gratis og gir gode muligheter for egendefinerte scripts og utvidet funksjonalitet.
- **Fremvisning av statisk data:** Applikasjonen skal kunne motta og fremvise statiske data.
- **Mulighet for brukerhåndtering:** Applikasjonen skal kunne skille mellom brukere, brukergrupper og tilegne ressurser/data til eieren/tenanten.
- **API:** Ved å benytte en eksisterende applikasjon skal det være mulig å kommunisere med applikasjonen igjennom et API. Dette er for å ha muligheten til å utvikle et eget program som automatiserer prosesser.
- **Kubernetes-overvåking:** Applikasjonen skal støtte overvåking av Kubernetes Clustere.

Vi utforsket de mulige løsningene vi kom fram til med litteratursøk og testing i testmiljø.

4.4.1 Zabbix

Med litteratursøk på Zabbix fant vi ut at Zabbix hadde en veldig god og intuitiv dokumentasjon. Det viste seg også at Zabbix tilbyr en fleksibel løsning som er relativt allsidig. Vi testet programvaren ved å laste den i Docker. Når vi først hadde fått satt opp applikasjonen var den enkel å manøvrere seg i. Med litteratursøk fant vi raskt at det var gjennomførbart å bruke applikasjonen i et multi-tenant miljø [75]. Utfordringene som følger med denne løsningen er hvordan oppretter man tenants, om det kan gjøres automatisk og hvordan dataene må se ut for å kunne gi til rett tenant.

4.4.2 Grafana

Litteratursøk viste at Grafana er et meget kraftig verktøy og et av de ledende verktøyene innen visualisering av data. Det er også meget fleksibelt og har mange tilpasningsmuligheter [76]. Det viste seg også at Grafana hadde god støtte for multi-tenancy ved bruk av Grafana Loki, et loggsystem del av Grafana porteføljen [41]. Vi testet programvaren ved å laste den i Docker. Vi erfarte at det ikke var veldig intuitivt å manøvrere seg i Grafana. Programvaren er forsøkt gjort veldig allsidig, som for oss opplevdes tungvint. Det finnes mye god veiledning på hvordan man kan starte å lære seg Grafana. Etter veiledning fra ulike uformelle Youtube kilder fikk vi forståelse for hvordan man setter opp dashboard i Grafana. Utfordringen med Grafana er å finne ut av hvordan tenants kan opprettes og hvordan data kan assosieres med de forskjellige tenantsene.

4.4.3 Lage et verktøy i React

I dialog med veileder ble det foreslått å benytte React, et JavaScript-bibliotek, til å utforme en egen applikasjon for visualisering av dataene. React benyttes til å utvikle avanserte single-page applikasjoner [46]. Ved å lage applikasjonen selv vil vi ha full kontroll over funksjonalitet og logikken til applikasjonen, samt kunne skreddersy den til oppgaven. React følger en komponentbasert struktur, noe som muliggjør gjenbruk av selvbygde UI-komponenter. Ved litteratursøk på mulit-tenancy for React fant vi at det finnes tilgjengelige plugins og pakker som støtter brukerhåndtering for en ønsket løsning. Selv om disse verktøyene legger til rette for implementeringen av multi-tenancy ved å håndtere ulike brukerkontoer og separasjon, er det fortsatt utviklerens ansvar å sikre applikasjonens sikkerhet.

4.4.4 InfluxDB

InfluxDB tilbyr hovedsakelig en tidsseriedatabase optimalisert for rask lagring og analyse av tidsstemplet data. Ved testing erfarte vi at verktøyet er spesialisert til databaser som er mindre interessant for oppgaven, allikevel er den fleksibel slik at den tillater tilpasning til vårt behov. Den har et relativt kraftig API som lar oss automatisere mange prosesser. Som for Grafana, blir utfordringen å sette opp tenants og transformere dataene slik at de blir assosiert med riktig tenant. InfluxDB brukes gjerne i sammenheng med Grafana, der lagringen av data ligger i InfluxDB. Med bakgrunn i ønske fra brukere har InfluxData lansert multi-tenancy Grafana støtte ved bruk av InfluxDB Cloud. Dette tillater brukere å lagre data i skyen og skille mellom tenants, slik at man tilbyr forskjellige dashboard til brukerne. Denne funksjonaliteten er desverre ikke open-source og er relativ kostbar [77]. Det er allikevel multi-tenancy støtte i InfluxDB uten bruk av Grafana, slik at InfluxDB er en mulig løsning.

4.4.5 Valg av design

For valg av kodespråk for å scripte prosesser valgte vi å benytte Python, et programmeringsspråk som begge i gruppen allerede var kjent med. Dette valget tillot oss å utnytte vår eksisterende kunnskap og erfaring, noe som bidro til en mer effektiv utviklingsprosess. For valg av verktøy å basere løsningen rundt, benyttet vi sammenligningsfaktorene presentert tidligere i kapitlet. Resultatet vises i tabell 1.

Tabell 1: Sammenligning av ulike open-source verktøy

	Zabbix	Grafana	Influx-DB
Open-Source			
Fremvisning av statisk data			
Mulighet for brukerhåndtering			
API			
Kubernetes-overvåking			

Tabell 1 beskriver funksjonaliteten til de ulike verktøyene. Grønn viser til ”out-of-the-box” funksjonalitet, mens gul viser til at funksjonalitet er mulig å oppnå ved tilleggspakker eller plugins. Som det fremgår i tabell 1, er det tydelig at ikke alle verktøyene inneholder den etterspurte funksjonaliteten direkte ”out-of-the-box”. Å utvikle et eget verktøy i rammeverket React var også en mulighet som gir oss stor handlefrihet, med dette ville vært tidkrevende for oss. Ettersom ingen av oss hadde noen omfattende erfaring med React fra før, konkluderte vi med at det ikke ville være hensiktsmessig å lage en løsning fra bunnen, men heller bruke eksisterende verktøy. Med beslutningen om å se bort fra en egen laget løsning i React, så vi videre på å utvikle et program som kunne automatisere prosesser og utvide funksjonaliteten til enten Zabbix, Grafana eller InfluxDB.

Grafana og InfluxDB krever installasjon av ekstra tillegg for å oppnå det som er ønskelig for oppgaven, noe som innebærer ytterligere kompleksitet. Zabbix er utstyrt med den nødvendige funksjonaliteten fra start, uten behov for ytterligere tilpasninger eller tillegg. Zabbix Trapper Item muliggjør fremvisning av statisk data på en enkel måte, og Kubernetes-overvåking er integrert i tjenesten.

Som visualiseringsverktøy er Grafana en svært allsidig applikasjon. Dens funksjonalitet begrenses derimot av behovet for tredjepartsapplikasjoner og tilleggstjenester for å fungere slik vi ønsker. Zabbix og Grafana er i hovedsak rettet mot overvåking i sanntid, og har noe begrenset funksjonalitet for slik vi ønsker å presentere statiske data.

InfluxDB har som hovedfunksjon å lagre tidsseriedata. Da denne funksjonaliteten kan være nyttig, vil ikke funksjonaliteten være like nyttig for løsning av oppgaven. På samme premisser som Grafana, har InfluxDB et behov for tilleggstjenester for å fungere slik vi ønsker, og oppnår derfor ikke kriteriene i denne sammenligningen.

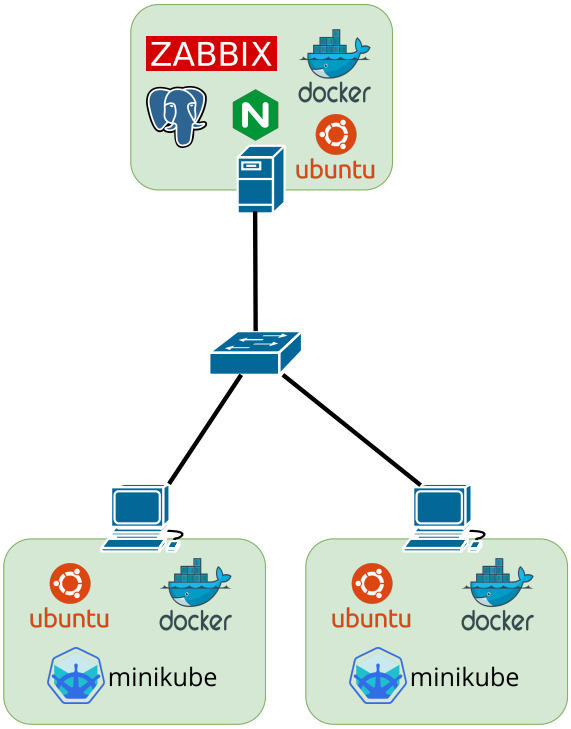
Etter en grundig evaluering av de ulike verktøyene, ble det tydelig at Zabbix var det mest passende valget for vår oppgave. Dette valget begrunnes med Zabbix sin allsidighet og funksjonalitet.

Den brede funksjonaliteten til Zabbix var den avgjørende faktoren for oss. Med dette valget om å designe løsningen rundt Zabbix, gjenstår utfordringene om å finne ut hvordan man skal skille mellom de ulike tenantene innad i Zabbix, samt tilegne riktige data til riktige brukere.

5 Design og utvikling

5.1 Testmiljøet

For testmiljøet satt vi opp et nettverk bestående av to fysiske klienter og en server som beskrevet i figur 14.



Figur 14: Testmiljøet og programvare bestående av to klienter og en server

Enhetene er koblet sammen med en svitsj som vist i figur 14. På hver av klientene ble det kjørt Minikube i Docker for å gi Kubernetes Cluster overvåkning. På serveren installerte vi Zabbix Server med PostgreSQL i Docker. Vi installerte Zabbix Frontend med NginX, i nettverk med Zabbix Serveren i Docker. Hvilke host systemer du kjører Zabbix- server og frontend med er forholdsvis valgfritt og påvirker ikke oppgaven. For Kubernetes-overvåkning måtte vi sette opp port forwarding på serveren for å nå tjenestene, men dette er beskrevet i detalj i kapittel 5.4 om Kubernetes. Vi benyttet oss av den Linux-baserte distribusjonen Ubuntu 22.04 på alle enhetene, ettersom vi mener Linux har best støtte for Kubernetes og generell programutvikling.

5.1.1 Oppsett av Zabbix

Første steg var å sette opp Zabbix og skaffe initiell kommunikasjon mellom Zabbix-server og vår applikasjon. Som nevnt krever Zabbix en rekke roller for å fungere, og ved å benytte conteinisering via Docker kan man enkelt sørge for at alle rollene lastes ned

riktig og fungerer sammen. Når Zabbix er lastet vil man kunne aksessere det webbaserte grensesnittet ved IP-adressen til serveren. APIet kan nås på $http://<server-ip>/api\_jsonrpc.php$. For å oppnå initiell kontakt mellom applikasjonen og Zabbix benyttes *requests*, som er pakke man kan laste ned fra PyPI. For å kunne kommunisere mellom applikasjon og Zabbix, krever Zabbix en autorisasjonsbærer på forhånd av alle operasjoner. Her finnes et eget API-kall som returnerer bæreren.

```
link = "http://192.168.0.93/api_jsonrpc.php"

data = {
        "jsonrpc": "2.0",
        "method": "user.login",
        "params": {
            "username": "Admin",
            "password": "zabbix"
        },
        "id": 1
        }

resp = requests.post(link, headers = {"content-type": "application/json
   -rpc"}, json=data)
```

Koden beskrevet over viser syntaxen på API-kallet. I dette kodeeksempelet benyttes standard brukernavn og passord for Zabbix. Responsen fra serveren er strukturert data, som kan omformes til eksempelvis JSON.

5.2 Datakilder

For testing og utvikling benyttet vi først og fremst statiske datasett vi fikk fra veileder. Disse filene inneholdt data hentet fra VMware Vsphere, Ansible og Kubectl. Disse statiske dataene var hensiktsmessig for å utvikle Python script, slik at vi kunne transformere dataene og sende til Zabbix. Alternativet var å benytte agenter for deler av datainnsamlingen, men vi konkluderte med at det ville komplisere feilsøkingen og utviklingen.

