[Denne siden er blank med hensikt]

Sammendrag

Ordforklaring

Tabell 1

IoT	Internet of Things
Metrics	Informasjon om dataobjekter
Syntax	Reglene for hvordan ord og symboler organiseres
Services	Programprosesser på en enhet
API	Application programming interface
Mock-data	Eksempeldata til bruk for testing
Properties	Egenskaper som beskriver et objekt
Anamalideteksjon	Oppdage mønstre som avviker fra det normale

[Denne siden er blank med hensikt]

Innholdsfortegnelse

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]. Forsvaret er helt avhengig av IKT for vår operative virksomhet. Allikevel belyser Riksrevisjonens undersøkelse av Forsvarets informasjonssystemer 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 derimot ingen standardisert og vanlig løsning for alle virksomheter. 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 allsidig og fleksibel løsning som kan presentere data fra mange kilder.

I dagens komplekse IT-landskap deles gjerne ressursene blant avdelinger innad i en virksomhet. Det betyr at flere avdelinger bruker samme fysiske enhet til å utføre oppgaver. Oversikten over informasjonen om denne enheten blir da begrenset til IT-personell som er ansvarlig for sikkerheten til infrastrukturen. Dette medfører at ansatte i ulike avdelinger får begrenset oversikt over sine ressurser. 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 virskomhetens ressurser kan man oppnå en proaktiv promblemhåndtering ved å tilby overvåkning av ressursstatus gi tilgang til nødvendig konfigurasjonsdata for gruppens ressuser. 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” dashbord-lønsing vil bidra til å adskille datene

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 effektivt samle og presentere data fra mange kilder i en virksomhet med multi-tenant informasjonsinfrastruktur, for å sikre at brukerne kan lese informasjonen som beskriver deres ressurser.

1.2.2 Forskningsspørsmål / problemstilling

- Hvordan håndtere data fra mange kilder?

Forskningsspørsmålet har som mål å kartlegge hvilken påvirkning multi-tenancy har, og hvordan det kompliserer kravet til løsningen. Forskningsspørsmålet vil også ta for seg hvilke muligheter som finnes for å oppnå multi-tenancy.

- Hvilke krav stilles til løsningen?

Forskningsspøsmålet skal belyse hva løsningen skal tilby og hvilken fleksibiltet løsningen trenger.

- Hvilke open-source verktøy med støtte for multi-tenancy egner seg for å presentere data fra diverse 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 vil utforske hvilke verktøy virksomheter benytter seg av idag. Det vil gjøres en vurdering av ulike verktøy og vi vil velge det mest egnede verktøyet som vi vil lage en proof-of-consept løsning til.

1.3 Leserveiledning

[Denne siden er blank med hensikt]

2 Teori

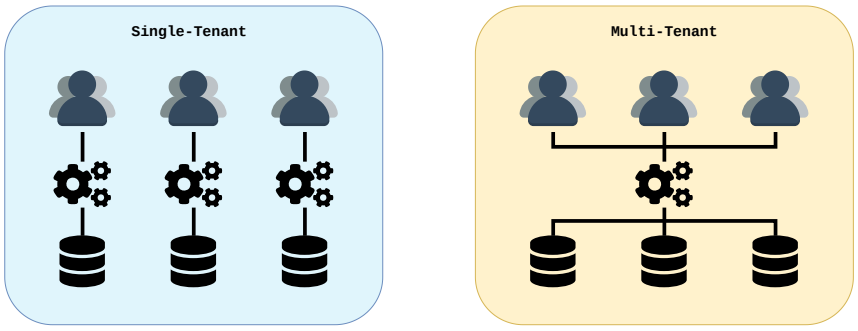
2.1 Multi-Tenant Software as a Service

Multi-tenant software as as service (SaaS) representerer en arkitekturmodell innenfor skytjenester hvor en enkelt programvareapplikasjon og dens underliggende infrastruktur tjener flere kunde 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. I en ”private cloud”, vil behovet for sentralisert infrastruktur og sentraliserte databaser og tjenster sikre 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 datene er logisk isolert på en slik måte at de ikke kan akksesseres av andre tenants. Basert på denne tilnærmingen krever det nøye utformede datamodeller og tilgangskontrollmekanismer for å sikre sikkerhet og privatliv [7][8].



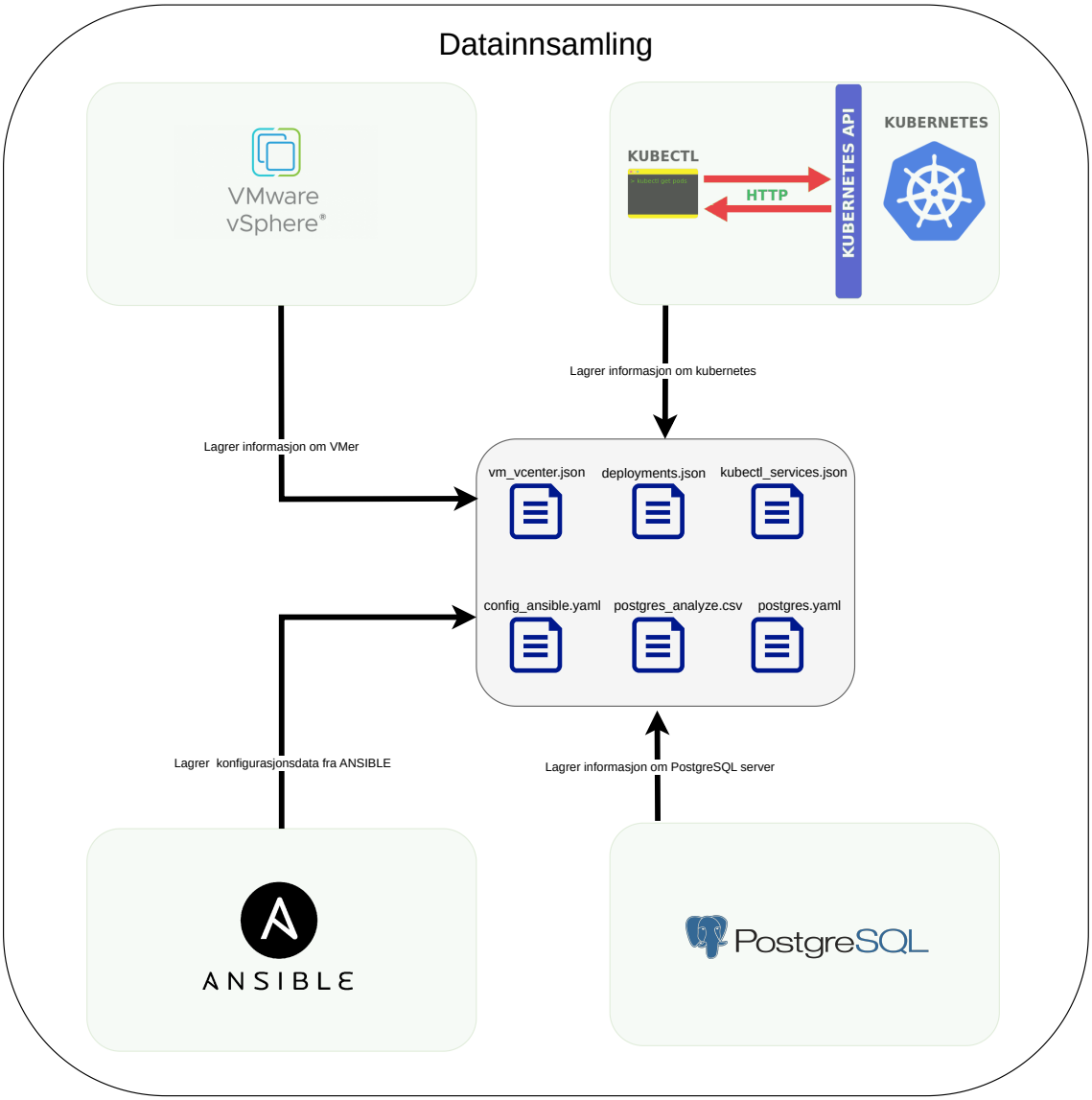
Figur 1: Forskjellen mellom single- og multi-tenant arkitektur

Figur 1 viser prisippet og forskjellen mellom single-tenant og multi-tenant. Som vist, i et multi-tenant miljø, benytter flere brukere samme applikasjon, mens deres data er logisk isolert fra hverandre.

Kompleksiteten i en slik arkitektur vil avhenge av tjensten. 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 Statiske data

I oppgaven benytter vi begrepet statiske data. Vår definisjon av statiske data skal beskrive strukturert eller semistrukturert data som i en lokal fil. Dette kan være data du får når du ber en tjeneste generere en fil om eksempelvis dens konfigurasjon eller tilstand. Begrepet statiske data kan også inkludere tidsseriedata som er data som samles inn i tidsintervaller. Innholdet i filen kan endre seg når du generer en ny fil, men den gamle filen forblir uendret. Denne statiske dataen kan brukes til informasjonsinnhenting om IT infrastrukteren, uten å utvikle en agent for datainnsamling. Figuren under skal illustrere hva vi mener med statiske data og eksempel på hvordan disse statiske dataene hentes.



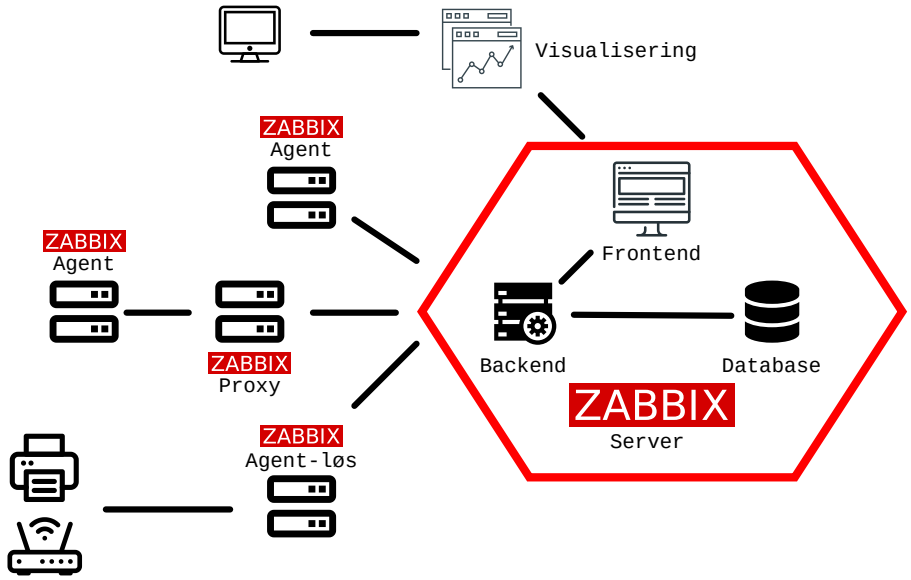
Figur 2: Eksempel på datainnsamling av statiske data

På figuren ser du hvordan vi generer 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 løsning for presentasjon av data. Når man har fått det til å fungere med statiske data iform av en fil kan man hente inn filene med en oppdateringsfrekvens slik at du i prinsippet får live data.

2.3 Zabbix

Zabbix er et "open-source" overvåkingsverktøy for IT-infrasturktur 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 nettverksenhet, servere, klienter, virtuelle maskiner og tjenester. Nøkkelfunskjoner inkluderer datainnsamling, problemoppdagelse, visualisering og varslingssytemer [9][10].



Figur 3: Zabbix arkitektur

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 monitoreringssystem, krever det en kombinasjon av flere nøkkelkomponenter som arbeider sammen. Disse komponentene er essensielle for å samle inn, prosessere, lagre, og presentere data [11]

- **Zabbix Server:** Serveren er hjerte i tjenesten, den har ansvaret for å motta innhentet data, for så å evaluere forholdene, identifisere hvor dataen hører til, generere varsler

og lagre date i databasen. Serevren utførere all logikk som ligger bak for at tjensten skal fungere [12]

- **Database:** Databasen lager 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 [13].
- **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 terskeler for varsling og administrere sytemvarsler. 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 [14].
- **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-monitorering spiller også proxy en viktig rolle. Proxies kan også hjelpe med monitorering av nettverk som er segmentert eller isolert av brannmurer [15].

2.3.2 Datainnsamling

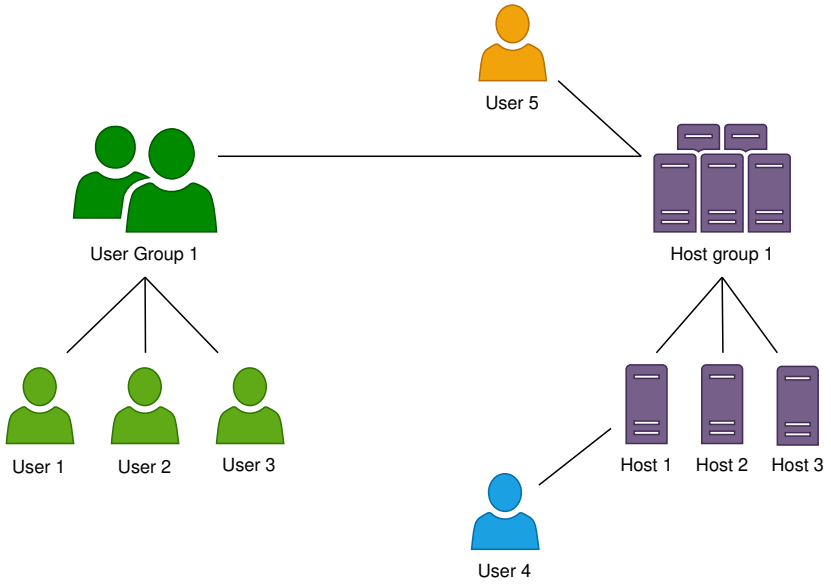
Zabbix tilbyr datainnsamling igjennom en rekke ulike metoder, noe som sikrer god fleksibilitet i ulike driftsmiljøer.