5.2.1 Hvilke datakilder og data

De konkrete filene vi benyttet var:

- Data fra VMware Vsphere som beskriver ulike VMene på en server.
- Data fra Kubectl, som snakker med Kubernetes API for å hente data om hvilke Kubernetes deployments den har, samt en fil om hvilke services som kjører.
- Data fra Ansible som beskriver hvordan de ulike VMene er konfigurert og deployert.

Vi benyttet også diverse test-data i ulike dataformater for å undersøke at løsningen støttet det gitte formatet og dens struktur.

5.2.2 Krav til dataene som mottas

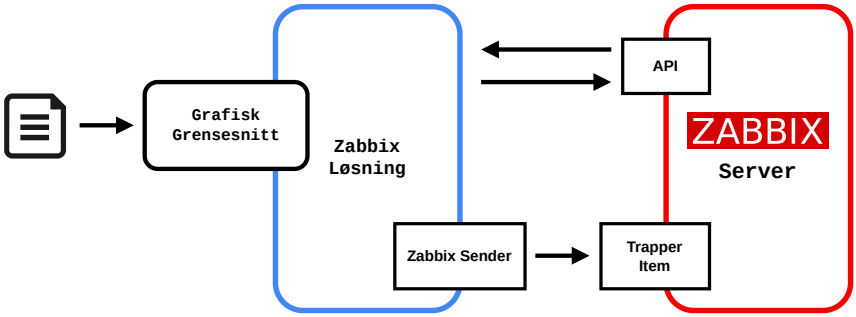
Etter litt testing fant vi ut at dersom koden vår skal støtte mange ulike formater må alle de ulike formatene ha et felles standpunkt. Det er altså nødvendig å sette krav til dataene vi sender til løsningen. Det er nødvendig at det beskrives hvilke tenants de ulike ressursene tilhører i filen. Dette er for at data kan assosieres med de forskjellige tenantsene. Det er nødvendig at det er beskrevet hvilken type data filen beskriver, for at koden skal kunne iterere gjennom og lese dataene inn til Zabbix API. Det er også nødvendig å beskrive hvilken kilde dataene mottas fra, for å oppnå riktig presentasjon i dashboardet. Følgende krav er satt til dataene som mottas:

- Dataene i filen må beskrive hvilken kilde (**source**) ressurser kommer fra. Dette kan eksempelvis være vCenter som har data om VMer, der vCenter er kilden.
- Dataene i filen må beskrive hvilken **type** data den inneholder. Dette kan eksempelvis være en liste med VMer der ”VM” er typen ressurs det skal itereres gjennom.
- Dataene i filen må beskrive hvilken **tenant** de ulike ressursene tilhører. Dette kan eksempelvis være at for hvert item i en liste med VMer er det tilført hvilken tenant itemet tilhører.

5.3 Implementasjon av generisk data til Zabbix

5.3.1 Generell ide

Vi har valgt å utvikle et webgrensesnitt for brukervennlighet og enkelt vise hvordan løsningen vil se ut. Figur 15 under viser overordnet hvordan løsningen fungerer.



Figur 15: Illustrasjon over kommunikasjonen mellom løsningen vår og Zabbix. Filer lastes opp til vår løsning, som deretter kommuniserer med Zabbix-server for å visuelt fremstille dataene.

Som figur 15 viser skal man kunne laste opp filene man ønsker til Zabbix via det grafiske grensesnittet. Videre vil vårt Python-program tolke dataene, og hente nødvendig informasjon fra Zabbix, for deretter å laste opp dataene til de gitte tenants, med bruk av Zabbix Trapper Item. Python programmet vil hente all nødvendig informasjon den trenger for at data som sendes blir assosiert med riktig tenant og kilde.

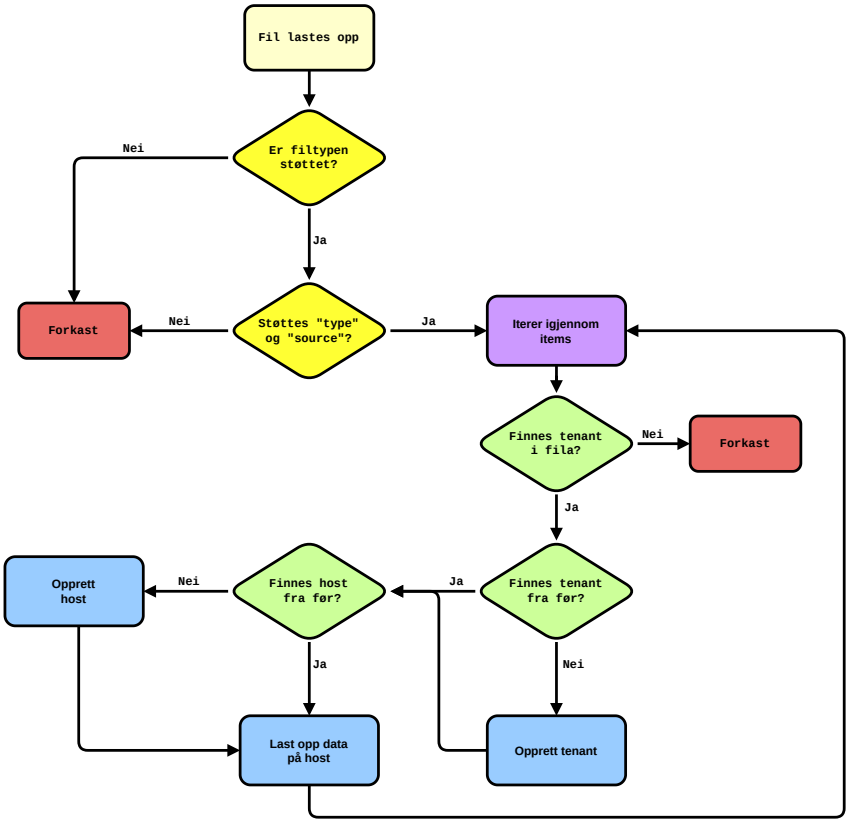
Som nevnt i kapittel 2.3.3 er brukerhåndteringen og tilgangsstyring i Zabbix logisk inndelt med grupper for både systemer (Hosts) og brukere (Users). Et viktig aspekt med designet var å velge hva som skulle defineres som tenant i vår løsning. Valget endte på å definere tenants som Host-groups. Dette valget tillot oss å enkelt inkludere flere kilder til samme tenant, samt opprette egne User-groups som er tilknyttet hver tenant (Host-group). Dette gjør det enkelt å tilegne brukere tilganger. Denne User-groupen gis altså automatisk tilgang til tenanten og dermed kan man ved å gi Users tilgang til User-groupen, for å gi brukere tilgang til all informasjon tilknyttet tenanten.

For administratorer som ønsker å laste opp data for presentasjon kan vårt grafiske grensesnitt benyttes. Ved å laste opp filen skal alt av API-kall og requests gjennomføres automatisk, slik at brukeren kan logge inn i Zabbix, og ha tilgang til sin data.

5.3.2 Logikk i koden

For at dataene som lastes opp skal assosieres med riktig bruker på riktig Host, er det nødvendig at logikken i koden er nøye planlagt, slik at riktige hendelser og endringer blir håndtert riktig.

Figur 16 viser hvordan den overordnede logikken i koden fungerer. Innledende i koden blir det gjort flere sjekker for at filen som lastes opp faktisk støttes.



Figur 16: Flytskjema over hvordan generisk data behandles

For brukeren skal det ikke være nødvendig med manuell konfigurering i Zabbix frontend. Ved opplasting av data skal applikasjon sette opp nødvendige tilganger og presentasjon i dashboard tilegnet tenant. Figur 16 viser at dersom tenant eller Host ikke finnes, vil dette opprettes før informasjonen lastes opp. Logisk vil dette være et flytskjema i seg selv, og noe vi vil se nærmere på i kapittel 5.3.3. Dersom filen som lastes opp inneholder flere dataobjekter/Hosts, vil applikasjonen iterere igjennom disse en etter en og utføre alle nødvendige endringer på hver av de.

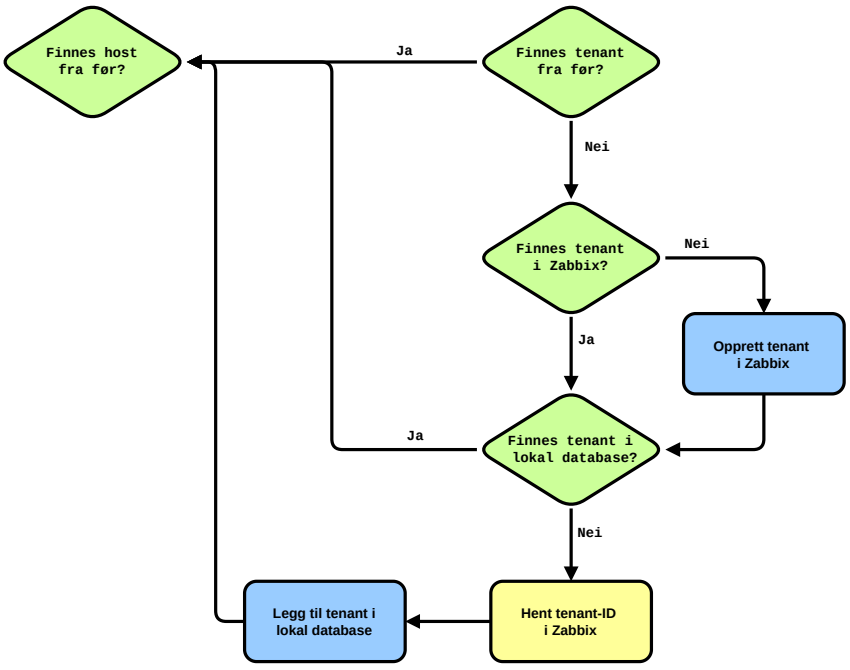
5.3.3 Multi-tenancy i Zabbix

For å opprette tenant i Zabbix benyttet vi Zabbix Host-group. For at dataene skal bli gitt til rett tenant må de følge de gitte kravene, gitt i kapittel 5.2.2. Koden vår lager automatisk tilsvarende User-group for hver Host-group. Dette gjør at den eneste manuelle prosessen i Zabbix Frontend er å operette brukere som man tilegner en brukergruppe for den aktuelle tenanten den tilhører. For å holde god oversikt over tenants i systemet var det hensiktsmessig med en egen lokal database. Databasen minimerer antall API kall til Zabbix, da oversikten benyttes for å utføre logiske operasjoner. Denne databasen ble en enkel JSON fil, med nødvendig informasjon om tenants.

```
1 {
2     "tenants": [
```

```
        {
            "name": "tenant_2",
            "id": "44",
            "description": "Kontor for intops"
        },
        {
            "name": "IKT",
            "id": "5",
            "description": "Kontor for IKT"
        },
        {
            "name": "tenant_1",
            "id": "67",
            "description": "Personellavdeling"
        }
    ]
}
```

For å sjekke at tenant finnes fra før blir det derfor gjennomført sjekk i både lokal tenant-database og i Zabbix. Dersom det ikke korresponderer vil dette fikses og IDer blir oppdatert. IDen til tenanten baseres på den interne IDen til Host-groupen i Zabbix.



Figur 17: Detaljert flytskjema over multi-tenancy aspekter ved koden. Utvidelse av figur 16

Figur 17 viser mer av logikken om hvordan applikasjonen vår holder kontroll over tenants, både ved sjekker i Zabbix og sjekker i lokal database. Figuren er en utvidelse av steget "opprett tenant" i figur 16

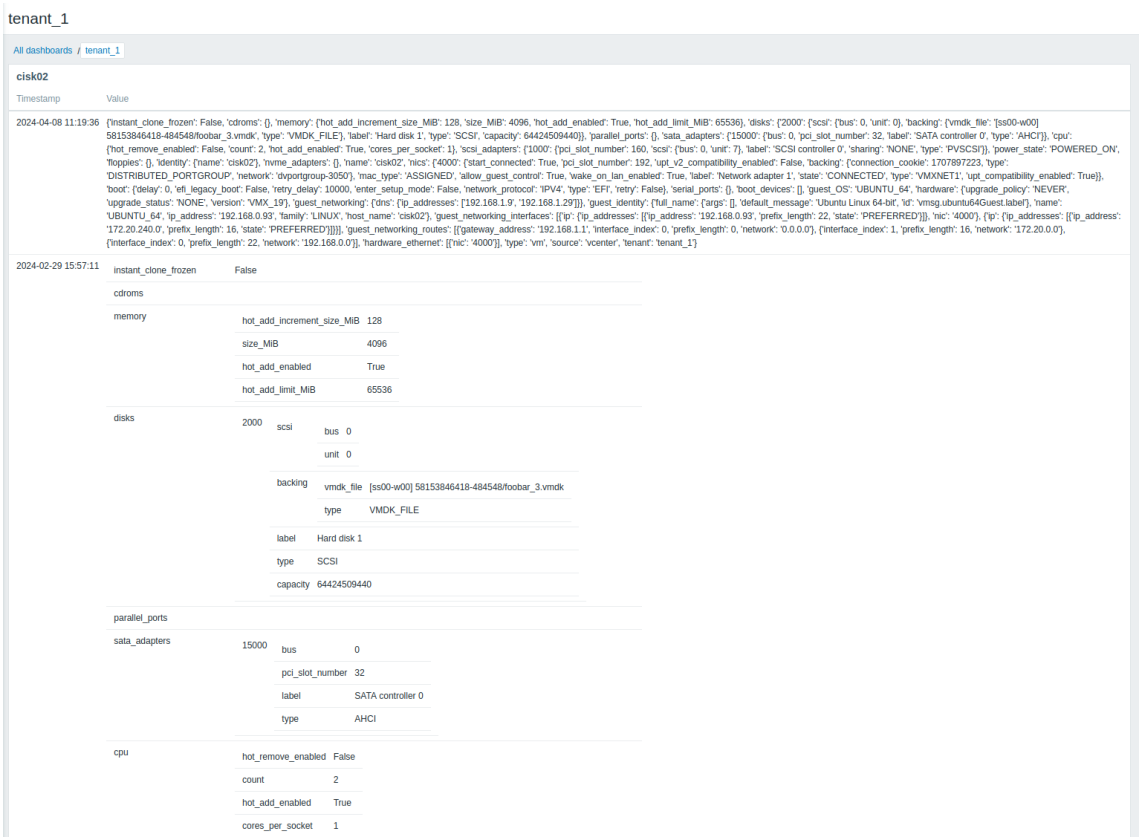
```
def hvilke_hostgroup_finnes():
    data = {
```

```
3         "jsonrpc": "2.0",
4         "method": "hostgroup.get",
5         "params": {
6             "output": "extend"
7         },
8         "id": 1
9     }
10    auth_tok = f"Bearer {auth()}"
11    resp = requests.get(link, headers = {"content-type" : "application/
      json-rpc", "Authorization": auth_tok}, json = data)
12    navn_svar = resp.json()
13    host_groups = []
14    for x in navn_svar["result"]:
15        host_groups.append(x["name"])
16    return(host_groups)
```

Koden over er API-kallet som ble benyttet for å sjekke hvilke tenants som finnes i Zabbix, her benyttes metoden *hostgroup.get*, med parmeteret ”output : extend” for å hente all informasjon om alle Host-groups. Ved å deretter iterere igjennom Host-groups og lagre navnene i en egen liste, har vi hentet navnet på alle tenants i Zabbix. Dette brukes blant annet for å sammenligne lokal database mot Zabbix sin.

5.3.4 Presentasjon av data

Zabbix tilbyr mange måter å presentere data på. I denne oppgaven ønsker vi en generisk løsning som presenterte alt som Zabbix mottok. Vi opprettet et dashboard for hver tenant, med tilganger kun til den gitte tenanten. Hvert dashboard må ha widgets for å gi noen informasjon. Vi ønsket å benytte en widget som var mest mulig allsidig. Til å begynne med benyttet vi vanlig *plain text* widget for å presentere item data fra de ulike Hostene, altså datakildene. Dette presenterer data på samme måte og i samme format som i den faktiske filen. Dette var en grei nok måte for å styrke en PoC. Allikevel utbedret vi koden etterhvert til å presentere disse JSON og YAML dataene i samme format. Vi valgte å benytte *plain text* widgeten sin HTML-funksjonalitet og omforme all data fra itemet til HTML. Dette ble gjort ved å bruke en Python pakke som gjorde om dataene til HTML, før det ble sendt til Zabbix. HTML er ment til å være menneskelig lesbart, og gav oss en ryddigere måte å presentere data til brukerne.



Figur 18: Presentasjon av data i JSON (øverste del), og HTML (nederste del)

Figur 18 viser samme data presentert i JSON i øverste del og HTML i nederste del av bildet. Presentasjonen i JSON er en kompakt mengde data som er lite oversiktlig og lesbart. Når vi valgte å gjøre om dataene til HTML, som vist i nederste del av bilde, fikk vi en mye mer oversiktlig og luftig presentasjon av data. Zabbix APIet støttet oppsett av dashboard, som tillot oss å automatisere prosessen. I koden sjekkes det om datakilden finnes fra før, og dersom den ikke finnes fra før lager vi en ny widget i tenant gruppen den tilhører. Dersom datakilden tilhører en ikke-eksisterende tenant blir den tilegnet en ny tenant og nytt dashboard. Dersom datakilden finnes fra før, oppdateres kilden i det gitte dashboardet for den gitte tenanten. Ved oppdatering av eksisterende kilder behandler Zabbix dataen og tilegner den et timestamp da dataen ble sendt til Zabbix. Dette gjør at brukeren kan se tidligere data fra samme kilde, noe som er hensiktsmessig for å oppdage tidsrom en endring skjedde i eksempelvis konfigurasjonen eller tilstanden. Det interessante med løsningen er at den er generisk for all type data som mottas i dataformatene presentert i kapittel 5.7. Det betyr at alt den mottar i disse formatene presenteres på tilsvarende struktur som nederste del av figur 18.

5.4 Kubernetes implementasjon

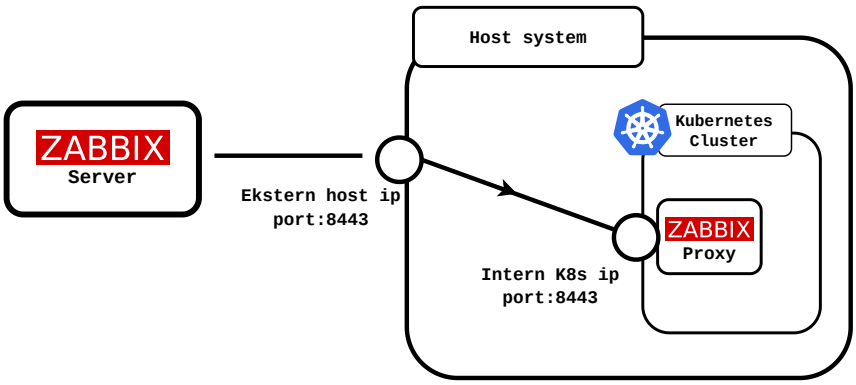
Kubernetes-overvåking ble implementert med bruk av Zabbix-Helm-Chart og Kubernetes templates. Løsningen krever noe manuell konfigurasjon da det er nødvendig å installere

Helm-Chartet på Clusteret man ønsker å overvåke. Videre skal vår løsning ta inn riktige parametere for å sørge for at overvåkingen settes opp riktig, disse parameterne innebærer tenant, navn, ip-adresse, proxy navn og en generert autentisering token. For at ikke det skal oppstå komplikasjoner må proxy navn være unikt for hvert Cluster. Som konsekvens må man før installasjon endre verdien for proxy navn i Helm-Chartet til noe unikt for hvert Cluster. Etter installasjon av Zabbix-Helm-Chart kan man hente token med kommandoen under.

```
kubectl get secret zabbix-service-account -n monitoring -o jsonpath={.data.token} | base64 -d
```

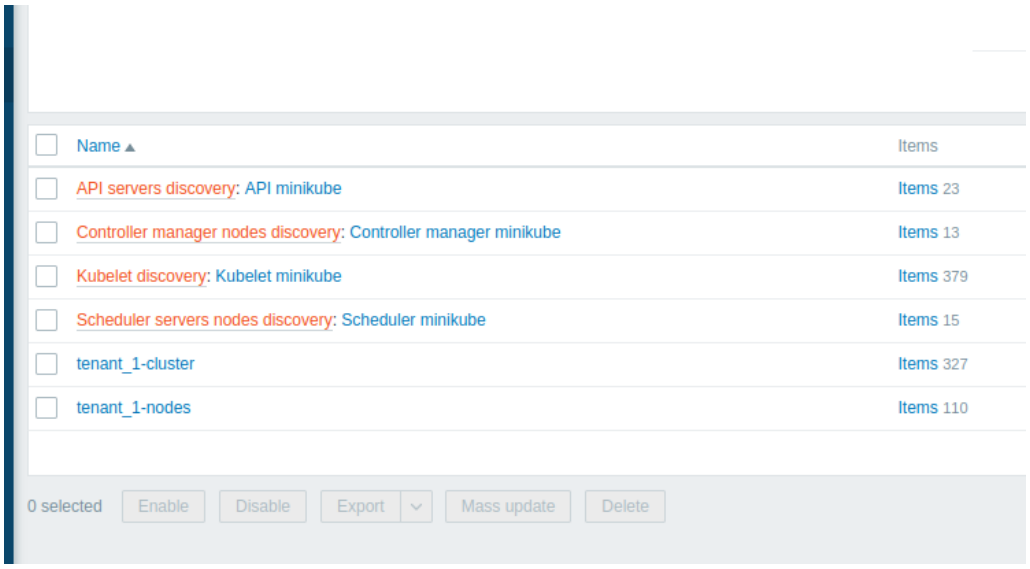
På samme måte som ved implementasjonen for generisk data, skal applikasjonen ta inn informasjonen og automatisk sette opp det nødvendige innad i Zabbix. Løsningen baserer seg på grensesnittet, der et skjema må fylles ut med alt av nødvendig informasjon, før dette omformes til en JSON-fil og sendes til back-end og legges automatisk inn.

I tillegg til å endre proxy-navn i Helm-Chartet krever løsningen, slik testmiljøet er satt opp, at det blir gjort endringer i brannmuren for at det skal fungere. Spesifikt må det gjøres endringer i Port-forwarding tabellen til serveren, eller klienten Clustert kjører på. For at Zabbix-serveren skal kunne svare på request er de nødvendig at det er toveis kommunikasjon mellom proxy og server. Ettersom Clusteret ligger på maskinen med en intern IP-adresse, som illustrert i figur 19, må man legge inn en port-forwarding i IP-tabellen for å sikre at alt som sendes til port 8443 på serveren videresendes til port 8443 på Clusteret. Port 8443 er standard port som åpnes for Zabbix-Proxy.



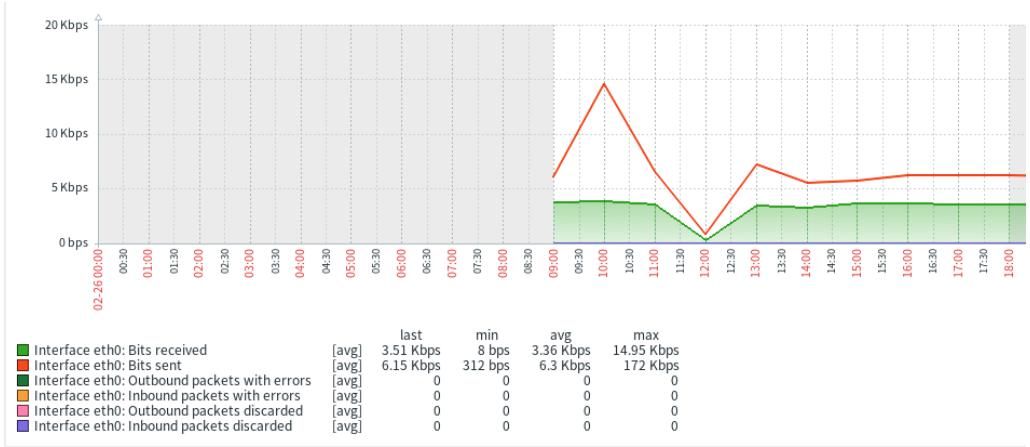
Figur 19: Kommunikasjon mellom Zabbix Server og Kubernetes Cluster. Figuren beskriver port-forwarding som var nødvendig for kommunikasjon mellom Zabbix-server og Zabbix-Proxy

Etter riktig installasjon av Helm-Chart og innsending av utfylt skjema kan man se i Zabbix at Clustert dukker opp som Host og begynner å motta data. Template inneholder alt nødvendige for at de forskjellige itemene skal få korrekte verdier, live fra Clusteret.