- **Agentbasert overvåkning:** Zabbix agent installert på enheten som skal overvåke. Agenten samler inn detaljert informasjon om systemets operativsystem, programvareappliksjoner og systemressurser [16].
- **Agentløs overvåking:** For enheter der innstallasjon 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 [17].
- **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 hostgroup, som er en samling host, ofte skilt ved enten funksjonalitet, lokasjon eller begge.

Overordnet har også users inndeling i grupper. Users og User-groups kan begge gis tilganger til hosts og host-groups [18].

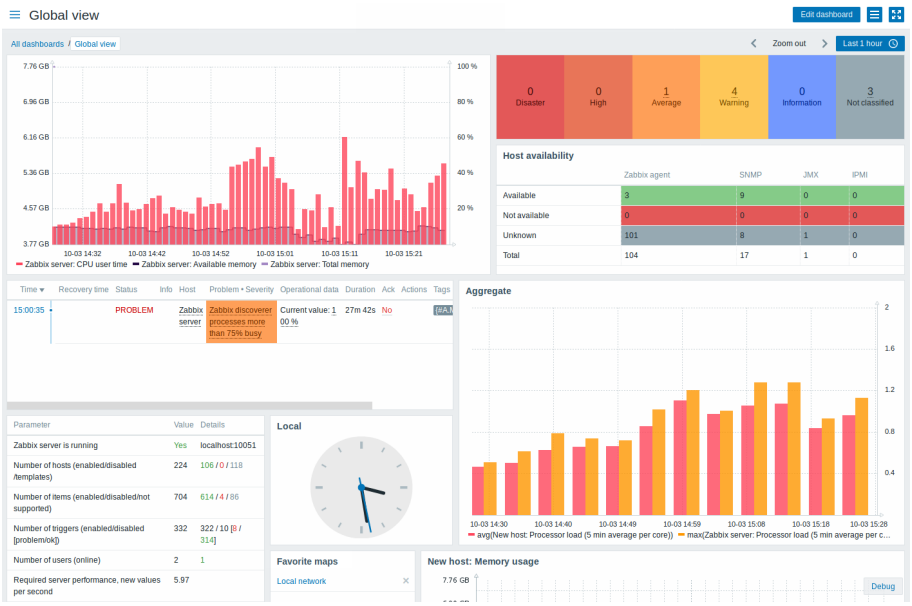


Figur 4: Users og Hosts i Zabbix

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 grensenittet implementert i Zabbix tilbyr en rekke funskjonalitet for visualisering, som vist i figur 5. Basert på ønske for brukeren kan forhåndsdefinerte templates hente informasjon fra databasen og operette grafer og raporter som kan vises i brukerdefinerte dashbord. På denne måten kan brukere og administratorer ha tilgang til sin data og selv definere dashbord slik de selv ønsker [19].



Figur 5: Ekesmpel på dashbord i Zabbix

Dashbord er også tilgangstyrt, og både users og user-groups kan gis tilgang til de ulike dashbordene.

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 [20].

```
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 datainnsamlingstyper i Zabbix, venter trapper item på at data skal bli sendt til den. Dette gjør trapper item spessielt 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 lagres av Zabbix, akkuratt som data samlet inn via den konvensjonelle metoden [21].

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

Kommandoen over benytter Zabbix_sender[22] til å sende informasjon til Zabbix trapper item. Dette er per nå eneste metoden som kan benyttes for å sende informasjon til trapper item.

2.4 InfluxDB

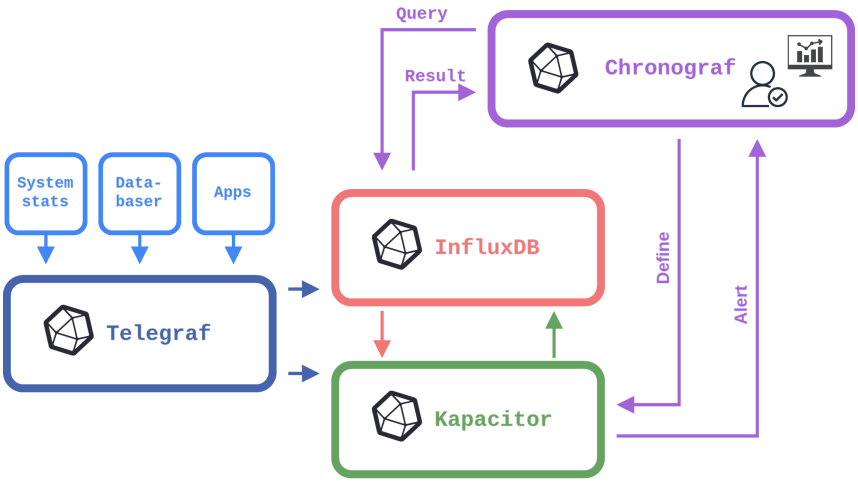
InfluxDB er en open-source tidsdatabasessytem designet for å håndtere tidstemplet data. Et tidsdatabasesystem er optimalisert for høy lese- og skriveytelse [23], noe som gjør den ideell for applikasjoner som involverer store mengder tidsbestemte data. InfluxDB skiller seg ut ved å tilby effektiv lagring 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 [24].

- **Telegraf:** Telgraf 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 [25].
- **InfluxDB:** Tidsseriedatabase for lagring av data.
- **Chronograf:** Chronograf er InfluxData sin løsning på et visualiseringsverktøy og sørger for et grensensitt for brukeren. Tjenesten sørger for oppsett av dashbord, administrasjon og håndtering av alarmer innad i hele stacken [26].

- **Kapacitor:** Kapacitor er hjernen i stacken. Den er hendelsesorientert og sørger for alarmering og anaomalideteksjon. Kapacitor analyserer resultater og sørger for at det ved uventede eller resultater blir informert videre til brukeren [27].



Figur 6: Relasjoner i InfluxDB stacken

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 [28].

2.4.2 Tidsseriedata

Tidsseriedata er sorterte sekvenser av verdier av en variabel målt ved et likt tidsintervall[23]. Et eksempel på tidsseriedata kan være temperatur målt fra en temperaturmåler. La oss si måleren måler temperaturen hvert 10min, da vil resultatet fra disse målingene defineres som tidsseriedata, ettersom det er en verdi som måles med lik antall tid mellom hver måling. På samme måte kan tidsstemplede data fra loggfiler hos endesystemer bli sett på som tidsseriedata. Målingene forteller at verdiene er sortert i en tidlinje, 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 kan bidra til å vide trender og anomalier i målingene. Dersom man f.eks. måler CPU temperatur, kan plutselige økninger i denne temperaturen antyde at et problem har oppstått.

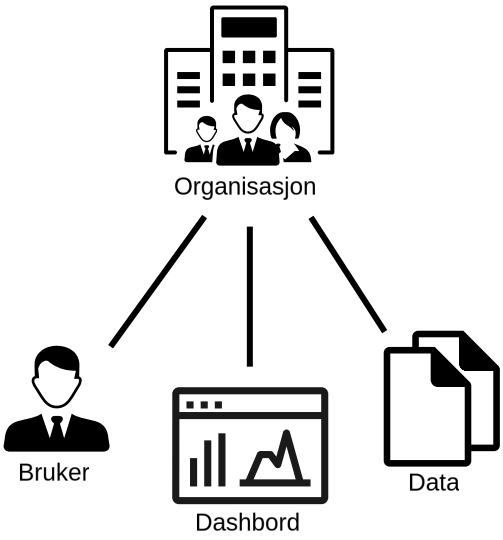
2.4.3 Datainnsamling

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 komplieres til binært slik at den kan kjøre på alle systemer uten ekterne avhengigheter. Ettersom prosjektet er open-source finnes det over 300+ offentlige plugins til tjenesten, og tilbyr overvåking av det meste man kan tenke seg [25].
- **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 [29].
- **Scrapers:** Scrapers er en innhentingsmetode som henter data tilgjengelig via åpne HTTP endepunkter [30].
- **Ecosystem:** Mottar data og hendelser fra tredjepartsapplikasjoner direkte inn i InfluxDB [31].

2.4.4 Brukerhåndtering

I InfluxDB styres brukerhåndtering av organisasjoner. Alle data, brukere, dashbord, oppgaver, hendelser og overvåkede systemer hører til en organisasjon. Utenfor organisasjonen er det ikke noe inndeling av brukere og endesystemer [32].



Figur 7: Tilordninger i InfluxDB

Som vist i figur 7 vil en organisasjon være tilordnet alt som befinner seg i tjenesten. For at en bruker skal ha tilgang til sin data, må han være innmeldt i organisasjonen den dataen tilhører. Brukere kan ikke gis individuelle tilganger, men heller overordnede tilganger til enten alt innad i sin organisasjon, eller ikke være del av den organisasjonen i det hele tatt.

2.4.5 Visualisering

For visualiering og det grafiske grensesnittet benytter InfluxData sin egen Chronograf. Basert på brukerens ønsker kan dashbord tilpasses for å vise sanntidsdata og/eller data lagret influxDB. Chronograf er designet for visualisering av tidsdatabaser og tilbyr et web-basert grensesnitt [26][33].

2.4.6 API

InfluxDB tilbyr et eget API som muliggjør interaksjon direkte med databasen. APIet benytter HTTP-svarkoder, HTTP-autentisering, JSON web tokens(JWT) for autentisering, og svarene kommer i JSON-format. APIet er tilknyttet databasen og dermed består funksjonene av */query*, */ping*, */get*, */write* og */health* [34].

```
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 [35].

2.5 Grafana

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 dashbord som visuelt representerer sanntidsdata fra ulike kilder [36]. 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 åra [37][38].

2.5.1 Grafana Loki

Grafana Labs er utviklerene bak Grafana, og tilbyr en rekke verktøy med ulike bruksområder for brukerene. Grafana er primært et verktøy for visualisering og overvåking, men det finnes også andre verktøyer Grafana Labs har som fokuserer på andre aspekter. Et av disse er Grafana Loki. Grafana Loki er et høyt tilgjengelig loggaggregeringssystem. Loki tilbyr en multi-tenant arkitektur for kostnadseffektiv lagring av logger. I motsetning til andre loggsystemer, lagrer Loki bare etikketer av informarsjonen og ikke all teksten motatt i loggen. Denne metoden å lagre logger på gjør tjenesten mer effektiv, både i form av tid og kostnad. 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 observabilitet [39].

Som nevnt støtter Grafana Loki multi-tenancy. I Loki defineres bruker som tentats og har med det ikke noe øvrige inndelinger. Skille mellom tenatns skjer i spørringene til Loki, der et eget *tenatn id* felt definerer hvilke tenant spørringen hører til. Spørringene og dataen fra tentant A vil ikke være akksesserbart for Tenant B, altså det oppstår et logisk skille mellom de [40].

2.5.2 Datainnsamling

Ettersom hovedfokuset til grafana ligger i visualisering og dashbordløsninger, 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 overvåkingsfunksjonalitet og langtidslagring av tidsseriedata. Når det gjelder loggdata og logger, er Grafana agent konfigurert til å sende disse direkte til Grafana Loki [41].

Grafana kombineres gjerne med Prometheus, et overvåkings- og tidsdatabasessytem [42], 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 [43]. Ettersom grafana er open source, finnes det mange flere metoder og progammer som tilbyr en komplett overvåking og visualisering i kombinasjon med Grafana [37].

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 med populære vektøyene for front-end utvikling. Hovedfunskjonen til react er dens mulighet til å endre deler av siden, enten det er data eller utforming, 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 [44].

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 håndtere data [45].

- **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 [46].
- **JSX (Javascript XML):** JSX er en syntaxutvidelse 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 [45].
- **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 [47]. Props, eller Properties, er en måte å sende data fra en overordnet komponent til en underordnet, og muliggjør gjenbruk av komponenter med ulikt innhold [48][45].