Figur 20: Skjermutklipp fra datakilder i Zabbix frontend. Alt i bildet tilhører en Kubernetes monitorering og genereres automatisk når man benytter template.

Figur 20 viser at Clusteret (tenant_1-cluster) har mottatt over 300 metrics i form av items med live verdier for tilstanden til Clusteret og alle dens podder. Dersom en tenant overvåker flere Cluster, kan man filtrere disse på tilhørende proxy, ettersom den er unik for hvert Cluster. Noen eksempler på metrics som blir hentet er CPU-bruk, antall podder, pod-status og status på kontainere. Templatet som blir benyttet for oppsett av monitoreringen, har predefinerte widgets for visualisering av ulike metrics. Figur 21 under viser eksempel på en slik widget, der nettverkstrafikk vises som en graf.

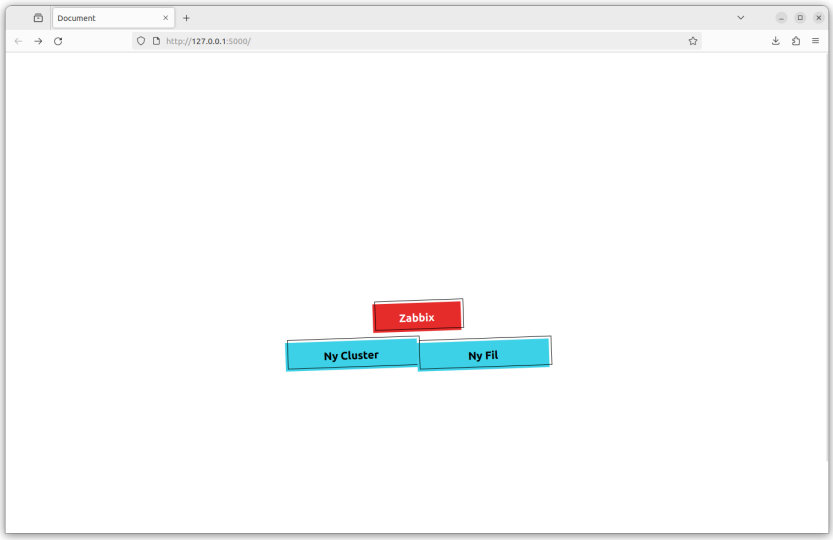


Figur 21: Predefinert widget i Zabbix dashboard for nettverkstrafikk i Cluster.

5.5 Grensesnittet

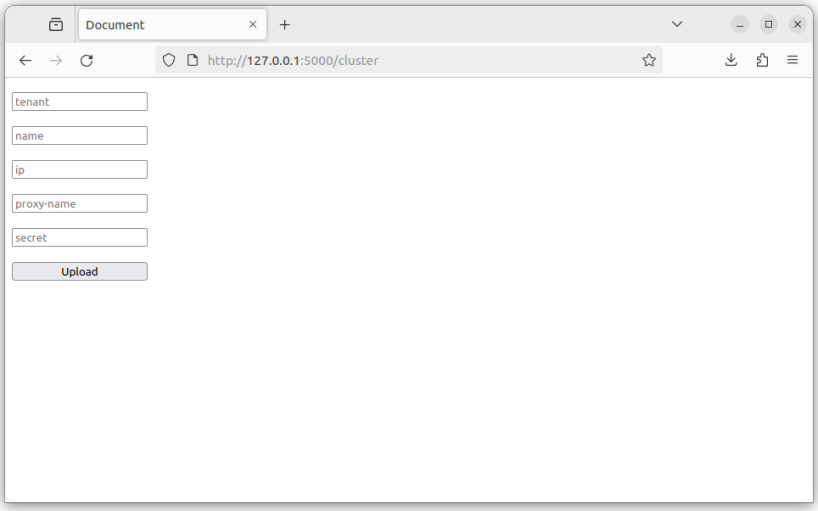
Brukeropplevelsen er ikke hovedfokus i vår PoC. Likevel valgte vi å utforme et enkelt brukergrensesnitt for å bevise hvor enkelt applikasjonen kan brukes. For å bygge grensesnittet ble det benyttet Flask, som muliggjorde et raskt og enkelt oppsett av et simpelt

webbasert grensesnitt. Grensesnittet tilbyr tre valg, som vist i figuren 22. Zabbix valget sender deg til Zabbix Frontend. De to andre valgene er innlasting av datafiler (Ny Fil), eller opprette Kubernetes-overvåking (Ny Cluster).



Figur 22: Egenutviklet webgrensesnitt bestående av tre valg.

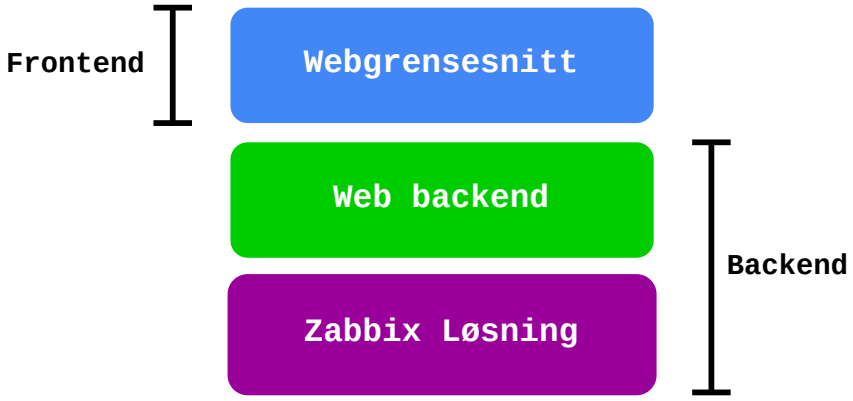
Ved innlasting av datafiler, vil brukeren kunne laste opp datafiler til grensesnittet, som sendes til applikasjonen og videre legges inn i Zabbix dersom kravene er fulgt. Ved Kubernetes-overvåking vil brukeren bli presentert et skjema, vist i figur 23. Når skjemaet er ferdig utfylt og sendes inn, vil grensesnittet omforme svarene til en JSON-fil og sende denne videre til applikasjonen for behandling. Ved å omforme dataene til en JSON-fil, kan brukeren i teorien lage denne filen selv og manuelt laste den opp.



Figur 23: Skjema i grensesnitt for å legge til nytt Cluster

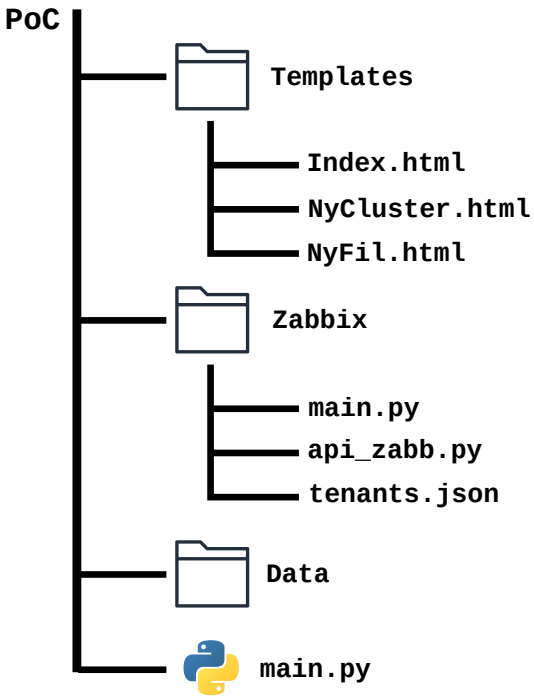
5.6 Utforming av koden

Utformingen av koden og vår løsning kan deles inn i frontend og backend. Her består backend av to lag, illustrert i figur 24. Web backend håndterer forespørslene fra grensesnittet, som ruting og dataklargjøring. Zabbix løsningen gjennomfører alt som er nødvendig for at filene som lastes opp legges riktig inn i Zabbix. Den mest omfattende av de tre lagene er Zabbix løsningen, da logikken og utformingen av denne prosessen er betydelig mer detaljert og omfattende enn de to overordnede.



Figur 24: Stacken i hvordan koden er utformet bestående, av front-end (Webgrensesnitt) og Backend (Web backend og Zabbix Løsning)

Koden vår har en filstruktur som har en annerledes utforming. Overordnet består den av tre mapper, med sine respektive filer, og et Python-script, som illustrert i figur 25. Som vist i figuren under, er *main.py* plassert utenfor mappene. Dette er Web backend som krever at templates mappen befinner seg på samme nivå i filstrukturen for å fungere. Videre ble det valgt å legge Zabbix løsningen i en egen mappe for å skaffe et klart skille mellom hva som hører til hva. Data mappen er lokasjonen der opplastede datafiler legges.



Figur 25: Filstrukturen i koden

5.6.1 Webgrensesnittet

Webgrensesnittet består av tre filer, eller sider, som er utformet med relativt simpel HTML og CSS kode. CSS koden er strengt tatt ikke nødvendig, da den bare endrer visuelt på nettsiden, men er valgt å ta med likevel for å forbedre brukeropplevelsen. Filen *index.html* er hjemmesiden til applikasjonen og består av tre elementer; Zabbix, Ny Fil og Ny Cluster. Med CSS benyttes det flexbox for å sentrere elementene på nettsiden, i tillegg til noe stilsetting på knappene og overskriften.

```
1 <div>
2     <h2 class="zabb">Zabbix</h2>
3 </div>
4 <div id="button">
5     <a href="/cluster" class="button-53">Ny Cluster</a>
6     <a href="/fil" class="button-53"> Ny Fil</a>
7 </div>
```

Linje 5 og 6 i HTML-delen over viser knappene på nettsiden. Dette er linker som videresender brukeren til */Cluster* og */fil*. Dette er ruter som blir behandlet av web backend, noe som ses nærmere på i kapittel 5.6.2.

NyCluster.html og *NyFil.html* er begge enkle sider med skjemaer der svaret som sendes inn behandles av web backend. Som linje 1 i HTML-delen under viser, benytter de *action* og *method="post"* for at backend skal kunne behandle dataene.

```
1 <form action = "/fil" method = "post" enctype="multipart/form-data">
```

```
    <div class="button-wrap">
        <label class="button" for="upload">Upload File</label>
        <input id="upload" type="file" name="file" multiple>
    </div>
    <Button type="submit" class="button-53" value="Upload">Upload</
    Button>
</form>
```

5.6.2 Web Backend

Web backend er i hovedsak bygd opp av main.py filen nederst i figur 25. Rutene i web-grensesnittet er definerte til å følge ruten til den knappen som trykkes på. I eksempelet under ser man et eksempel på en slik rute.

```
@app.route("/fil", methods = ["GET", "POST"])
def nyFil():
    if request.method == 'GET':
        return render_template("nyFil.html")
    if request.method == 'POST':
        files = request.files.getlist('file')
        for fil in files:
            #Transformer til JSON og send til Zabbix
        return render_template("index.html", name = fil.filename)
```

Linje 1 i koden over definerer hvilke endepunkt denne ruten tilhører, i dette tilfellet */fil*. I tillegg til ruten defineres tillate metoder i form av *GET* og *POST*. Videre har vi spesifisert hva programmet skal gjøre ved de forskjellige metodene. I dette tilfellet vil programmet ved en *GET* forespørsel returnere siden *NyFil.html*. Ved *POST* forespørsel vil den hente filen/filene som lastes opp, deretter lagre filen, og sende filbanen til Zabbix løsningen. Til slutt vil programmet returnere hjemmesiden til brukeren.