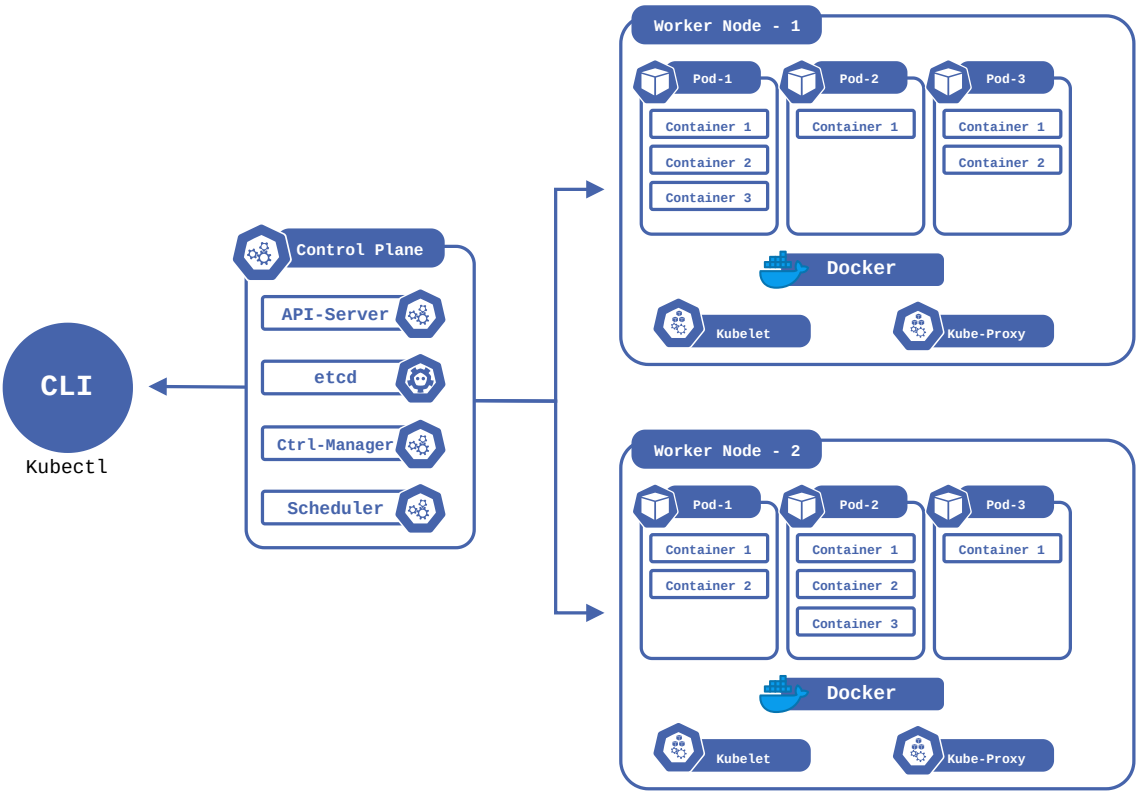
2.6.2 React-økosystemet

På bakgrunn av at React er open-source, har biblikteket fostret et rikt og variert økosystem av tillegsverktøy, biblioteker og praksiser. Dette økosystemet utvider funksjonaliteten til det grunnleggende bibiliteket ved å implementere ny funksjonalitet som tilstadsforvaltning, ruting, API og datahenting, testing, ulike utviklerverktøy og forhåndskonfigurerte 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 [49] [50].

2.7 Kubernetes

Kubernetes er en ”open-source” orkistreringsplattform for konteinere. Plattformen er utviklet av Google og muliggjør automatisert distribusjon, skalering og administrasjon av konteiniserte 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 [51].

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 [51].



Figur 8: Kubernetes Arkitektur

Figur 8 viser et eksempel på et Kubernetes cluster, Control-plane, bestående API-server, etcd (databasen Kubernetes benytter til å lagre clusterdata), Scheduler og Kube controll-manager. Hver av disse rollene bidrar til orkestreringen av både noder og poder. Som vist kan en node inneholde flere Poder, der hver pod inneholder en eller flere containere [52]. I bunn av hver node benyttes gjerne Docker for virtualisering av containerene [53].

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 applikasjonenen. Prosjektet er open-source og driftes av Helm Community etter sin uteksaminering? fra Cloud Native Computer Foundation(CNCF) i 2020[54][55].

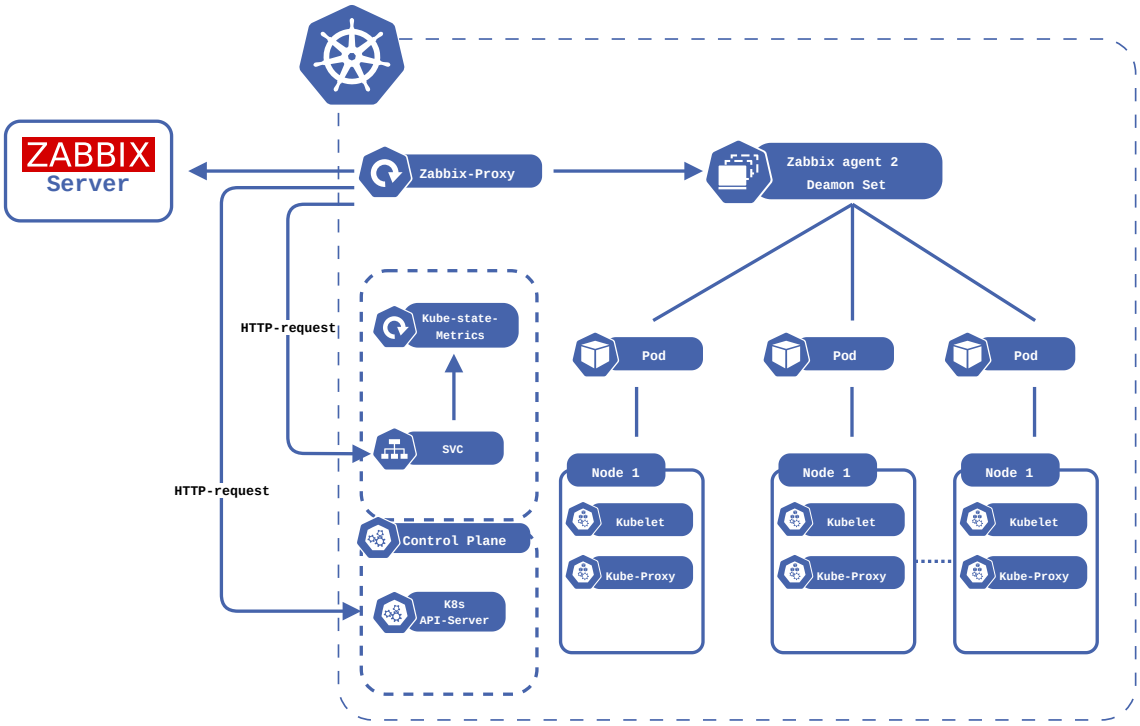
2.7.2 Overvåking

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 på potensielle problemer. Hva som skal monitorers avhenger av clusterets omfang, behov og intensjon [56][51].

- **clusterressurser:** Overvåking av CPU, lagring, minne og nettverksressurser for både master og worker-noder gir informasjon om klustrets genrelle helse og ressursbruk/ytelse.
- **Podder og containere:** Monitorering av individuelle podder og containere hjelper driftspersonell å indentifisere oppdukkende problemer i applikasjoner.
- **Tjenester og ingress:** Overvåking over nettverkstrafikk, forespørselslatens, feilrater og nettverkbruk 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 forhånds-installert i hver node, til å hente metrics om nodens helse og resursbruk. I tillegg til cAdvisor, utnytter Zabbix Kube-State-Metrics som er en sideliggende-applikasjon som lytter til Kube-API og genererer metrics om objektstatuser [57].



Figur 9: Zabbix Kubernetes overvåking

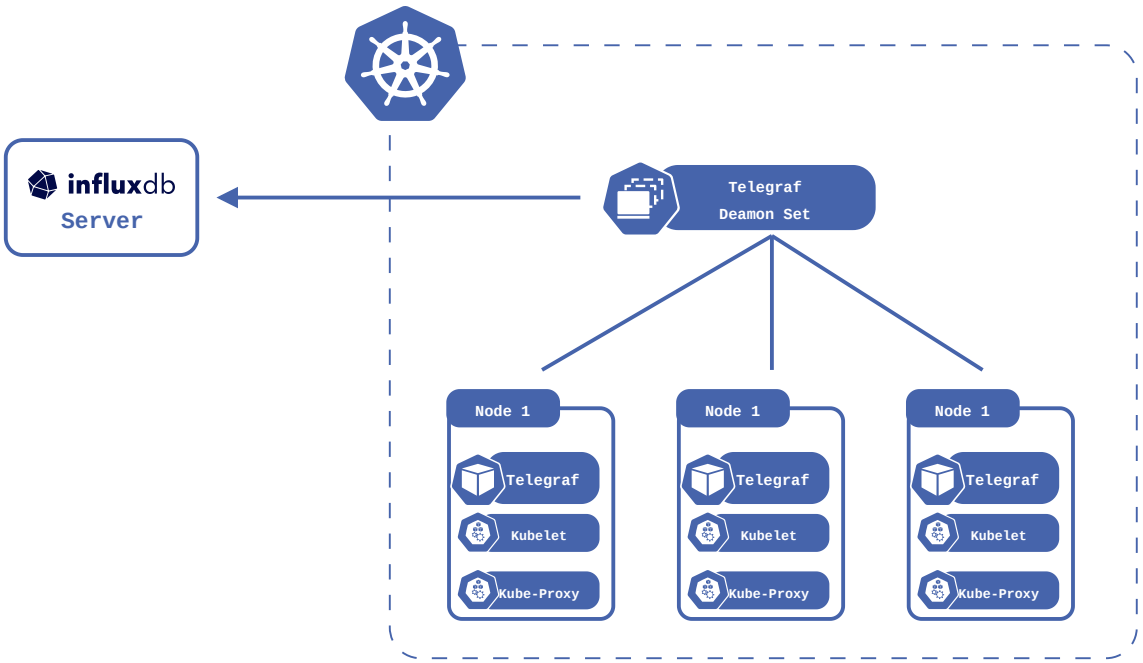
Som 9 viser overordnet hvordan overvåking av Kubernetes clustere gjennomføres i Zabbix. Som figuren viser vil en Zabbix-proxy i clusteret hente eller motta informasjon fra agentene i poddene. I tillegg til dette vil proxyen motta metrics fra Kube-State-Metrics og Kubernetes API-server i Kontrollplanet via HTTP. Underliggende vil en servicekonto være nødvendig for at alle rollene i overvåkngsmekanismen skal kommunisere [58].

Selv om arkitekturen for overvåkningen kan vikre intrikat og komplisert, finnes det løsninger som simplifiserer prosessen. Ved å benytte Zabbix-helm-chart[59] vil man ved en kommando kunne deployere agenter, proxy og alt nødvendig for å fullstendig overvåke clusteret. Innad i Zabbix, finnes det templates for overvåkingen som muliggjør enkel og automatisert oppsett av clustermonitoreringen.

2.7.2.2 InfluxDB

Kubernetes overvåking i InfluxDB kan gjøres på forskjellige måter ut ifra hva som er ønskelig å overvåke og omfanget av overvåkingen [60].

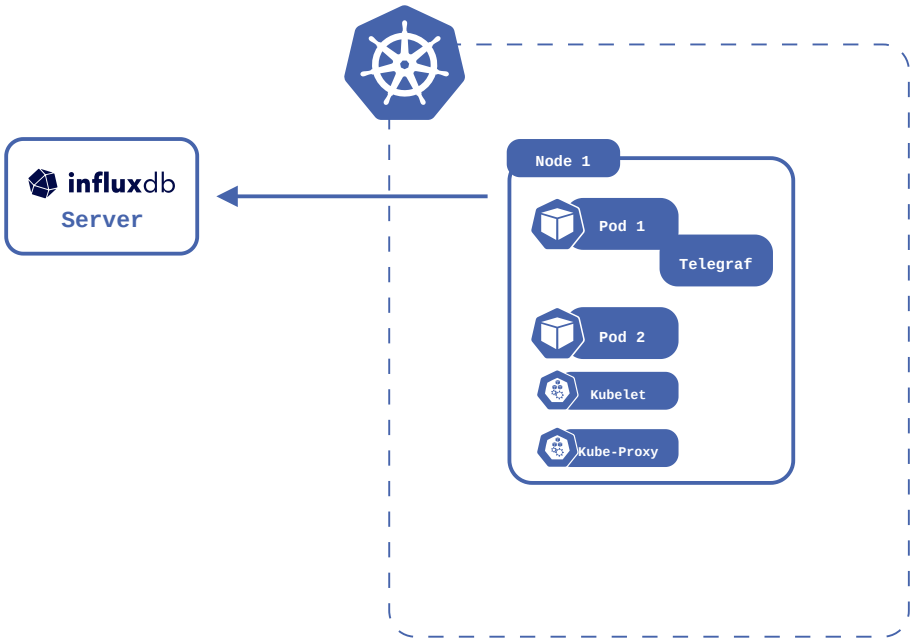
Telegraf deployert i hver node: Som metoden benyttet i Zabbix kan agenter deployeres i hver Kubernetes node. På denne måten kan Telegraf hente data fra noden, podder i noden, containere og applikasjoner via push-pull mekanismer [60].



Figur 10: Telegraf agent deployert i hver node

Som figur 10 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 monitorering av verdier innad i hver pod og node.

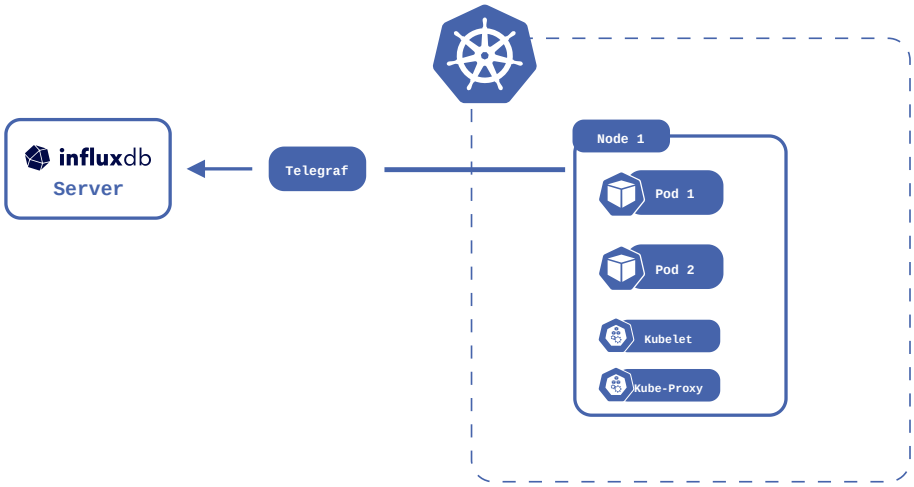
Telegraf deployert som sidebil: Ved instanser der det er ønskelig å monitorere poder, ved å isolere omfanget av overvåkingen kan man deployere Telegraf som en sidebil til poden [60].