For å implementere støtte for flere filtyper enn JSON og YAML, benyttet vi som nevnt Pandas biblioteket for å transformere filene som lastes opp til JSON. Koden under viser et eksempel på hvordan transformering av filer gjennomføres ved å benytte Pandas.

```
import pandas as pd

if fil.filename.split(".")[1] == "csv":
    pd_fil = pd.read_csv(fil)
elif fil.filename.split(".")[1] == "html":
    pd_fil = pd.read_html(fil)[0]

pd_fil.to_json(filpath, orient="table")

json_object = {
    "type":"structured",
    "source":"table",
    "tenant": request.form["tenant"],
    "name" : "filnavn",
    "items":[]
```

```
16 }
17
18 with open(filpath, "r") as jsonFile:
19     data = json.load(jsonFile)
20
21 json_object["items"].append(data["data"])
```

Som vist benyttes *pandas.to_json()* for å transformere objektene til JSON, for deretter legges i *json_object* som vist i linje 10, 18 og 21.

5.6.3 Zabbix løsningen

Overordnet er logikken for denne løsningen representert i figur 16. All logikken i koden skjer i *main.py* under mappen *Zabbix*. For å kunne holde oversikt over alle API-kallene som benyttes valgte vi å legge de i en egen fil (*api_zabb.py*), der hvert API-kall har en spesifikk funksjon og "method", beskrevet i koden.

```
1  def lag_hostgruppe(hostgroupnavn):
2      data = {
3          "jsonrpc": "2.0",
4          "method": "hostgroup.create",
5          "params": {
6              "name": hostgroupnavn
7          },
8          "id": 1
9      }
10     auth_tok = f"Bearer {auth()}"
11     resp = requests.get(link, headers = {"content-type" : "application/
       json-rpc", "Authorization": auth_tok}, json = data)
12     id_svar = resp.json()
13     return id_svar["result"]["groupids"][0]
```

Koden over er et eksempel på en slik funksjon. Metoden som benyttes er beskrevet i linje 4. Denne funksjonen henter navnet på Host-groupen som skal lages, for deretter å senere bruke dette i API-kallet til Zabbix-serveren. Funksjonen itererer igjennom svarene og returnerer Host-group id. *api_zabb.py* inneholder over 20 slike metoder, som alle gjør nødvendige operasjoner for at tjenesten skal fungere. Mange av funksjonene benytter selv API-kall, spesielt funksjoner som trenger *host_id*. Vår løsning benytter i hovedsak Host-navn når det refereres til Hosts, slik at ved API-kall der *host_id* er nødvendig, henter vi dette med *host.get* metoden fra Zabbix Serveren.

Etter at brukeren har lastet opp en fil gjennomføres det en sjekk for å sikre at filtypen støttes. Filtyper som YAML og JSON kan enkelt lastes rett til løsningen, slik at de kan behandles uten bruk av tilleggspakker. Dette er vist i koden under.

```
1 if filnavn.split(".")[1] == "json":
2     data = json.load(f)
3 elif filnavn.split(".")[1] == "yml" or filnavn.split(".")[1] == "yaml":
4     data = yaml.safe_load(f)
```

```
5 else:
6     return 0
```

Logikken i koden baserer seg på at programmet leser igjennom datafilen som lastes opp og søker etter forhåndsdefinerte datafelt. Datafeltene innebærer *type*, *source* og tenant for hvert item.

```
1 if data["type"] == "vm" and data["source"] == "vcenter":
2 for vm in data["vm"]:
3     eksisterende_tenants = eksisterendeTenants(tenantsJsonPath)
4     tenants_zabbix = api_zabb.hvilke_hostgroup_finnes()
5     if tenantSjekkOgFiks(vm["tenant"], eksisterende_tenants,
     tenants_zabbix, tenantsJsonPath):
6         hostSjekkOgFiks(vm["identity"]["name"], vm["tenant"], vm)
7         api_zabb.dashboard_fiks(vm["tenant"])
```

Koden over beskriver hvordan programmet sjekker for de definerte datafeltene, for deretter å hente ut de riktige datafelt. Basert på hvilke *source* og *type* vil dataene ha noe forskjellig struktur, så ved å ha definert disse kan man sikre at alle ønskelige data kan forstås. Den mer generiske funksjonen er noe likt utformet, men inneholder mer generiske datafelt, som *item* istedenfor *vm*, vist i linje 2. Linje 5 og 6 i koden kaller funksjonene *tenantSjekkOgFiks()* og *hostSjekkOgFiks()*, som er funksjoner som sørger for at tenanten og Hosten legges inn i både Zabbix og lokal database. Linje 7 kaller *dashboard_fiks()*, med tenant som parameter, for å oppdatere dashboardet til tenanten, slik at dataene presenteres.

For Kubernetes-overvåking opprettes to Hosts basert på to predefinerte templates i Zabbix. Deretter vil koden oppdatere verdier på Hosten og legge inn en ny Zabbix-proxy slik at Zabbix Server kan kommuniserer med agenten på Clusteret. Hvordan Hosten oppdateres er beskrevet i koden under.

```
1 def link_kubernetes_template(hostid, template, secret, ip):
2     data = {
3             "jsonrpc": "2.0",
4             "method": "host.update",
5             "params": {
6                 "hostid": hostid,
7                 "templates": [
8                     {
9                          "templateid" : hent_template_id(template)
10                    }
11                ],
12                "macros": [
13                    {
14                        "macro": "{$KUBE.API.TOKEN}",
15                        "value": secret
16                    },
17                    {
18                        "macro": "{$KUBE.API.URL}",
19                        "value": f"https://{ip}:8443"
20                    }
21                ]
```

```
22            },
23            "auth": auth(),
24            "id": 1
25        }
26     resp = requests.get(link, headers = {"content-type" : "application/
       json-rpc"}, json = data)
27     return(resp.json())
```

De verdiene som sendes inn via grensesnittet sendes videre til vår løsning via paramatere i funksjonskallet. Som vist i koden over tar funksjonen fire paramatere, som den deretter bruker for å legge inn riktige verdier på rikige steder. Macros (linje 12) er forhåndsdefinerte parametre til templaten, som må tilegnes en verdi via *host.update*.

5.7 Resultat PoC løsning

I dette kapitlet legger vi fram hva vi har fått til med løsningen. For å gi en intuitiv forklaring av hva som er resultatet av løsningen har vi valgt å forklare det med et scenario. La oss si du er del av en bedrift som jobber med programvareutvikling. I denne bedriften jobber flere team med flere ulike prosjekter. Du er teamleder for gruppen ”CodePulse” som jobber på et applikasjonsprosjekt. Du har et ansvar om å holde oversikt over alle IT-ressurser som ditt team bruker og trenger. Dette innebærer å sørge for at teamet ditt har oversikt over hvilke ressurser som er tilgjengelige, hvordan de er konfigurert, samt å opprettholde en overvåkning som forteller gruppen om ressursen fungerer som de skal. De fysiske ressursene som gruppen CodePulse benytter er to servere, åtte virtuelle maskiner, fem fysiske maskiner og et Kubernetes Cluster. Serveren som kjører CodePulse sine VMer, kjører også VMer for to andre grupper. Serverene du benytter deler også ressurser med andre grupper, tilsvarende gjør Kubernetes instansen. Du ønsker at CodePulse kun skal se data som tilhører sin gruppe. Med vår løsning skal du som teamleder kunne sørge for en komplett og oppdatert visning av alle IT-ressurser som er kritiske for teamets operasjoner, alt fra ett sted. Løsningen har et dashboard for hver tenant i systemet, der du får en oversikt over tilstanden til IT-ressursene, og hvordan ressursen faktisk er konfigurert. Denne informasjonen hentes fra blant annet JSON- og YAML-filer som inneholder konfigurasjonsdata og statusoppdateringer. CodePulse får en proaktiv promblemhåndtering som gir muligheten til å overvåke ressursstatus i sanntid og ha tilgang på nødvendig konfigurasjonsdata for gruppens ressurser. Dette tillater gruppen å oppdage problemer og utføre handlinger raskt som bidrar til å forebygge nedetid og fokus på applikasjonsprosjektet. Du som avdelingsleder lager et script som henter data fra alle de aktuelle kildene dine, og legger til informasjon om tenant, type og source. Du sender filene til vårt program gjennom interfacet vårt, dette lager automatisk dashboard til tenants, samt widgets med oversikt over dataene dine. Løsningen presenterer data du sender i HTML. Løsningen støtter følgende filformater:

Tabell 2: Oversikt over støttede språk/formater, grønn viser til at formatet er testet og verifisert at fungere.

Format	Data Beskrivelse	Suffix	Testet
text	CSV	.csv	
text	Fixed-width text file	.txt	
text	JSON	.json	
text	HTML	.html	
text	XML	.xml	
text	YAML	.yaml / .yml	
binært	MS Excel	.xlsx	
binært	OpenDocument	.odt	
binært	Feather format	.feather	
binært	Parquet format	.parquet	
binært	Sata	.dta	
binært	Python Pickle format	.py	

Selv om løsningen støtter disse formatene, er det viktig å notere at dataene som lastes opp enten må være strukturerte eller semi-strukturerte. Med strukturerte data mener vi tabeller inndelt i rader og kolonner, og med semi-strukturerte data menes filformater som JSON og YAML.

6 Diskusjon

6.1 Designvalg

Som det fremgår i kapittel 4.4.5 sammenlignet vi de ulike løsningene vi fikk presentert i en tabell. Det finnes mange flere muligheter enn hva vi evaluerte. Flere andre løsninger kom fram i intervjuet og i kapittel 4.2.3. Vi vurderte forslagene basert på om de var open-source, hvor allsidig og hvor egnet de var for å løse vår problemstilling. Det var viktig å avgrense valgene for oppgaven. De presenterte løsningene i kapittel 4.2.1 var løsninger som i hovedsak vår veileder presenterte. Under intervjuet ble vi spesielt foreslått Zabbix, basert på fleksibilitet. Respondentens meninger ble tatt i betraktning, med bakgrunn i hans lange erfaring innen IT-drift. Vi tok også i betraktning at ett intervju gir oss et smalt narrativ. Vi analyserte derfor flere muligheter. Designvalget mellom de ulike mulige løsningene baserte vi på egendefinerte sammenligningsfaktorer. Sammenligningsfaktorene utformet vi basert på oppgaveteksten og vår analyse av den. Faktorene skal klargjøre i hvilken grad det er praktisk mulig å anvende verktøyet for å utvikle en løsning i form av et PoC. Ulempen med måten vi valgte verktøy på er at den begrenses av fagfolk sine erfaringer. Vi gjorde ikke en helhetlig litteraturstudie med valg basert på tidligere forskning. Dette kunne gjort valget mer objektivt, men ville vært mer tidkrevende. Vi gjorde enkle litteratursøk på de presenterte verktøyene for å sammenligne de og sjekke oppslutningen rundt disse open-source løsningene. Dette gav oss allikevel derfor en form for objektivitet rundt hvor populære og gode løsningene var omtalt som.