Figur 11: Telegraf deployert som sidebil

Figur 11 viser at Telegraf vil være sidestilt med poden, og dermed befinne seg på samme nettverk. På denne måten kan Telegraf hente informasjon om tilstanden og brukeren har mer kontroll over hvilke metrics som er ønskelig å overvåke [60].

Telegraf deployert som sentral agent: Telegraf støtter tjenesteoppdagelse i form av åpne endepunkter for Kubernetes metrics innad i et cluster. På denne måten vil Telegraf søke igjennom podden etter de åpne endepunktene, hente dataen og sende videre til InfluxDB.



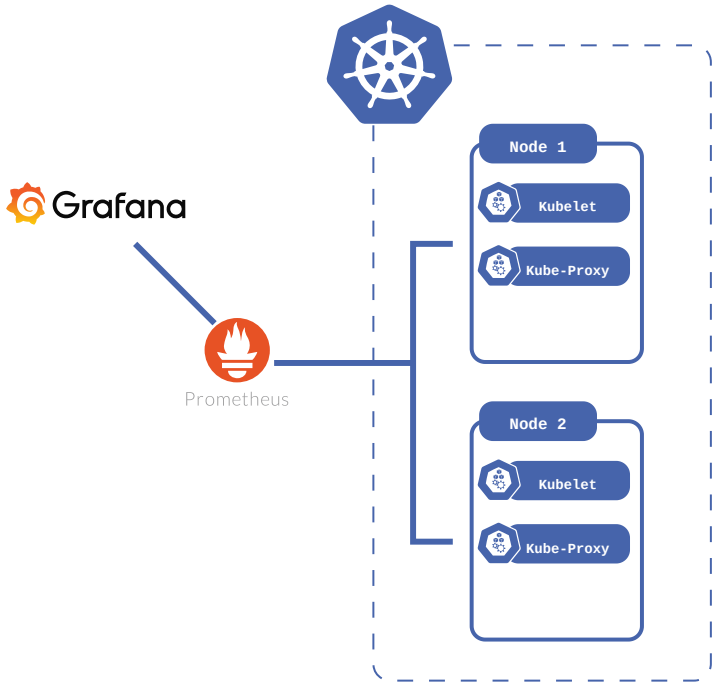
Figur 12: Telegraf deployert som sentral agent

Som vist i figur 12 vil Telegraf her være utenfor clusteret, og heller lytte på endepunkter på innsiden. Ved å benytte denne moetoden vil man kunne monitorer alle applikasjoner med et åpent */metrics* endepunkt. Telegraf skraper da dette 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 noe særlig konfigurasjon innad i Kubernetes, utenom åpning av */metrics* endepunktet for de applikasjonene man ønsker å overvåke [61].

2.7.2.3 Grafana

For Grafana blir det ikke særlig benyttet Kubernetes monitorering 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 kontainermiljøer som Kubernetes [62]. Prometheus samler inn metrics via de åpne endepunktene i applikasjonene, for deretter å lagre disse og tilby dataene til Grafana [63][43].



Figur 13: Utformingen til Kubernetes overvåkning i Grafana

Figur 13 viser hvordan Grafana har minimal tilgang til innhentingen 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 [64].

2.8 Python-pakker

Python pakker er moduler eller pakker som utvider og forsterker funksjonaliteten til kodespråket. Pakkene er forhåndsskrevet kode som gjenbrukes i alle applikasjoner. Disse pakkene tilbyr en rekke verktøy og funksjoner som kan benyttes til løse ulike oppgaver

fra maskinlæring, webutvikling, dataanalyse og systemadministrasjon. PyPI, som er databasen for publiserte python-pakker, inneholder over 5 millioner ulike verktøy, fordelt over 10 millioner filer, og muligjør enkel installasjon og implementering av pakker via 3.parts pakker/programvare.

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 sitt behov. [65]

2.8.2 Pandas

Pandas, Panel datas, er et Python-bibliotek som tilbyr høy-ytelses datamanipulasjon og analyse. Måten Pandas er strukturert er ved å benytte kraftfulle data-strukturer tilby lasting, forberedelse, manipulering, modellering og analysering data på en rekke ulike filformater.

Pandas ble initialt utviklet med finansmarkedet i bakhodet, da dens funksjonalitet i hovedsak omhandlet tidsseriedata og prosessering av aksjepriser og informasjon. For å kunne gjennomføre dette krevde Pandas en rekke funksjonaliteter som strekker seg forbi finansmarkedet, og kan dermed brukes til mer generiske områder som konsekvens av det. På bakgrunn av denne rike funksjonaliteten kan pakken benyttes til en rekke områder, f.eks transformere filen fra en filtype til en annen [66].

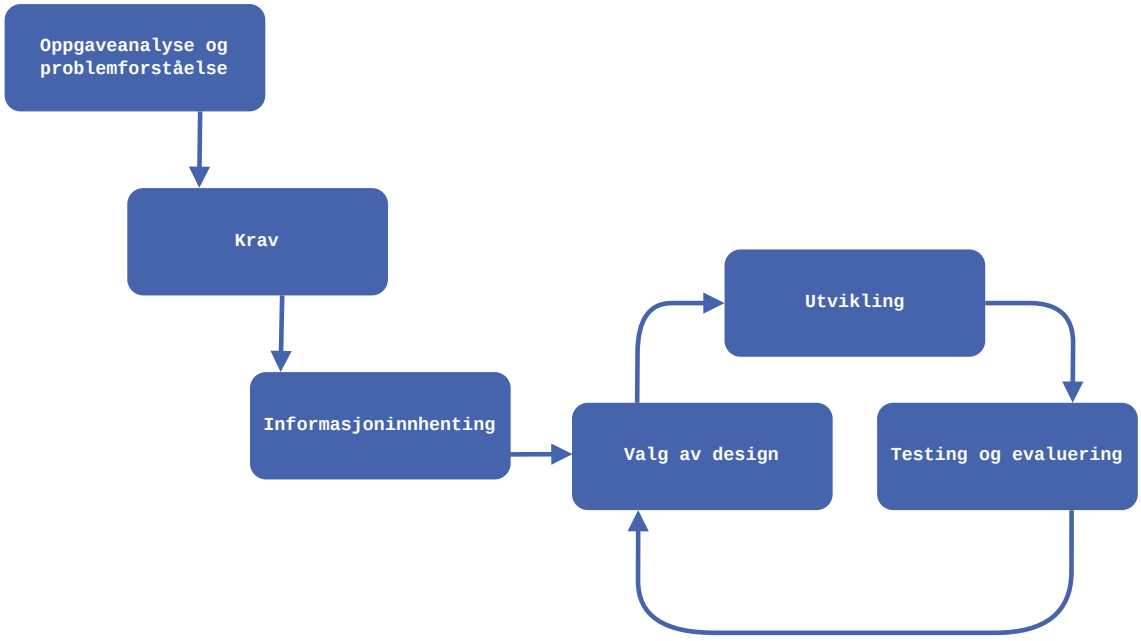
3 Metode

3.1 Metodevalg

Til vår oppgave baserer vi store deler av hoveddesignet på metoder innenfor teknologiforskning. Solheim og Stølen definerer teknologiforskning som ”forskning for å frambringe nye og forbedrede artefakter”[67]. Med artefakter menes menneskeskapte objekter. For vår oppgave der vi skal lage en proof-of-consept løsning kan vi bruke 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[67]. I vårt metodevalg har vi tatt utgangspunkt i stegene i en teknologiforskningsmetode, som er et iterativ prosess. Solheim og Stølen beskriver hovedstegene i denne prosessen [67]:

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 tilfredstiller det potensielle behovet. Den overordnede hypotesen er at artefaktet tilfredstiller 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 tilfredstiller behovet.

Stegene i denne prosessen har som mål å enten styrke eller svekke hypotesene som tilhører artefaktet. Hypotesene til denne oppgaven samsvarer med kravene som er presentert i resultatkapittelet. Vår overordnede hypotese er: *Løsningen tilfredsstiller behovet*. Negative testresultater i som ikke tilfredstiller kravene/hypotesene stimulerer til nye iterasjoner i hovedstegene; problemanalyse - innovasjon - etterprøving [67]. Vi har valgt en metode som tar utgangspunkt i denne iterative utviklingsprosessen. Vi har delt hoveddesignet inn i seks faser. Metoden er illustert i figur 14.



Figur 14: 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 2 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 benyttet i den iterative utviklingsprosessen, der hvert av kravene ble utforsket, utviklet i henhold til oppdragsgivers behov, samt perspektiver

fra respondenter ved intervju, for deretter å bli testet og evaluert. På bakgrunn av kravene ble også informasjonsinnhentingen gjennomført på en mer systematisk måte, da kravene definerte hvilke komponenter løsningen skulle inneholde. Kravanalysen starter ofte med å utforme prosjektbehov ut ifra organisasjonsbehov. I dialog med veileder kunne krav endres eller fjernes basert på gjennomførbarhet underveis.

Kravarbeidet utgir seg for det meste på å identifisere og analysere kravene. 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 sammenlikne og drøfte litteraturen basert på et målbevisst utvalg [68][69]. 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 1 og 3. Det var også hensiktsmessig å samle infromasjon 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. Det var sentralt for planlegging av en løsning som oppfylte behovene oppdragsgiver hadde.

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 informasjoninnhentingen 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 medfølger at testing skjer underveis i utviklingsprosessen. Basert på dette ble det gjennomført en rekke mindre tester for å sikre at konseptet ikke avviker fra opprinnelig design. Fasen startet med å sikre funskjonalitetskrav for løsningen å sjekke disse opp mot det valgte programvareverktøyet. Ved å første 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 føre 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 potensielle 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, og som ble konkludert tidsmessig rettferdig å implementere.

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 software og API fungerer og hvilke kapabiliteter de har. Dette var helt nødvendig for å få en fortsåelse for hvordan løsningen skulle bli utformet. I den iterative modellen testes det underveis i utviklingsprosessen. Dette medførte en rekke mindre tester for å sikre at konseptet ikke avviker fra opprinnelig design. Fasen startet med å sikre funskjonalitetskrav for løsningen å 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.

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, YAML og CSV filer

Løsningen skal ikke basere seg utelukkende på overvåking, men heller på muligheter rundt presentasjon av vilkårlige datafiler med skille mellom tenants. Hensikten med dette er å kunne tilby overvåkning og presentasjon av data til brukergrupper eller kunder av en bedrift, ved å kun ha en instans av programvaren. Dette skal gjøre det lettere for brukergrupper å få tilgang til informasjon rundt sine deler av driftsmiljøet. Løsningen skal kunne hente rådata og strukturere og visualisere dette i et dashbord. Denne rådataen er kommer gjerne i ulik struktur og språk, det er derfor nødvendig at løsningen skal kunne støtte JSON, YAML og CSV filer. Løsningen skal automatisere alt som er mulig å automatisere for at det skal være enkelt å implementere for en virksomhet. I dagens IT-landskap er det blitt vanlig å bruke Kubernetes, det er derfor interresant å sørge for at løsningen skal støtte dette.

4.2 Intervju

I dette kapittelet legger vi fram resultater fra et kvalitativt intervju, samtaler med en professor ved Cyberingeiørskolen og samtaler med veileder. Respondenten har over 10 år erfaring erfaring innen leveranse og drift av IT tjenester. Med intervju og samtaler ønsket vi å finne ut hvilke mulige software løsninger vi kan bruke for å løse problemet. Vi ønsket også å finne ut om hvordan virksomheter idag får en tilfredsstillende oversikt over driftsmiljøets tilstand og konfigurasjon. Det var ønskelig å få perspektiver 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 gjør 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 brukerene kan se det. Zabbix har et dashbord 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 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 hovedsaklig 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, får man en samlet plattform for eksempelvis å sette rettigheter for andre applikasjoner som forenkler 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.

4.2.3 Hvordan ville en virksomheten løst problemet idag

Dette kapittelet tar for seg intervju-svar på spørsmålet *hvilke verktøy og løsninger bruker virksomheter for å holde oversikt over driftsmiljøet idag?*. Det kommer fram at det ikke

finnes en standardisert løsning og 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. HaloPSA er også en løsning man ser oss 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 nettverksrealsjoner, men kan også brukes til VMer og deres realsjoner. Løsningen har mulighet for automatisering. Respondenten har gode erfaringer med Netbox og forteller at det brukes mye av virskomheter idag. 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 som er god for å håndtere feil, samt gi gode visualiseringer. Zabbix er mye brukt for overvåkning. Respondenten ser på Zabbix som et av de beste verktøyene for overvåkning og kontroll over driftsmiljø. Zabbix holder 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

Implementasjon for støtte av flere filformater, og dermed flere datakilder, ble utforsket i form av litteraturstudie. Vi ressonerte oss fram til at det ville være lønnsomt å transformere all data til samme format for å danne et felles standpunkt. Dette gjør koden mindre kompleks. Ved å undersøkte forksjellige tilnærminger til datahåndtering. Vi ønsket et verktøy som kunne implementeres i koden, for å unngå ekstern datahåndtering og dermed minske kompleksitet. Vi fant et python bibilotek som heter Pandas som muliggjorde enkel manipulasjon og tranformasjon av ulike dataformater. Pandas tilbydde muligheten til å omforme en rekke datafiler til ett format. Basert på litteraturstudie tok vi initativ til å utforske Pandas i en praktisk ramme, der testing av ulike funskjoner viste oss hvordan de kan anvendes i løsningen. Ved å transformere dataene til JSON, for deretter presentere de til brukeren, vil man ved få kodelinjer kunne støtte en rekke flere filtyper. Ved å implementere støtte for disse filtypene, kunne vi validere funnene fra litteraturstudie og samtidig utvikle en mer allsidig og fleksibel løsning.

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 hensiktsmessig 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 funksjonalitet 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.
- **Fremvising 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 [70]. Vi testet softwaren 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 [71]. 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 krafig verktøy og et av de ledende verktøyene innen visualisering av data. Det er også meget fleksibelt og har mange tilpasningsmuligheter [72]. Det viste seg også at Grafana hadde god støtte for multi-tenancy ved bruk av Grafana Loki, et loggsystem del av Grafana porteføljen [39]. Vi testet softwaren 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 tungvindt. Det finnes mye god veiledning på hvordan man kan starte å lære seg Grafana. Etter veiledning fra ulike uformelle Youtube kilder fikk vi fortåelse for hvordan man setter opp dashbord 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 diskusjon med veileder ble det forslått å benytte React, et JavaScript-bibliotek, til å utforme en egen applikasjon for visualisering av dataene. React er utviklet av Meta og benyttes til å utvikle avanserte single-page-applikasjoner [44]. 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 selvvbygde UI-komponenter. Ved litteratursøk på mulit-tenancy innefor 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 forsatt utviklerens ansvar å sikre applikasjonens sikkerhet.

4.4.4 InfluxDB

InfluxDB tilbyr hovedsaklig en tidsseriedatabase optimalisert for rask lagring og analyse av tidsstemplet data. Det viser seg at det er mulig å bruke "Telegraf" agenten til InfluxDB til å sende statisk data til InfluxDB (?) [73]. Ved testing erfarte vi at verktøyet er spesialisert til database som vi lite interresert i til oppgaven, allikavel er den fleksibel slik at den tillater å bli tilpasset 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 rett 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 dashbord til brukerene. Denne funksjonaliteten er desverre ikke open-source og er relativ kostbar [74]. Det er alikkevel 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 og som gir et bedre resultat. For valg av mulige eksisterende verktøy som kan brukes for oppgaven utviklet vi sammenligningsfaktorer som er presentert i tabell 2.

Tabell 2: Sammenligning av ulike open-source verktøy

	Zabbix	Grafana	Influx-DB
Open Source			
Fremvising av statisk data			
Mulighet for brukerhåndtering			
API			
Kubernetes Overvåking			

Som det fremgår av tabell 2, er det tydelig at ikke alle vurderte verktøy inneholder den etterspurte funskjonaliteten 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 egen laget løsning i React, så vi videre på å utvikle et program som kunne automatisere prosessen med å konfigurere og presentere data i enten Zabbix, Grafana eller InfluxDB.

Grafana og InfluxDB krever innstallasjon av ekstra tillegg for å nå det som er ønskelig for oppgaven, noe som innebærer ytteligere kompleksitet. Zabbix er utstyrt med den nødvendige funksjonaliteten fra start, uten behov for ytterligere tilpasninger eller tillegg. Zabbix Trapper Item muliggjør fremvising 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 dermed kommer til kort i denne sammenligningen.

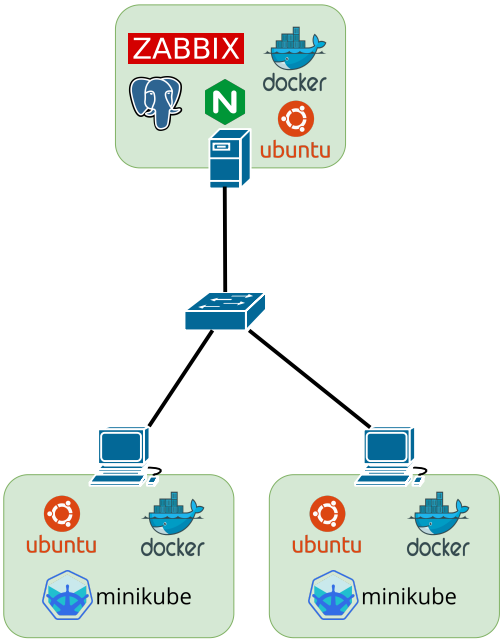
Etter en grundig evaluering av de ulike verktøyene, ble det tydelig at Zabbix var det mest passende valget for vår oppgave. Dette valget ble begrunnet med Zabbix sin allsidighet og funksjonalitet.

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 tenatene 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 15.



Figur 15: Testmiljøet og programvare

Enhetene er koblet sammen med en svitsj som vist i figur 15. 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 kappitel 5.4 om Kubernetes. Vi benyttet oss av Ubuntu Linux på alle enhetene etterom 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 applikasjonen. Som nevnt krever Zabbix en rekke roller for å fungere, og ved å benytte kontainisering via Docker kan man enkelt sørge for at alle rollene lastes ned riktig og fungere 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 standard pakke i Python-biblioteket. For å kunne kommunisere mellom applikasjon og Zabbix krever Zabbix en autorisasjonsbærer på forhånd av alle operasjoner. Dette er et eget API-kall som returnerer bæreren.

```
1 link = "http://192.168.0.93/api_jsonrpc.php"
2
3 data = {
4         "jsonrpc": "2.0",
5         "method": "user.login",
6         "params": {
7             "username": "Admin",
8             "password": "zabbix"
9         },
10        "id": 1
11        }
12
13 resp = requests.post(link, headers = {"content-type": "application/json
    -rpc"}, json=data)
```

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

5.2 Datakilder

For testing og utvikling benyttet vi først og fremst statiske datsett vi fikk fra veileder. Disse filene inneholdt data hentet fra VMware Vsphere, Ansible og Kubectl. Disse statiske datene var hensiktsmessig for å utvikle python script for å transformere dataene og sende til Zabbix. Alternativet var å benytte agenter for deler av datainnsamlingen, som vi besluttet ville komplisere feilsøking og utvikling.

5.2.1 Hvilke datakilder og data

De konkrete filene vi benyttet var:

- Data fra VMware Vsphere som beskriver ulike VMene en server kjører
- 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 mock-data i ulike språk for å skjekke at løsningen støttet det gitte språket og dens struktur.

5.2.2 Krav til dataene som mottas

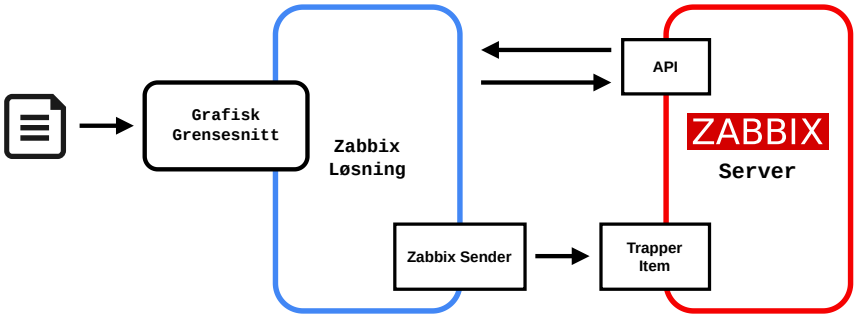
Etter litt testing fant vi ut at dersom koden vår skal støtte mange ulike språk må alle de ulike formatene ha et felles standpunkt. Det er altså nødvendig å sette krav til dataene vi sender til løsningen. Det er nødvendig at det beskrives hvilke tenants de ulike ressursene tilhører i filen. Dette er 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 daschbordet. 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 enklere vise hvordan løsningen vil se ut. Figur 16 under viser hvordan overordnet programmet vil fungerer



Figur 16: Illustrasjon over kommunikasjonen mellom løsningen vår og Zabbix

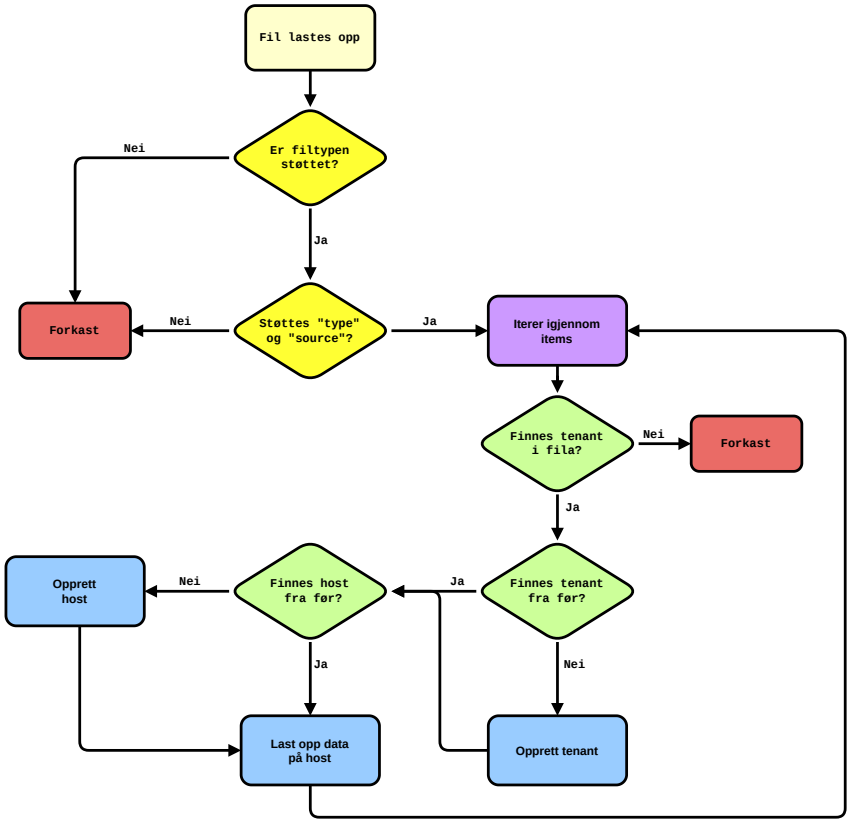
Som figur 16 viser skal man kunne laste opp filene man ønsker til Zabbix via det grafiske grensesnittet. Videre vil et vårt python-program tolke dataene, og hente nødvendig informasjon fra Zabbix for å deretter kunne laste opp datene til riktige tenants med Zabbix trapperitem. Python programmet vil hente 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 tilgangstyring 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. Ved å benytte denne definisjonen tillot det oss å enkelt inkluderer flere kilder til samme tenant, samt opprette egne user-groups tilknyttet hver tenant (host-group) for å gjøre det enkelt å tilegne brukere tilganger. Denne User-groupen gis automatisk tilgang til tenanten og dermed kan man ved å gi users tilgang til user-groupen, også gi de tilgang til all informasjon tilknyttet tenanten.

For administratorer ønsker å laste opp data for presentasjon benyttes vårt grafiske grensesnitt. 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. Dersom filen som lastes opp ikke støttes, eller har feil, vil brukeren få tilbakemelding om dette.

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 17 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 17: 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 dashbord tilegnet tenant. Figur 17 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 senere. Dersom filen som lastes opp innholder flere dataobjekter/hosts, vil applikasjonen iterere igjennom disse en etter en og utføre alle nødvendige endringer på hver av de.

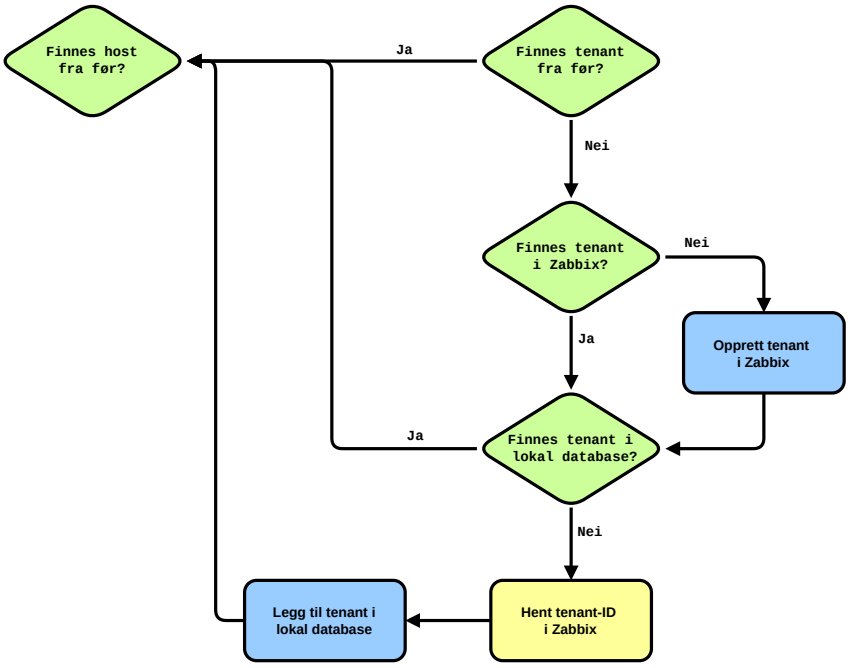
5.3.3 Multi-tenancy i Zabbix

For å opprette tenant i Zabbix benyttet vi Zabbix *Host group*. Dette ga oss muligheten til å ha flere datakilder til samme tenant. 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 oprette 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 {
```

```
    "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 18: Detaljert flytskjema over multi-tenancy aspekter ved koden