Valget falt på Zabbix for å basere løsningen rundt. Valget av Zabbix var også basert på vår egen estimering av hva som virket mest gjennomførbart med tanke på tidsperspektivet til denne oppgaven. Å benytte tid som en faktor i valget er nødvendig, men svekker samtidig hvor grundig forskningsarbeidet rundt alternative løsninger er. Vi planla og gjennomførte en uke med testing og evaluering av verktøyene som ble presentert som mulige løsninger for oppgaven. I denne perioden fikk vi teoretisk og praktisk erfaring med hvordan tjenestene fungerte. Evalueringsfaktoren om hvor tidkrevende bruk av verktøyet ville være baserte vi på erfaringer fra denne uken med testing. Noen av tjenestene var for oss mindre intuitive og mer kompliserte som det fremgår i kapittel 4.4, der vi presenterer erfaringene og resultatene våre. Vi baserte vår vurdering på hvor intuitiv tjenesten var på å sette opp tjenestens standard funksjonalitet. Det kan argumenteres at en uke med testing er for lite for å danne et helhetlig bilde på hvor god løsningen oppleves som. Det kan også argumenteres for at løsning som oppleves komplisert kan ha større funksjonalitet og bredde i sin funksjonalitet, som gjør løsningen mer fleksibel og tilpasningsdyktig. En faktor som også var relevant i valget var oppdragsgivers ønsker. Valget om løsning var åpent så lenge det løser utfordringen presentert i oppgaveteksten. Allikevel var oppdragsgiver interessert i funksjonaliteten til Zabbix opp mot Kubernetes. Designvalget falt derfor på å utvikle en løsning som benytter Zabbix.

6.2 Datainnsamling og krav til data for multi-tenancy

I denne seksjonen vil vi diskutere gjennomførbarheten av datainnsamlingen og kravene. Vi har valgt en datainnsamling som er agentløs og baserer seg på å laste ned filer fra kilden. I test-sammenheng har vi manuelt lastet ned datasett og brukt et script for å tilføre source, type og tenant. Vi ser for oss en lignende datainnsamling i praksis. En virksomhet som vil benytte seg av løsningen, kan benytte et script som henter inn alle filene de ønsker å presentere. Dette scriptet kan deretter legge til source, type og tenant informasjon slik at det står i stil med kravene. Avhengig av oppdateringsfrekvensen på innsamlingen, kan dataene defineres som live data. Dette er en ideell måte å samle data uten å bruke agenter, ettersom det lar virksomheten teste løsningen uten å måtte installere noe fysisk på enhetene sine. Det er også mulig å utvikle en agent, men dette er tidskrevende og lite hensiktsmessig for å teste et konsept.

Vi har satt tre ulike krav til dataene. Kravet om *type* er nødvendig for at koden vet hva slags data den skal lese gjennom. Eksempelvis dersom dataobjektet er VMer vil programmet iterere gjennom alle VMene og finne ut hvilke tenant den tilhører. Tenant informasjonen er derfor også nødvendig å tilføre til de ulike objektene. *Source* eller kilde er sentralt for å presentere data fra riktig kilde i Zabbix dashboardet. Det kan finnes flere kilder som viser samme type objekter. Eksempelvis kan både Vsphere og Ansible inneholde VM objekter, men det er ønskelig å skille kildene i dashboardet. Vi har derfor satt disse tre minimumskravene for at dataene skal presenteres riktig i Zabbix. Det kan derimot settes flere krav for bedre presentasjon. Dersom data skal presenteres på annet enn slik vi gjør, med HTML, for eksempel med grafer eller søyler, er det nødvendig med mer informasjon om utformingen av formatet objektene er beskrevet på. Dersom Zabbix også skal utføre kommandoer vil det være nødvendig med mer informasjon til programmet vårt om strukturen på dataene. Eksempelvis dersom det er ønskelig å sende ping eller http forespørsler til IP adressene må det settes krav til hvordan dette skal utformes slik at det kan utvikles en integrasjon som viser informasjonen i Zabbix. Generelt dersom vi skal presentere dataene på mer detaljert og oversiktlig måte vil det være hensiktsmessig å bruke *versjon* som et krav til data som mottas. De ulike datakildene kan endre strukturen på hvordan formatet i filen er utformet, noe som kan føre til inkompatibilitet med Zabbix, dersom det er benyttet avanserte dashboard visninger gjennom predefinerte widgets. Eksempelvis kan data som hentes fra Vsphere endre helt på hvordan innholdet i filen du eksporter er strukturert, som kan føre til at funksjonene som er satt opp i Zabbix ikke vil fungere. Slik dataene presenteres nå har det ingen ting å si hvilken versjon av kilden som den mottar er. Krav om versjon kunne gjort det mulig å automatisere og lage predefinerte widgets i Zabbix som presenter data mer detaljert. Flere krav til dataene fra datainnsamlingen gir altså flere muligheter, men samtidig kompliserer det løsningen og kan føre til flere feilkilder. For oppgaven har vi valgt å fokusere på at konseptet om multi-tenancy skal fungere.

6.3 Praktisk bruk av verktøyet og resultatene

I dette kapitlet analyserer vi praktisk bruk av løsningen i en bedriftskontekst. Bedrifter som vurderer å implementere en tilsvarende løsning, møter en rekke valg, spesielt med tanke på hvilke deler av infrastrukturen som skal integreres. Våre resultater demonstrerer at løsningen effektivt kan visualisere et bredt spekter av informasjon om dataobjekter på dashboardet. Gjennom vår PoC har vi vist at det er mulig å differensiere data for å oppnå en multi-tenant dashboard-løsning. Dette indikerer at bedrifter kan dra nytte av mange muligheter, selv om brukervennligheten og kvaliteten på visualiseringer kan være begrenset hvis det ikke er klarhet i hvilke data som skal presenteres. Det er derfor gunstig for bedrifter å definere spesifikke tjenester og systeminformasjon som skal integreres, noe som muliggjør bruk av forhåndsdefinerte widgets. Disse widgets kan også støtte opprettelse av varsler for anomalideteksjon og og samtidig gi mulighet for å lage en mer interaktiv dashboardopplevelse, som kan utføre oppgaver som ping og http-forespørsler.

Ved integrering av nye tjenester i vår løsning, der man kjenner strukturen av informasjonen, kan bedrifter øke funksjonaliteten og brukervennligheten utover en ren tekstbasert visning. Samtidig krever opprettholdelsen av et multi-tenancy system at data, som omfatter informasjon fra flere brukere, tillegges parametere som muliggjør sortering, prosessering og fremvisning. Data som ikke faller under denne kategorien, kan integreres i Zabbix på måter som er foretrukket av tjenesten, vanligvis gjennom bruk av en agent for datainnsamling. I vår studie har vi integrert Kubernetes med Zabbix på en konvensjonell måte gjennom Zabbix sine foretrukne innsamlingsmetoder. Alternativt har vi bevist at det er mulig å integrere nokså tilfeldig data fra filer og sende dem via Zabbix API ved å benytte Trapper-Item-funksjonen, dersom bruk av en agent ikke er mulig, lønnsom eller hensiktsmessig.

For å utnytte vår løsning i praksis, kreves det at en Zabbix-server er installert på en serverenhet, og at en administrativ klient i nettverket kan kjøre vårt script for å sende data til serveren. Basert på bedriftens infrastruktur må det tas hensyn til hvor dataene som skal vises, lagres. Dette kan involvere opprettelse av en dedikert database for dataene, som vi har sett er vanlig i en CMDB, eller lagring i Git. Dette understreker viktigheten av god planlegging for å opprettholde en effektiv infrastruktur.

Når det gjelder brukeropplevelsen, har vi prioritert tilgjengelighet og enkelhet. Vi har som tidligere nevnt, valgt å presentere data i HTML-format for å maksimere lesbarheten og tilgjengeligheten av informasjonen. Selv om dette valget begrenser mulighete i visualiseringene, har vi forbedret brukeropplevelsen ved å tilby en mer visuell presentasjon enn det originale dataformatet. Samtidig har vi også testet implementasjon av Kubernetes Clustere og sett at dette kan gi relativt gode visualiseringer.

Selv om vi i denne oppgaven har fokusert på å møte spesifikke krav og demonstrere PoC, kunne vi ha vektlagt brukeropplevelsen og sikkerhetsaspektene mer. Dette ville være kritisk for et ferdig produkt. Gitt tidsbegrensningene valgte vi å prioritere oppfyllelsen av de grunnleggende kravene til løsningen. I slutten av oppgaven diskuterer vi løsningens

videre utviklingspotensial, i kapitlet om videre arbeid.

6.4 Data fra mange kilder

Vi vil ta for oss utfordringene og valgene vi har gjort ved presentasjon av data samlet fra et bredt spekter av kilder. Data fra mange kilder refererer til informasjon samlet fra diverse programvare-instanser distribuert i en virksomhets nettverksinfrastruktur. Vi valgte en agentløs tilnærming til datainnhenting som var hensiktsmessig i utviklingen av en PoC. Datainnsamlingen er illustrert i figur 2. Som tidligere diskutert i kapittel 6.2, må brukeren av løsningen benytte seg av et script som henter inn data og tilfører parametere. Den løsningsspesifikke utfordringen for håndtering av data fra mange kilder er derfor å støtte mange filtyper. I utviklingen av vårt program valgte vi å fokusere på JSON og YAML som de primære dataformatene for innlesing. Dette valget ble gjort med bakgrunn i disse formatenes utbredelse og relevans i dagens moderne IT-landskapet, samt deres fleksibilitet og lesbarhet. For å inkludere data fra andre formater, benyttet vi en mekanisme som konverterer disse formatene til JSON, før de prosesseres. Dette skrittet sikrer at vi kan håndtere data fra mange ulike kilder, uavhengig av deres opprinnelige format. Ved å anvende et transformasjonsverktøy for å konvertere kjente formater til JSON, adresserer vi behovet for en fleksibel løsning som kan håndtere mange datakilder. Denne tilnærmingen gjør det mulig for oss å samle og presentere et bredt utvalg av data.

Vi brukte Pandas for datatransformasjon. Vi valgte Pandas fordi vi ønsket et verktøy som kunne implementeres i koden, for å unngå ekstern datahåndtering og dermed minske kompleksitet. Vi valgte å bruke Pandas til å transformere alle filer til JSON format. Ved å transformere dataene til JSON, for deretter la koden vår håndtere Zabbix implementasjonen, vil man ved få kodelinjer kunne støtte en rekke flere filtyper med et felles standpunkt. Dette forenklet kodingen. Pandas har støtte for 19 ulike dataformater, der vi har testet 12. Vi møter begrensninger med at Pandas støtter bare et begrenset sett av filformater som lar seg overføres til deres tabulære datamodell [78]. Dataene som samles inn må enten være strukturert data eller semistrukturert data. Denne begrensningen påvirker hvilke datakilder vi kan integreres mot løsningen. Det kan argumenteres for at de aller fleste dataformatene som beskriver tilstand og konfigurasjon i et IT-landskap dekkes. Zabbix i seg selv er utformet for å være en allsidig løsning og har støtte for mange andre implementasjoner som ikke dekkes av filformater vi støtter i vår løsning. Ettersom Kubernetes er mye brukt i dagens IT verden har vi også inkludert live overvåkning av Kubernetes Clustere i løsningen, der hvert Cluster tilhører en tenant. Det kan altså være flere Cluster for samme tenant. Løsningen støtter Cluster overvåking og presentasjon av tilstand- og konfigurasjonsdata fra server, klient, VM og Kubernetes så lenge dataene er strukturert eller semistrukturert, og finnes på et av de 12 formatene vi har testet.

6.5 Metodevalg

I vårt metodevalg bygget vi på trinnene i en teknologiforskningsmetode, som er en iterativ prosess, og tilpasset denne med egendefinerte steg. Denne tilnærmingen viste seg å være effektiv, og ga oss verdifulle startverdier som grunnlag for å begynne utviklingsprosessen. De innledende fasene, som var analyse, kravsetting og informasjonsinnhenting, var viktige faser for å avgrense oppgaven og tidlig kunne sette i gang med utvikling av artefaktet. Oppgaven fokuserer på et meget spesifikt og nisje problemområde som kan ha mange løsninger. For å sikre progresjon i arbeidet, var det essensielt å etablere klare krav til løsningen. Disse kravene begrenset valgmulighetene for artefaktets utforming, noe som forenklet utviklingsprosessen. Samtidig medførte det at vi begrenset oss til å kun utforske løsningene presentert i oppgaven. Dette kan potensielt tidlig låse oppgaven til en spesifikk tilnærming, og risikoen er at løsningen kan være ineffektiv uten vår viten. Vi utformet oppgaven med utgangspunkt i å videreutvikle en eksisterende programvare, og stilte krav til hva denne skulle tilby. Hadde vi valgt å utvikle en egen-laget løsning, kunne det muligens ha resultert i en mer fleksibel løsning, bedre tilpasset sluttbrukeren. Det er imidlertid utfordrende å vurdere om vi kunne ha levert et slikt produkt innenfor de gitte tidsrammene, gitt vår manglende erfaring med å utvikle ny programvare fra bunnen av.