Figur 18 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 17

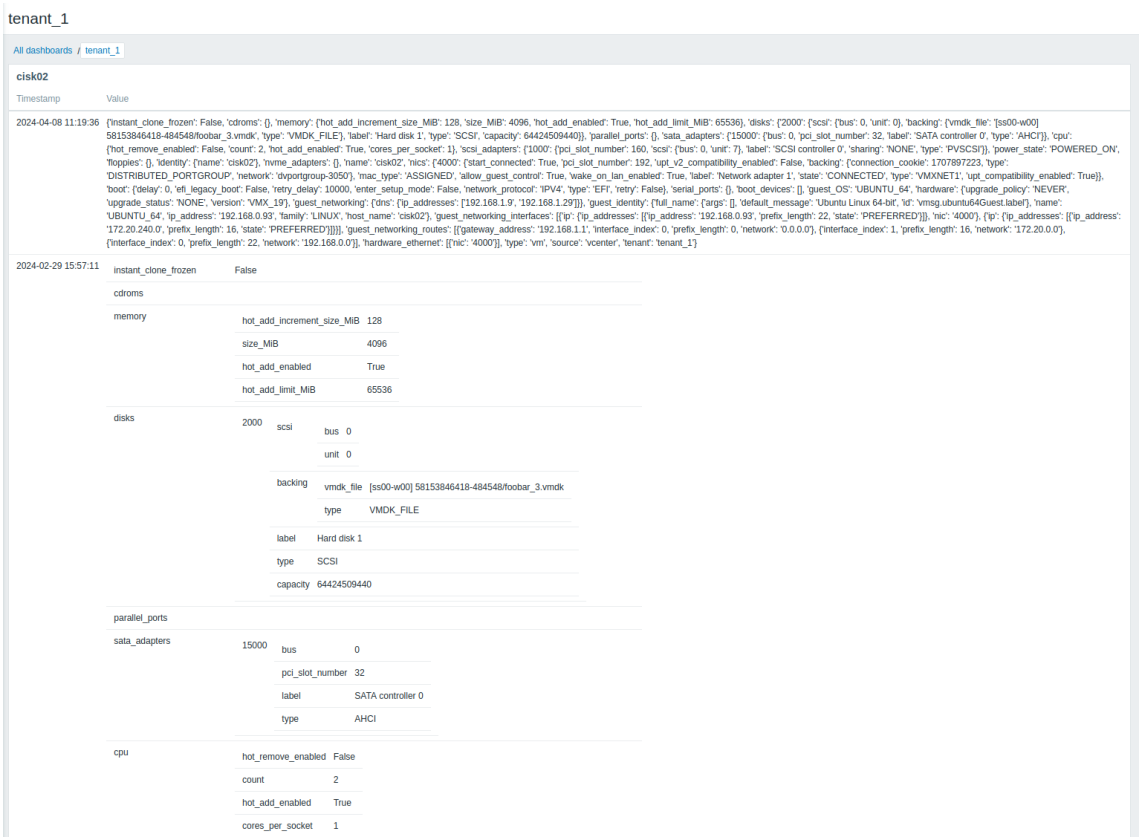
```
def hvilke_hostgroup_finnes():
    data = {
```

```
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 hostgroups. Ved å deretter iterere igjennom hostgroups og lagre navnene i en egen liste, har vi hentet navnet på alle tenants i Zabbix. Dette brukes eksempelvis 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 dashbord for hver tenant, med tilganger kun til den gitte tenant. Hvert dashbord må ha widgets for å gi noen informasjon. Vi ønsket å benytte en widget som var mest mullig 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 språk 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, YAML og CSV 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 19: Presentasjon av data i JSON (øverserte del), og HTML (nedeste del)

Figur 19 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 ved nederste timestamp, fikk vi en mye mer oversiktlig og luftig presentasjon av data. Zabbix APIet støttet oppsett av dashbord, som tillot oss å automatisere prosessen. I koden skjekkes 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 dashbord. Dersom datakilden finnes fra før, oppdateres kilden i det gitte dashbordet 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 **??**. Det betyr at alt den mottar i disse formatene presenteres på tilsvarende struktur som nedeste del av figur 19.

5.4 Kubernetes implementasjon

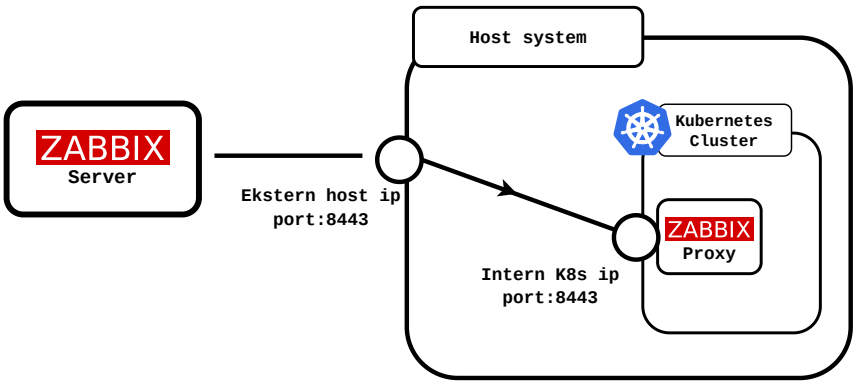
Kubernetes overvåking ble implementert med bruk av Zabbix-Helm-Chart og Kubernetes templates. Løsningen krever noe manuell konfigurasjon da det er nødvendig å installere

Helm-Chartet på clusteret man ønsker å overvåke. Videre skal vår løsning ta inn riktige parametre for å sørge for at overvåkingen settes opp riktig, disse parmeterene innbærer tenant, navn, ip-adresse, proxy navn og en generert autentiserings token. For at ikke det skal oppstå komplikasjoner må proxy navn være orginalt og kan ikke være like. Av denne grunn må man før installasjon endre verdien for proxy navn i Helm-Chartet til noe unikt for hver 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
```

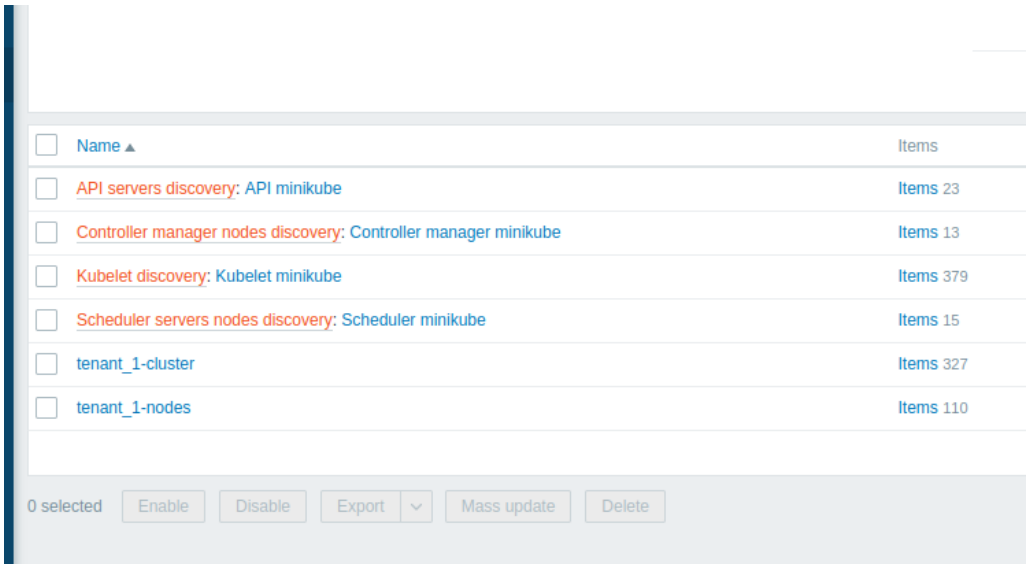
På samme måte som ved implementasjonen for generisk data, skal applikasjonen ta inn informasjonen og automatisk sette opp det nødvendige innad i Zabbix. Løsningen baserer seg på grensesnittet, der et skjema må fylles ut med alt av nødvendig informasjon, før dette omformes til en JSON-fil og sendes til 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, mer nøyaktig, 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 20, 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.



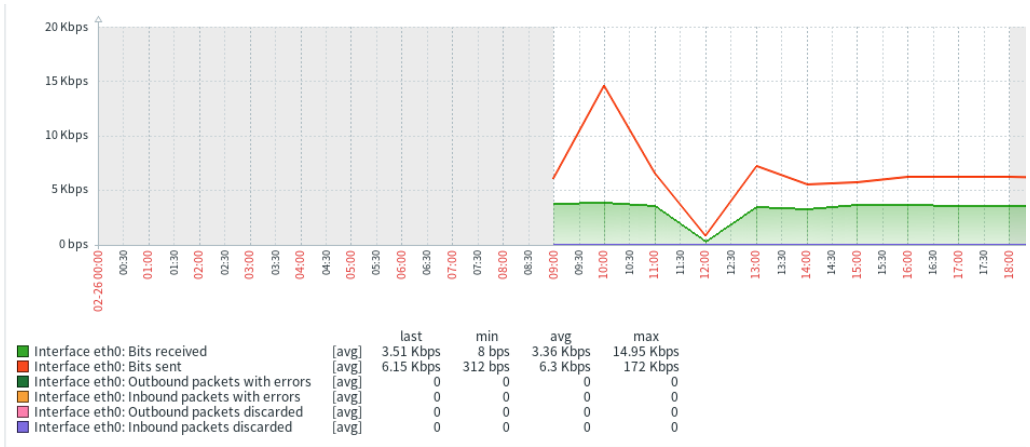
Figur 20: Kommunikasjon mellom Zabbix server og Kubernetes cluster

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 nødvednige for at de forksjellige itemsene skal populeres med riktige verdier, live fra clusteret.



Figur 21: Skjermutklipp fra datakilder i Zabbix frontend

Figur 21 viser at klusteret (tenant_1-cluster) har mottat over 300 metrics i form av items med live verdier for tilstanden til clusteret og alle dens podder. Videre kan man, dersom man har flere clustere til hver tenant, filtrere disse på tilhørende proxy som er unik. På denne måten vil man enkelt kunne få informasjon om ønskelig cluster og monitorere helsetilstanden. Noen eksempler på metrics som blir hentet er CPU-bruk, antall podder, pod-status og status på containere. Til slutt kommer templatet som ble benyttet med predefinerte widgeter for visualisering av ulike metrics. Figur 22 under viser eksempel på en slik widget, der nettverkstrafikk vises i form av en graf.

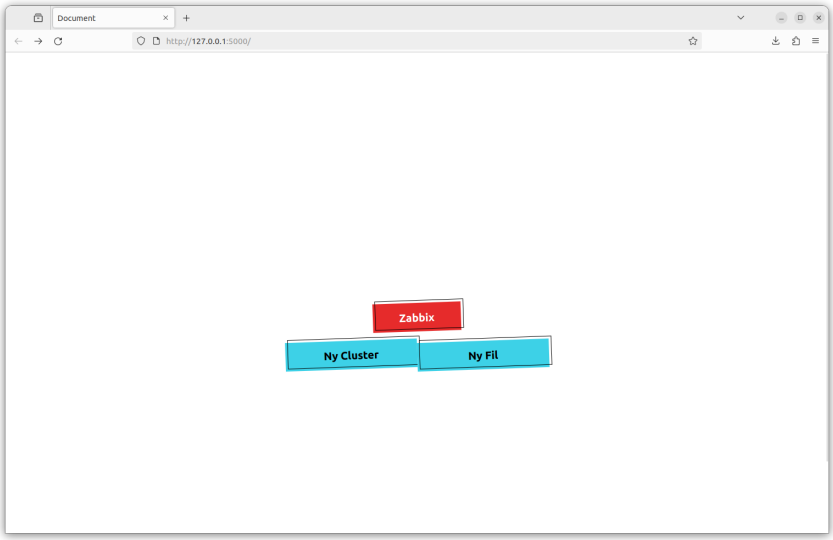


Figur 22: Predefinert widget i Zabbix dashbord

5.5 Grensesnittet

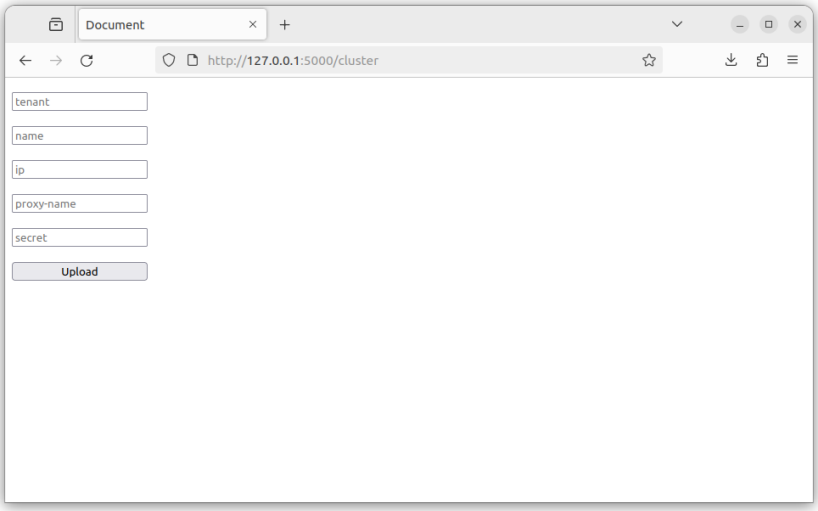
Brukeropplevelesen vil ikke være i hovedfokus for et PoC. Likevel valgte vi å utforme et enkelt brukergrensesnitt for å bevise hvor enkelt applikasjonen kan brukes. For å bygge

grenesesnittet ble det benyttet Flask, som muligjorde raskt og enkelt oppsett av et simpelt webbasert grensesnitt. Grensesnittet tilbyr tre valg, som vist i figuren 23. Zabbix valget sender deg til Zabbix Frontend. De to andre valgene er innlasting av datafiler (Ny Fil), eller oprette Kubernetes cluster monitorering (Ny Cluster).



Figur 23: Egenutviklet webgrensesnitt

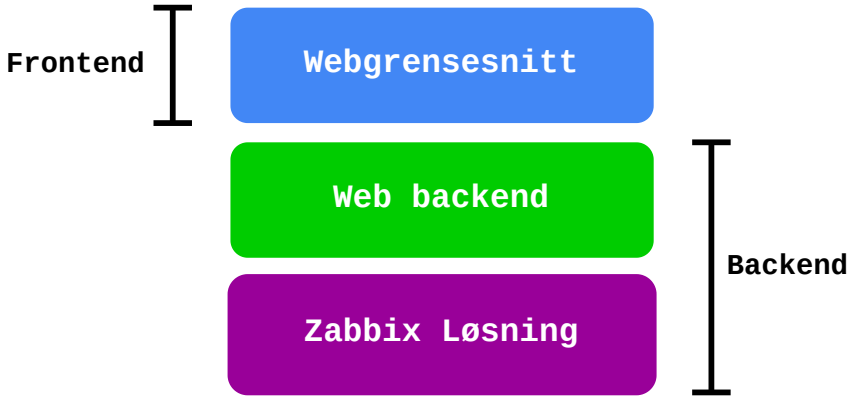
Ved innlasting av datafiler, vil brukeren kunne laste opp datafiler til grensesnittet, før disse sendes til applikasjonen og videre legges inn i Zabbix, med forbehold om kravene er fulgt. Ved Kubernetes monitorering vil brukeren bli presentert med et skjema, vist i figur 24. Når skjemaet er ferdig utfylt og sendes inn, vil grensesnittet omforme svarene til en JSON-fil og sende denne videre til applikasjonen for behandling. På denne måten kan brukeren i teorien utforme denne json-filen selv, for så å sende den inn via innlasting av datafiler.



Figur 24: Skjema i grensesnitt for å legge til nytt cluster

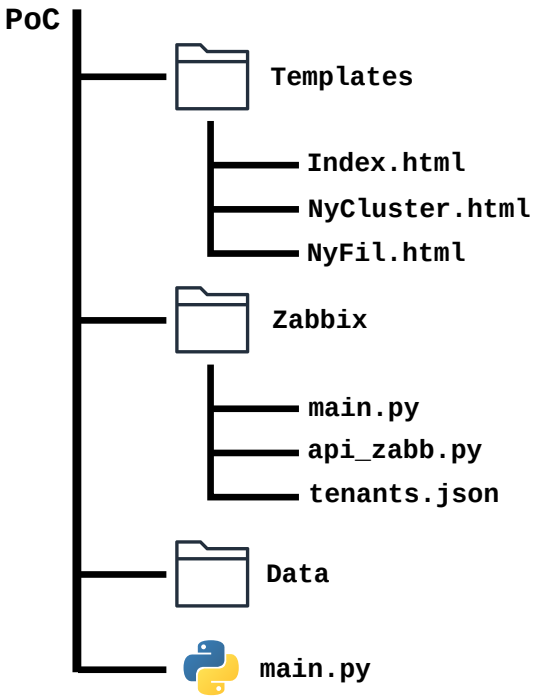
5.6 Utforming av koden

Urformingen av koden og vår løsning kan inndeles i frontend og backend. I vår utforming består backend av to lag, illustrert i figur 25. Web backend håndterer forespørslene fra grensesnittet, som ruting og dataklargjøring, mens 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 25: Stacken i hvordan koden er utformet

I utformingen av koden har vi en filstrukturen til tjenesten som er annerledes inndelt. Overordnet består tjenesten av tre mapper, med sine respektive filer, og et Python-script, som illustrert i figur 26. Som vist i figuren under er *main.py* plassert utenfor mappene. Dette er Web backend som krever at templates mappen befinner seg lateralt 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 26: Filstrukturen i koden

5.6.1 Webgrensesnittet

Webgrensesnittet består 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 or å forbedre brukeropplevelsen. *index.html* er hjemmesiden til applikasjonen og består av de tre elementer, Zabbix, Ny Fil og Ny Cluster. Med CSS benyttes det flexbox for å sentrere elementene på siden, 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 representerer de 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 næremere 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">
```

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

5.6.2 Web Backend

Web backend er i hovedsak bygd opp av main.py filen nederst i figur 26. Rutene er definerte slik at de returnerer de sidene brukeren ønsker. I eksempelet under ser man et eksempel på en slik rute. I denne koden defineres hvilke metoder som godtas og hvilke operasjoner som skal gjennomføres på de forksjellige metodene.

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

Linje 1 i koden over definerer hvilke endepunkt denne ruten tilhører, i dette tilfellet */fil* som vi var innom tidligere. 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 filen/siden *NyFil.html*. Ved *POST* forespørsel vil den hente filen/filene som lastes opp, deretter lagre filen, og sende filbanen til Zabbix løsningen, som vist i linje 8-10. 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. Ved å transformere filene som lastes opp til JSON, legge disse inn i et objekt med allerede nødvendig informasjon, og la brukeren via et grensesnitt velge hvilke tenant filen skal tilhøre, kunne vi sikre både allsidigheten og brukervennligheten til løsningen. Koden under viser et eksempel på hvordan tranformering av filer gjennomføres ved å benytte pandas.

```
1 import pandas as pd
2
3 if fil.filename.split(".")[1] == "csv":
4     pd_fil = pd.read_csv(fil)
5 elif fil.filename.split(".")[1] == "html":
6     pd_fil = pd.read_html(fil)[0]
7
8 pd_fil.to_json(filpath, orient="table")
9
```

```
10 json_object = {
11     "type":"structured",
12     "source":"table",
13     "tenant": request.form["tenant"],
14     "name" : "filnavn",
15     "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 i dette laget representert i figur 17. All logikken i koden skjer i *main.py* under mappen *Zabbix*. For å kunne holde oversikt over alle API-metodene som benyttes ble dette valgt å legge i en egen fil (*api_zabb.py*), der hver metode er representert av en egen funksjon.

```
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, og funksjonen tar inn navnet på host-groupen som skal lages som parameter, for deretter å implementere dette i API-kallet til Zabbix-serveren. Videre leser funksjonen 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, da spesielt funksjoner som krever *host_id*. Vår løsning benytter i hovedsak hostnavn når det refereres til hosts, slik at ved API-kall der *host_id* er nødvendig, henter vi bare dette først via *host.get* metoden, som er en egen funskjon.

Etter at brukeren har lastet opp sin fil gjennomføres det en sjekk for å sikre at filtypen støttes. Filtyper som YAML og JSON kan enkelt lastes som dataelementer i python, slik

at de kan behandles uten tilleggspakker senere, dette er beskrevet i koden under.

```
if filnavn.split(".")[1] == "json":
    data = json.load(f)
elif filnavn.split(".")[1] == "yml" or filnavn.split(".")[1] == "yaml":
    data = yaml.safe_load(f)
else:
    return 0
```

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

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

Koden over beskriver hvordan programmet sjekker for de definerte datafeltene, for deretter henter riktige datafelt videre. Basert på hvilke *source* og *type* vil dataene ha noe forskjellig struktur, så ved å ha definert noen av disse kan man sikre at alle ønskelige data kan forstås. Den mer generiske løsningen er noe likt utformet, da denne bare inneholder generiske datafelt, som *item* istedenfor *vm* 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 riktig. Linje 7 kaller *dashbord_fiks()* med tenant som parameter for å oppdatere dashbordet til tenanten, slik at dataene visualiseres.

For kubernetes-monitorering opprettes to tenants 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.