Den iterative delen av metoden tillot kontinuerlig testing, evaluering og utvikling. Denne tilnærmingen muliggjorde hyppige tester av ulike verktøy, funksjoner og strategier, og vi kunne dermed kontinuerlig vurdere hva som fungerte godt og mindre godt. Dette gjorde det mulig for oss å stadig forbedre løsningen i tråd med de oppsatte kravene. Selv om den iterative prosessen viste seg å være tidkrevende, var den verdifull. I situasjoner der vi oppdaget at metoder kunne forbedres, førte det ofte til at vi måtte revidere en betydelig mengde arbeid. Dette var imidlertid essensielt for å forbedre kvaliteten på den endelige løsningen. Metoden vår har hjulpet oss å lage en velutviklet og pålitelig PoC løsning.

[Denne siden er blank med hensikt]

7 Konklusjon

Hovedmålet med denne oppgaven har vært å besvare problemstillingen og utvikle et artefakt som adresserer utfordringen: *"Hvordan kan man samle inn data fra flere kilder og distribuere informasjonen om infrastrukturen, slik at brukerne enkelt kan få informasjonen om ressursene de bruker?"*. Gjennom de seks fasene av oppgaven har vi formulert krav til løsningen, samlet empiri og resultater, og utviklet et produkt som løser denne utfordringen.

I Forsvaret foreligger det et stort behov for god oversikt over informasjonsinfrastrukturen. For å holde oversikt i dagens IT-miljø er det lønnsomt å samle informasjon til en plattform. Det er et behov for en allsidig og fleksibel løsning som kan presentere data fra mange ulike kilder. Det viser seg at oversikt over infrastruktur også kan være en utfordring i det sivile. I denne oppgaven ha vi utelukkende konsentrert oss om perspektiver fra det sivile. I dagens IT-miljø deler ofte flere avdelinger samme fysiske infrastruktur for å utføre oppgaver. Dette gjør at kun IT-personell, som styrer sikkerheten, har full oversikt over infrastrukturen, mens ansatte i de ulike avdelingene har begrenset innsikt i ressursene de bruker. I denne oppgaven har vi analysert og utforsket hvordan en bedrift kan samle inn data fra mange kilder og distribuere informasjonen om infrastrukturen, slik at brukerne enkelt kan få informasjonen om ressursene de bruker.

I løpet av denne oppgaven har vi utforsket ulike programvarer, inkludert Zabbix, Grafana og InfluxDB, for å utvikle en multi-tenant dashbordløsning. Valget falt på Zabbix, som tilbyr et robust grunnlag for å utvikle en fleksibel og open-source løsning. Med et velutviklet API, muliggjør den effektiv integrasjon av generisk data fra ulike kilder. Videre tilbyr Zabbix avanserte funksjoner for brukerhåndtering og støtter overvåking av Kubernetes.

For å håndtere data fra mange kilder, benytter vi et script for datainnsamling som henter datafiler og tilfører parametere som source, type og tenant. Løsningen vår kan prosessere data fra mange ulike kilder, ved at den støtter mottak av en 12 ulike filformater, presentert i tabell 5.7. Dataene transformeres til et felles filformat før det prosesseres, sendes og presenteres i Zabbix. Dataene presenteres i tekstformat, med HTML. Zabbix tilbyr en effektiv måte å implementere Kubernets overvåking med et Helm-Chart. Løsningen vår automatiserer prosessen videre og setter opp nødvendig konfigurasjon mot Zabbix for et multi-tenant driftsmiljø.

Resultatene viser at med Zabbix som et grunnlag, sammen med script for datainnsamling og tilføring av parametere, kan vi effektivt samle inn data fra flere kilder og distribuere informasjon om infrastrukturen, slik at brukerne enkelt kan få informasjon om ressursene de bruker. Denne løsningen er en av flere mulige tilnærminger for håndtering og distribuering av data fra mange kilder.

Gjennom en PoC løsning har vi bevist at vår løsning kunne håndtere og differensiere data på tvers av flere brukere og presentere data fra mange ulike formater. Gjennom strategisk bruk av APIer og scripting har vi kunnet adressere og overvinne utfordringene knyttet til

datainnsamling og presentasjon av informasjon på tvers av flere plattformer og format. Vår løsning demonstrerer ikke bare en teknisk mulighet, men også en forbedret operativ effekt gjennom å tilby en virksomhet distribuert systeminformasjon for å raskere oppdage problemer, slik at handlinger kan håndteres raskt, og man forebygger nedetid.

Vi benyttet en iterativ teknologiforskningsmetode med egendefinerte steg. Fordelene inkluderte kontinuerlig testing og evaluering, som forbedret løsningen gjennom hyppige justeringer. Innledende faser med analyse og kravsetting var avgjørende for å raskt komme i gang med utviklingen. Ulempene var at prosessen var tidkrevende og krevde revisjon av mye arbeid. Fokuseringen på et spesifikt problemområde begrenset løsningens fleksibilitet, men ga en mer strukturert og effektiv utviklingsprosess.

7.1 Videre arbeid

Som videre arbeid til oppgaven vil man behøve å adressere en rekke utbedringer og forbedringer før løsningen kan etableres i en virksomhet. Løsningen vi har lagt fram er et PoC og det er flere funksjoner og aspekter som ikke er adressert.

Følgende områder bør prioriteres for videre utvikling:

- **Robust:** Til å begynne med bør koden gjøres mer robust for håndtering av datafiler som ikke støtter kravene, samt ved diverse feil i løsningen. Feil bruk av løsningen kan gå på bekostning av både sikkerhet og tilgjengelighet. For å sikre løsningens robusthet bør man implementere mer redundans og ”sjekker” i koden. Det kan også med fordel presentere brukeren for feilmeldinger dersom de oppstår.
- **Predefinerte Widgets:** I løsningen var fokuset på konsptet og mindre på visuell fremstilling av dataene. Man kan med fordel se videre på utforming av predefinerte widgets for kjente komponenter og dataobjekter fra filer. Dette kan være et omfattende arbeid, dersom det er veldig mange forskjellige kilder. Til gjengjeld kan man få gode grafiske fremstillinger av dataene.
- **Versjonskrav til data:** En forutsetning for predefinerte widgets er at datatypene som blir sendt til løsningen, samt informasjonen de inneholder, er kjent på forhånd. Dersom metric-navn og struktur fraviker fra det som er definert i Zabbix, vil man ikke kunne få en grafisk fremstilling av dataene. Versjonskrav til data bør fortelle hvilken utforming, struktur og dataobjekter filen inneholder.
- **Docker Composite:** Det kan være lønnsomt å utforske Docker Composite for å forbedre løsningens implementasjon og skalerbarhet. Ved å benytte Docker til løsningen, kan man skalere opp og ned tjenesten ved behov. Ved å benytte Docker composite kan man enkelt distribuere og implementere løsningen der det skulle være ønskelig.

I tillegg til dette bør man, før løsningen etableres, se på sikkerhetsaspektene rundt løsningen. Det vil være viktig å ta nødvendige forholdsregler for å beskytte både data og systemet mot potensielle trusler. Dette inkluderer å implementere tiltak som autentiseringsmetoder og loggføring. Til slutt må det adresseres at det finnes mange andre mulige løsninger på oppgaven vår.

8 Referanseliste

References

[1] Forsvaret, 'Forsvarssjefens fagmilitære råd 2023', 2023. Tilgjengelig: https://www.forsvaret.no/aktuelt-og-presse/publikasjoner/fagmilitaert-rad/bilder-og-video/Forsvaret-FMR-2023.pdf/_/attachment/inline/c9147b67-7913-48ef-ac78-e61a2805f9a0:fd23bf41d3431040024613dfb377c033d84e2796/Forsvaret-FMR-2023.pdf (Lasted ned: 09/02/2024).

[2] Rikrevisjonen, 'Riksrevisjonens undersøkelse av forsvarets informasjons- systemer for kommunikasjon og informasjonsutveksling i operasjoner', 2023. Tilgjengelig: https://www.riksrevisjonen.no/globalassets/rapporter/NO-2022-2023/forsvarets-informasjonssystemer-ugradert-versjon.pdf (Lasted ned: 09/02/2024).

[3] Forsvarstaben, *Forsvarets Fellesoperative Doktrine*. Forsvaret, 2019.

[4] M. Yousif, 'Cloud-native applications—the journey continues', *IEEE Cloud Computing*, vol. 4, no. 5, s. 4–5, 2017. DOI: 10.1109/MCC.2017.4250930.

[5] A. Rico, 'Extending multi-tenant architectures: A database model for a multi-target support in saas applications', *Enterprise Information Systems*, vol. 10, no. 4, s. 400–421, 2016. DOI: 10.1080/17517575.2014.947636.

[6] L. Coyne *et al.*, *IBM Private, Public, and Hybrid Cloud Storage Solutions*. IBM Redbooks, 2018. Tilgjengelig: https://books.google.no/books?id=-L5YDwAAQBAJ.

[7] IBM, 'What is multi-tenant (or multitenancy)?', Tilgjengelig: https://www.ibm.com/topics/multi-tenant (Lasted ned: 05/04/2024).

[8] CloudflareINC, 'What is multitenancy?', Tilgjengelig: https://www.cloudflare.com/learning/cloud/what-is-multitenancy/ (Lasted ned: 05/04/2024).

[9] S. N. Z. Naqvi, S. Yfantidou og E. Zimányi, 'Time series databases and influxdb', *Université Libre de Bruxelles*, s. 1–44, 2017. Tilgjengelig: https://cs.ulb.ac.be/public/_media/teaching/influxdb_2017.pdf.

[10] P. Brooks, *Metrics for IT Service Management*. Van Haren Publishing, 2006, s. 1–34. Tilgjengelig: https://books.google.no/books?id=PldeAgAAQBAJ.

[11] AWS, 'What is structured data', Tilgjengelig: https://aws.amazon.com/what-is/structured-data/#:~:text=Structured%20data%20is%20data%20that,due%20to%20its%20quantitative%20nature. (Lasted ned: 14/05/2024).

[12] R. Olups, A. Dalle Vacche og P. Uytterhoeven, *Zabbix: Enterprise Network Monitoring Made Easy*. Packt Publishing, 2017. Tilgjengelig: https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1465681&site=ehost-live.

[13] Zabbix-LLC, 'Zabbix documentation - features', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/introduction/features (Lasted ned: 05/04/2024).

[14] R. Olups, *Zabbix 1.8 Network Monitoring*. Packt Pub., 2010. Tilgjengelig: https://books.google.no/books?id=fsjNquHrQfYC.

[15] Zabbix-LLC, 'Zabbix documentation - server', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/concepts/server (Lasted ned: 05/04/2024).

[16] P. Uytterhoeven, *Zabbix Cookbook*. Packt Publishing, 2015, s. 2–10. Tilgjengelig: https://books.google.no/books?id=gNqFBwAAQBAJ.

[17] Zabbix-LLC, 'Zabbix documentation - requirements', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/installation/requirements (Lasted ned: 05/04/2024).

[18] Zabbix-LLC, 'Zabbix documentation - proxy', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/concepts/proxy (Lasted ned: 05/04/2024).

[19] Zabbix-LLC, 'Zabbix documentation - agent', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/concepts/agent (Lasted ned: 05/04/2024).

[20] P. Uytterhoeven og R. Olups, *Zabbix 4 Network Monitoring: Monitor the performance of your network devices and applications using the all-new Zabbix 4.0, 3rd Edition*. Packt Publishing, 2019. Tilgjengelig: https://books.google.no/books?id=eyyFDwAAQBAJ.

[21] R. Szulist, 'Zabbix user permissions explained', 2021. Tilgjengelig: https://medium.com/@r.szulist/zabbix-user-permissions-explained-75c5cb78dc78 (Lasted ned: 05/04/2024).

[22] Zabbix-LLC, 'Zabbix documentation - dashboard', Tilgjengelig: https://www.zabbix.com/documentation/5.0/en/manual/web_interface/frontend_sections/monitoring/dashboard (Lasted ned: 05/04/2024).

[23] Zabbix-LLC, 'Zabbix documentation - api', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/api (Lasted ned: 05/04/2024).

[24] Zabbix-LLC, 'Zabbix documentation - trapper', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/config/items/itemtypes/trapper (Lasted ned: 05/04/2024).

[25] Zabbix-LLC, 'Zabbix documentation - sender', Tilgjengelig: https://www.zabbix.com/documentation/current/en/manual/concepts/sender (Lasted ned: 05/04/2024).

[26] ScriptBees, 'What is tick stack, and why should we consider using it?', 2020. Tilgjengelig: https://medium.com/@scriptbees/what-is-tick-stack-and-why-should-we-consider-using-it-5a2bddcd7b10 (Lasted ned: 05/04/2024).

[27] InfluxData, 'Telegraf', Tilgjengelig: https://www.influxdata.com/time-series-platform/telegraf/ (Lasted ned: 05/04/2024).

[28] InfluxData, 'Chronograf', Tilgjengelig: https://www.influxdata.com/time-series-platform/chronograf/ (Lasted ned: 05/04/2024).

[29] InfluxData, 'Kapacitor', Tilgjengelig: https://www.influxdata.com/time-series-platform/kapacitor/ (Lasted ned: 05/04/2024).

[30] InfluxData, 'Influxdb 1.x tickstack', Tilgjengelig: https://www.influxdata.com/time-series-platform/ (Lasted ned: 05/04/2024).

[31] InfluxData, 'Influxdata - client libraries', Tilgjengelig: https://www.influxdata.com/products/data-collection/influxdb-client-libraries/ (Lasted ned: 05/04/2024).

[32] InfluxData, 'Influxdata - scrapers', Tilgjengelig: https://www.influxdata.com/products/data-collection/scrapers/ (Lasted ned: 05/04/2024).

[33] InfluxData, 'Influxdata - ecosystem', Tilgjengelig: https://www.influxdata.com/products/data-collection/ecosystem/ (Lasted ned: 05/04/2024).

[34] InfluxData, 'Influxdata documentation - manage organizations', Tilgjengelig: https://docs.influxdata.com/influxdb/v2/admin/organizations/ (Lasted ned: 05/04/2024).

[35] InfluxData, 'Influxdata documentation - visualize data with the influxdb ui', Tilgjengelig: https://docs.influxdata.com/influxdb/v2/visualize-data/ (Lasted ned: 05/04/2024).

[36] InfluxData, 'Influxdb v2 docs - api', Tilgjengelig: https://docs.influxdata.com/influxdb/v2/api/ (Lasted ned: 05/04/2024).

[37] InfluxData, 'Influxdata documentation - api intro', Tilgjengelig: https://docs.influxdata.com/influxdb/v2/api-guide/api_intro/ (Lasted ned: 05/04/2024).

[38] E. Salituro, *Learn Grafana 10.x: A beginner's guide to practical data analytics, interactive dashboards, and observability*. Packt Publishing, 2023, s. 6–10. Tilgjengelig: https://books.google.no/books?id=a7PlEAAAQBAJ.

[39] Grafana-Labs, 'About grafana', Tilgjengelig: https://grafana.com/docs/grafana/latest/introduction/ (Lasted ned: 05/04/2024).

[40] A. Yeruva og V. Ramu, *End-to-End Observability with Grafana: A comprehensive guide to observability and performance visualization with Grafana (English Edition)*. Bpb Publications, 2023. Tilgjengelig: https://books.google.no/books?id=0vrKEAAAQBAJ.

[41] GrafanaLabsCommunity, *Grafana Loki documentation - Operations - Multi-tenancy*. Tilgjengelig: https://grafana.com/docs/loki/latest/operations/multi-tenancy/ (Lasted ned: 03/04/2024).

[42] Grafana-Labs, 'Grafana loki - multi-tenancy', Tilgjengelig: https://grafana.com/docs/loki/latest/operations/multi-tenancy/ (Lasted ned: 05/04/2024).

[43] Grafana-Labs, 'Grafana agent', Tilgjengelig: https://grafana.com/docs/agent/latest/ (Lasted ned: 05/04/2024).

[44] TheLinuxFoundation, 'Prometheus documentation - what is prometheus', Tilgjengelig: https://prometheus.io/docs/introduction/overview/ (Lasted ned: 05/04/2024).

[45] OpsRamp, 'Prometheus vs grafana: Knowing the difference', Tilgjengelig: https://www.opsramp.com/guides/prometheus-monitoring/prometheus-vs-grafana/ (Lasted ned: 05/04/2024).

[46] A. Boduch, *React and React Native*. Packt Publishing, 2017, s. 9–17. Tilgjengelig: https://books.google.no/books?id=jLkrDwAAQBAJ.

[47] M. Schwarzmuller, *React Key Concepts: Consolidate your knowledge of React's core features*. Packt Publishing, 2022. Tilgjengelig: https://books.google.no/books?id=ykqkEAAAQBAJ.

[48] C. Pitt, *React Components*. Packt Publishing, 2016. Tilgjengelig: https://books.google.no/books?id=_97JDAAAQBAJ.

[49] D. Afonso, *State Management with React Query: Improve developer and user experience by mastering server state in React*. Packt Publishing, 2023. Tilgjengelig: https://books.google.no/books?id=rgu6EAAAQBAJ.

[50] Meta-Platforms, 'React - components and props', Tilgjengelig: https://legacy.reactjs.org/docs/components-and-props.html (Lasted ned: 05/04/2024).

[51] S. Saha, 'The react ecosystem', 2024. Tilgjengelig: https://medium.com/@mm.saha1982/the-react-ecosystem-9f0bb6f1d86c (Lasted ned: 05/04/2024).

[52] E. Elrom, *React and Libraries: Your Complete Guide to the React Ecosystem*. Apress, 2021. Tilgjengelig: https://books.google.no/books?id=5Ur3zQEACAAJ.

[53] K. Huang og P. Jumde, *Learn Kubernetes Security : Securely Orchestrate, Scale, and Manage Your Microservices in Kubernetes Deployments.* Packt Publishing, 2020, s. 3–16. Tilgjengelig: https://books.google.no/books?id=fmDwDwAAQBAJ.

[54] TheLinuxFoundation, 'Kobernetes documentation - kubernetes components', Tilgjengelig: https://kubernetes.io/docs/concepts/overview/components/ (Lasted ned: 05/04/2024).

[55] M. Gill, 'What is kubernetes and how does it relate to docker', 2019. Tilgjengelig: https://blog.payara.fish/what-is-kubernetes?utm_term=&utm_campaign=Payara+General&utm_source=adwords&utm_medium=ppc&hsa_acc=2033025017&hsa_cam=15180701657&hsa_grp=130696598578&hsa_ad=559336269836&hsa_src=g&hsa_tgt=dsa-1995938892 0&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjwwr6wBhBcEiwAfMEQsPryFImvFkEXPXVx50iIylQM7TF75fzzHnZNhoCIfkQAvD_BwE (Lasted ned: 05/04/2024).

[56] HelmCommunity, 'Helm - the package manager for kubernetes', Tilgjengelig: https://helm.sh/ (Lasted ned: 05/04/2024).

[57] HelmCommunity, 'Helm docs - charts', Tilgjengelig: https://helm.sh/docs/topics/charts/ (Lasted ned: 05/04/2024).

[58] V. Farcic, *The DevOps 2.5 Toolkit: Monitoring, Logging, and Auto-Scaling Kubernetes: Making Resilient, Self-Adaptive, And Autonomous Kubernetes Clusters*. Packt Publishing, 2019. Tilgjengelig: https://books.google.no/books?id=sPTADwAAQBAJ.

[59] Zabbix-LLC, 'User guide to setting up zabbix monitoring of the kubernetes', Tilgjengelig: https://www.zabbix.com/integrations/kubernetes (Lasted ned: 05/04/2024).

[60] M. DeForest. 'Zabbix meetup online, january 2023: Monitoring kubernetes with zabbix', Youtube. (2023), Tilgjengelig: https://www.youtube.com/watch?v=2dPFvJrK9Mw.

[61] Zabbix-LLC, 'Zabbix helm chart', Tilgjengelig: https://git.zabbix.com/projects/ZT/repos/kubernetes-helm/browse?at=refs%2Fheads%2Frelease%2F6.4 (Lasted ned: 05/04/2024).

[62] InfluxData, 'Influxdata - kubernetes monitoring', Tilgjengelig: https://www.influxdata.com/solutions/kubernetes-monitoring-solution/ (Lasted ned: 05/04/2024).

[63] TheLinuxFoundation, 'Kubernetes - sidecar containers', Tilgjengelig: https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/ (Lasted ned: 09/05/2024).

[64] gunnaraasen, 'Kube-influxdb - monitor kubernetes with the tick stack', Tilgjengelig: https://github.com/influxdata/kube-influxdb (Lasted ned: 05/04/2024).

[65] HoneyPot. 'Prometheus: The documentary', Youtube. (2022), Tilgjengelig: https://www.youtube.com/watch?v=rT4fJNbfe14.

[66] Prometheus. 'Overview: Prometheus'. (), Tilgjengelig: https://prometheus.io/docs/introduction/overview/.

[67] S. Kapare, 'Setup prometheus monitoring on kubernetes using grafana', 2023. Tilgjengelig: https://medium.com/@sushantkapare1717/setup-prometheus-monitoring-on-kubernetes-using-grafana-fe09cb3656f7 (Lasted ned: 07/04/2024).

[68] PyhtonSoftwareFoundation, 'Pypi - the python package index', Tilgjengelig: https://pypi.org/ (Lasted ned: 13/05/2024).

[69] M. Grinberg, *Flask Web Development*. O'Reilly Media, 2018. Tilgjengelig: https://books.google.no/books?id=cVlPDwAAQBAJ.

[70] M. Heydt, *Learning pandas*. Packt Publishing, 2017. Tilgjengelig: https://books.google.no/books?id=Sng5DwAAQBAJ.

[71] I. Solheim og K. Stølen, 'Teknologiforskning – hva er det?', *SINTEF Rapport A160, 22 p*, 2007.

[72] L. M. Jill Jesson og F. M. Lacey, *Doing your literature review: traditional and systematic techniques*. London Sage, 2011.

[73] N. Undervisning. 'Litteraturstudie som metode'. YouTube video. (Dec. 2018), Tilgjengelig: https://www.youtube.com/watch?v=KF3PtpaDsm8 (Lasted ned: 08/02/2024).

[74] E. Skomakerstuen, Personlig kommunikasjon, 14. Mars 2024.

[75] A. Lontons, 'Deploying and configuring zabbix 5.4 in a multi-tenant environment', *Zabbix Blog*, 2021. Tilgjengelig: https://blog.zabbix.com/deploying-and-configuring-zabbix-5-4-in-a-multi-tenant-environment/15109/ (Lasted ned: 14/05/2024).

[76] M. Mathur, 'Grafana: Powerful metrics analytics and visualization', 2023. Tilgjengelig: https://muditmathur121.medium.com/grafana-powerful-metrics-analytics-and-visualization-276457150594 (Lasted ned: 03/04/2024).

[77] T. Persen, 'Announcing multi-tenant grafana support for influxdb cloud service', 2016. Tilgjengelig: https://www.influxdata.com/blog/announcing-multi-tenant-grafana-support-for-influxcloud/ (Lasted ned: 03/04/2024).

[78] pydata, 'Pandas documentation - io tools', Tilgjengelig: https://pandas.pydata.org/docs/user_guide/io.html#io-perf (Lasted ned: 09/04/2024).

Vedlegg

Vedlegg A - Zippet kode for løsningen