```
def link_kubernetes_template(hostid, template, secret, ip):
    data = {
            "jsonrpc": "2.0",
            "method": "host.update",
            "params": {
                "hostid": hostid,
                "templates": [
                    {
                         "templateid" : hent_template_id(template)
                    }
                ],
                "macros": [
                    {
                        "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())
```

Som vist krever funskjonen fire som parametre den legger ved API-kallet for at hosten skal settes opp riktig. Macros (linje 12) er verdier til templaten, som må populeres via *host.update* for at de skal tilegnes verdier. Her benyttes informasjon hentet fra filen som sendes til Zabbix løsningen etter at brukeren har sendt inn skjema via frontend.

5.7 Resultat PoC løsning

I dette kapittelet legger vi fram hva vi har fått til med løsningen. Får å gi en intuitiv forklaring av hva som er resultatet av løningen 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 ressuser. 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 programmet vi

har utviklet gjennom interfacet vårt, dette generer automatisk dashbord til tenants, samt widgets med oversikt over dataene dine. Løsningen presenterer data du sender i HTML. Løsningen støtter følgende filformater:

Tabell 3: Oversikt over støttede språk/formater

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 f.eks JSON eller 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 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 er det 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 betrakning at ett intervju gir oss et smalt narrativ. Vi vurderte derfor fler muligheter. Designvalget mellom de ulike mulige løsningene baserte vi på egendefinerte sammeligningsfaktorer. 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 mer 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 hvor enkelt vi klarte å sette opp tjenestens default funskjonalitet. 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 funskjonalitet som gjør løsningen mer fleksibel og tilpasningsdyktig. En faktor som også var relevant i valget var oppdragsgivers ønsker. Valget om løsning var åpent sålenge 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 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 som beskriver objekter ved enheten. I testsammenheng har vi manuelt lastet ned datsett 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 datene. 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 dashbordet. 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 dashbordet. Vi har derfor satt disse tre minimumskravene for at datene skal presenteres riktig i Zabbix. Det kan derimot settes fler 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 presentert. 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 struktur på hvordan formatet i filen ser ut, noe som kan føre til inkompatibilitet dersom det er benyttet avanserte dashbord 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. Fler 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 Leveranse av overvåkning til kunder

For brukeren vil tilgjengelighet og enkelhet ved løsningen være nøkkelpunkter for at løsningen skal benyttes. Kravene til både løsningne og dataene setter grenser for brukeropplevelsen, samtidig som sikkerhet setter begrensinger for tilgjengeligheten. Viktigheten med en balanse mellom disse aspektene er viktig for at man opprettholde en god løsning. I vår løsning er ikke tilgjengelighet og sikkerhet et fokusområde, da vår oppgave baserer seg på å utforme et PoC. Kravene til løsningen er derimot sett nøye på, og det er disse vi i hovedsak har formet løsningen rundt. Et av kravene i kapittel 4.1 sier at løsningen skal være enkel å implementere. Dette kravet omhandler i hovedsak implementasjon fra leverende avdeling, ikke brukeren. Vi kunne med fordel hatt mer fokus på brukeropplevelsen og sikkerhet rundt løsningen under utviklingen, da dette vil være nødvendig ved et ferdig produkt, men med tanke på de gitte tidsrammene nedprioriterte vi dette for å sikre at kravene til løsningen ble oppfylt. Selv om implementeringen av en slik løsning kan være vellykket, er det en tendens til at den blir oversett eller undervurdert etter at den er lansert. Dette skyldes ofte at brukeropplevelsen ikke er tilfredsstillende, og at eksisterende normer og tidligere løsninger fortsatt er dypt forankret i kundens eller brukerens oppfatning.

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. 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 det moderne IT-landskapet, samt deres fleksibilitet og lesbarhet. For å inkludere data fra andre formater, benyttet vi en mekanisme som konverterer disse formatene til enten JSON eller YAML, 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 eller YAML, 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. For presentasjonen av dataene valgte vi HTML-formatet. Dette valget ble gjort med tanke på å maksimere lesbarheten og tilgjengeligheten av informasjonen. Vi valgte å ikke komplisere løsningen med mer avanserte visualiseringer da oppgaven fokuserer på presentasjon av data fra mange klider i multi-tenacy miljø. Vi har allikevel adressert muligheter for bedre presentasjon i kapittel 6.2. Ved å presentere dataene i HTML, kan vi forbedre brukeropplevelsen, gjennom en mer visuell presentasjon enn på dets originale dataformat. Vi bruker Pandas Import/Export (I/O) for datatransformasjon. Pandas I/O har støtte for 19 ulike dataformater. Vi møter begrensninger med at Pandas støtter bare I/O med et begrenset sett av filformater som lar seg overføres til deres tabulære datamodell [75]. 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 implementasjoner. Ettersom Kubernetes er mye brukt i dagens IT verden har vi også inkludert live overvåkning av Kubernetes clustere i løsningen, der hvert cluster betraktes som sin egen isolerte tenant. Løsningen støtter cluster overvåking og presentasjon av tilstand- og konfigurasjonsdata fra server, klient, VM og Kubernetes sålenge dataene er strukturert- eller semistrukturert og finnes på et av de 19 formatene Pandas er kompatible med.

6.5 PoC

7 Konklusjon

videre arbeid: Som videre arbeid til oppgaven vil man behøve en rekke forbedringer før løsningen kan etableres i en virksomhet. På bakgrunn av at løsningen er et PoC er det naturlig at flere funskjoner er utelukket fra løsningen. Til å begynne med bør koden gjøres mer robust for håndtering av datafiler som ikke støtter kravene, samt ved diverse feil i løsningen. -predefinerte widgets -versjonkrav til data - agent - koden mer robust - docker composite

[Denne siden er blank med hensikt]

8 Referanseliste

References

[1] Forsvaret Forsvarssjefen. 'Forsvarssjefens fagmilitære råd 2023'. In: (2023). URL: 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 (visited on 9th Feb. 2024).

[2] Rikrevisjonen. 'Riksrevisjonens undersøkelse av Forsvarets informasjons- systemer for kommunikasjon og informasjonsutveksling i operasjoner'. In: (2023). URL: https://www.riksrevisjonen.no/globalassets/rapporter/NO-2022-2023/forsvarets-informasjonssystemer-ugradert-versjon.pdf (visited on 9th Feb. 2024).

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

[4] Mazin Yousif. 'Cloud-Native Applications—The Journey Continues'. In: *IEEE Cloud Computing* 4.5 (2017), pp. 4–5. DOI: 10.1109/MCC.2017.4250930.

[5] Antonio Rico. 'Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications'. In: *Enterprise Information Systems* 10.4 (2016), pp. 400–421. DOI: 10.1080/17517575.2014.947636. eprint: https://doi.org/10.1080/17517575.2014.947636. URL: https://doi.org/10.1080/17517575.2014.947636.

[6] L. Coyne et al. *IBM Private, Public, and Hybrid Cloud Storage Solutions*. IBM Redbooks, 2018. ISBN: 9780738456843. URL: https://books.google.no/books?id=-L5YDwAAQBAJ.

[7] IBM. *What is multi-tenant (or multitenancy)?* URL: https://www.ibm.com/topics/multi-tenant (visited on 5th Apr. 2024).

[8] Cloudflare INC. *What is multitenancy?* URL: https://www.cloudflare.com/learning/cloud/what-is-multitenancy/ (visited on 5th Apr. 2024).

[9] Uytterhoeven P. Olups R Dalle Vacche A. *Zabbix: Enterprise Network Monitoring Made Easy*. Packt Publishing, 2017. URL: https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1465681&site=ehost-live (visited on 9th Feb. 2024).

[10] Zabbix LLC. *Zabbix Documentation - Features*. URL: https://www.zabbix.com/documentation/current/en/manual/introduction/features (visited on 5th Apr. 2024).

[11] R. Olups. *Zabbix 1.8 Network Monitoring*. From technologies to solutions. Packt Pub., 2010. ISBN: 9781847197696. URL: https://books.google.no/books?id=fsjNquHrQfYC.

[12] Zabbix LLC. *Zabbix Documentation - Server*. URL: https://www.zabbix.com/documentation/current/en/manual/concepts/server (visited on 5th Apr. 2024).

[13] P. Uytterhoeven. *Zabbix Cookbook*. Community experience distilled. Packt Publishing, 2015, pp. 2–10. ISBN: 9781784392253. URL: https://books.google.no/books?id=gNqFBwAAQBAJ.

[14] Zabbix LLC. *Zabbix Documentation - Requirements*. URL: https://www.zabbix.com/documentation/current/en/manual/installation/requirements (visited on 5th Apr. 2024).

[15] Zabbix LLC. *Zabbix Documentation - Proxy*. URL: https://www.zabbix.com/documentation/current/en/manual/concepts/proxy (visited on 5th Apr. 2024).

[16] Zabbix LLC. *Zabbix Documentation - Agent*. URL: https://www.zabbix.com/documentation/current/en/manual/concepts/agent (visited on 5th Apr. 2024).

[17] P. Uytterhoeven and 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. ISBN: 9781789345247. URL: https://books.google.no/books?id=eyyFDwAAQBAJ.

[18] Robert Szulist. 'Zabbix user permissions explained'. In: (2021). DOI: 2024-05-04. URL: https://medium.com/@r.szulist/zabbix-user-permissions-explained-75c5cb78dc78.

[19] Zabbix LLC. *Zabbix Documentation - Dashboard*. URL: https://www.zabbix.com/documentation/5.0/en/manual/web_interface/frontend_sections/monitoring/dashboard (visited on 5th Apr. 2024).

[20] Zabbix LLC. *Zabbix Documentation - API*. URL: https://www.zabbix.com/documentation/current/en/manual/api (visited on 5th Apr. 2024).

[21] Zabbix LLC. *Zabbix Documentation - Trapper*. URL: https://www.zabbix.com/documentation/current/en/manual/config/items/itemtypes/trapper (visited on 5th Apr. 2024).

[22] Zabbix LLC. *Zabbix Documentation - Sender*. URL: https://www.zabbix.com/documentation/current/en/manual/concepts/sender (visited on 5th Apr. 2024).

[23] Syeda Noor Zehra Naqvi, Sofia Yfantidou and Esteban Zimányi. 'Time series databases and influxdb'. In: *Studienarbeit, Université Libre de Bruxelles* 12 (2017), pp. 1–44.

[24] ScriptBees. 'What is TICK Stack, and why should we consider using it?' In: (2020). URL: https://medium.com/@scriptbees/what-is-tick-stack-and-why-should-we-consider-using-it-5a2bddcd7b10.

[25] InfluxData. *Telegraf*. URL: https://www.influxdata.com/time-series-platform/telegraf/.

[26] InfluxData. *Chronograf*. URL: https://www.influxdata.com/time-series-platform/chronograf/.

[27] InfluxData. *Kapacitor*. URL: https://www.influxdata.com/time-series-platform/kapacitor/.

[28] InfluxData. *InfluxDB 1.x TickStack*. URL: https://www.influxdata.com/time-series-platform/ (visited on 5th Apr. 2024).

[29] InfluxData. *InfluxData - Client Libraries*. URL: https://www.influxdata.com/products/data-collection/influxdb-client-libraries/ (visited on 5th Apr. 2024).

[30] InfluxData. *InfluxData - Scrapers*. URL: https://www.influxdata.com/products/data-collection/scrapers/ (visited on 5th Apr. 2024).

[31] InfluxData. *InfluxData - Ecosystem*. URL: https://www.influxdata.com/products/data-collection/ecosystem/ (visited on 5th Apr. 2024).

[32] InfluxData. *InfluxData Documentation - Manage Organizations*. URL: https://docs.influxdata.com/influxdb/v2/admin/organizations/ (visited on 5th Apr. 2024).

[33] InfluxData. *InfluxData Documentation - Visualize data with the InfluxDB UI*. URL: https://docs.influxdata.com/influxdb/v2/visualize-data/ (visited on 5th Apr. 2024).

[34] InfluxData. *InfluxDb v2 Docs - API*. URL: https://docs.influxdata.com/influxdb/v2/api/ (visited on 5th Apr. 2024).

[35] InfluxData. *InfluxData Documentation - API intro*. URL: https://docs.influxdata.com/influxdb/v2/api-guide/api_intro/ (visited on 5th Apr. 2024).

[36] E. Salituro. *Learn Grafana 10.x: A beginner's guide to practical data analytics, interactive dashboards, and observability*. Packt Publishing, 2023. ISBN: 9781801814331. URL: https://books.google.no/books?id=a7PIEAAAQBAJ.

[37] Grafana Labs. *About Grafana*. URL: https://grafana.com/docs/grafana/latest/introduction/ (visited on 5th Apr. 2024).

[38] A.R. Yeruva and V.B. Ramu. *End-to-End Observability with Grafana: A comprehensive guide to observability and performance visualization with Grafana (English Edition)*. Bpb Publications, 2023. ISBN: 9789355515483. URL: https://books.google.no/books?id=0vrKEAAAQBAJ.

[39] Grafana Labs Community. *Grafana Loki documentation - Operations - Multi-tenancy*. Grafana Labs, 2024. URL: https://grafana.com/docs/loki/latest/operations/multi-tenancy/ (visited on 3rd Apr. 2024).

[40] Grafana Labs. *Grafana Loki - Multi-tenancy*. URL: https://grafana.com/docs/loki/latest/operations/multi-tenancy/ (visited on 5th Apr. 2024).

[41] Grafana Labs. *Grafana Agent*. URL: https://grafana.com/docs/agent/latest/ (visited on 5th Apr. 2024).

[42] The Linux Foundation. *Prometheus Documentation - What is Prometheus*. URL: https://prometheus.io/docs/introduction/overview/ (visited on 5th Apr. 2024).

[43] OpsRamp. *Prometheus vs Grafana: Knowing the difference*. URL: https://www.opsramp.com/guides/prometheus-monitoring/prometheus-vs-grafana/ (visited on 5th Apr. 2024).

[44] A. Boduch. *React and React Native*. Packt Publishing, 2017. ISBN: 9781786469571. URL: https://books.google.no/books?id=jLkrDwAAQBAJ.

[45] M. Schwarzmuller. *React Key Concepts: Consolidate your knowledge of React's core features*. Packt Publishing, 2022. ISBN: 9781803240480. URL: https://books.google.no/books?id=ykqkEAAAQBAJ.

[46] C. Pitt. *React Components*. Packt Publishing, 2016. ISBN: 9781785883729. URL: https://books.google.no/books?id=_97JDAAAQBAJ.

[47] D. Afonso. *State Management with React Query: Improve developer and user experience by mastering server state in React*. Packt Publishing, 2023. ISBN: 9781803244839. URL: https://books.google.no/books?id=rgu6EAAAQBAJ.

[48] Meta Platforms. *React - Components and Props*. URL: https://legacy.reactjs.org/docs/components-and-props.html (visited on 5th Apr. 2024).

[49] Supriyo Saha. 'The React Ecosystem'. In: (2024). URL: https://medium.com/@mm.saha1982/the-react-ecosystem-9f0bb6f1d86c.

[50] E. Elrom. *React and Libraries: Your Complete Guide to the React Ecosystem*. Apress, 2021. ISBN: 9781484266953. URL: https://books.google.no/books?id=5Ur3zQEACAAJ.

[51] Huang Kaizhe and Jumde Pranjal. *Learn Kubernetes Security : Securely Orchestrate, Scale, and Manage Your Microservices in Kubernetes Deployments.* Packt Publishing, 2020, pp. 3–16. ISBN: 9781839216503. URL: https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=2521135&site=ehost-live.

[52] The Linux Foundation. *Kobernetes Documentation - Kubernetes Components*. URL: https://kubernetes.io/docs/concepts/overview/components/ (visited on 5th Apr. 2024).

[53] Matthew Gill. 'What is Kubernetes and How Does It Relate to Docker'. In: (2019). URL: 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-19959388920&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjwwr6wBhBcEiwAfMEQs2DlgFuS7xxGcYuJh_PryFImvFkEXPXVx50iIylQM7TF75fzBwE (visited on 5th Apr. 2024).

[54] Helm Community. *Helm - The package manager for Kubernetes*. URL: https://helm.sh/ (visited on 5th Apr. 2024).

[55] Helm community. *Helm Docs - Charts*. URL: https://helm.sh/docs/topics/charts/ (visited on 5th Apr. 2024).

[56] V. Farcic. *The DevOps 2.5 Toolkit: Monitoring, Logging, and Auto-Scaling Kubernetes: Making Resilient, Self-Adaptive, And Autonomous Kubernetes Clusters*. Packt Publishing, 2019. ISBN: 9781838642631. URL: https://books.google.no/books?id=sPTADwAAQBAJ.

[57] Zabbix LLC. *User guide to setting up Zabbix monitoring of the Kubernetes*. URL: https://www.zabbix.com/integrations/kubernetes (visited on 5th Apr. 2024).

[58] Michaela DeForest. *Zabbix Meetup online, January 2023: Monitoring Kubernetes with Zabbix*. Youtube. 2023. URL: https://www.youtube.com/watch?v=2dPFvJrK9Mw.

[59] Zabbix LLC. *Zabbix Helm Chart*. URL: https://git.zabbix.com/projects/ZT/repos/kubernetes-helm/browse?at=refs%2Fheads%2Frelease%2F6.4 (visited on 5th Apr. 2024).

[60] InfluxData. *InfluxData - Kubernetes Monitoring*. URL: https://www.influxdata.com/solutions/kubernetes-monitoring-solution/ (visited on 5th Apr. 2024).

[61] gunnaraasen. URL: https://github.com/influxdata/kube-influxdb (visited on 5th Apr. 2024).

[62] HoneyPot. *Prometheus: The Documentary*. Youtube. 2022. URL: https://www.youtube.com/watch?v=rT4fJNbfe14.

[63] Prometheus. *Overview: Prometheus*. URL: https://prometheus.io/docs/introduction/overview/.

[64] Sushant Kapare. 'Setup Prometheus monitoring on Kubernetes using Grafana'. In: (2023). URL: https://medium.com/@sushantkapare1717/setup-prometheus-monitoring-on-kubernetes-using-grafana-fe09cb3656f7 (visited on 7th Apr. 2024).

[65] M. Grinberg. *Flask Web Development*. O'Reilly Media, 2018. ISBN: 9781491991695. URL: https://books.google.no/books?id=cVlPDwAAQBAJ.

[66] M. Heydt. *Learning pandas*. Packt Publishing, 2017. ISBN: 9781787120310. URL: https://books.google.no/books?id=Sng5DwAAQBAJ.

[67] Ida og Stølen Ketil Solheim. 'Teknologiforskning – hva er det?' In: *SINTEF Rapport A160, 22 p* (2007).

[68] Lydia Matheson Jill Jesson and Fiona M. Lacey. *Doing your literature review: traditional and systematic techniques*. London Sage, 2011.

[69] NTNU Undervisning. *Litteraturstudie som metode*. YouTube video. Dec. 2018. URL: https://www.youtube.com/watch?v=KF3PtpaDsm8 (visited on 8th Feb. 2024).

[70] Zabbix LLC. *Zabbix Documentation*. Zabbix LLC, 2021.

[71] Arturs Lontons. 'Deploying and configuring Zabbix 5.4 in a multi-tenant environment'. In: *Zabbix Blog* (2021).

[72] Mudit Mathur. ‘Grafana: Powerful Metrics Analytics and Visualization’. In: *Medium* (2023). URL: https://muditmathur121.medium.com/grafana-powerful-metrics-analytics-and-visualization-276457150594 (visited on 3rd Apr. 2024).

[73] InfluxDB Community. *InfluxDB v2 Documentation*. InfluxData, 2024. URL: https://docs.influxdata.com/influxdb/v2/ (visited on 3rd Apr. 2024).

[74] Todd Persen. ‘Announcing Multi-Tenant Grafana Support for InfluxDB Cloud Service’. In: *InfluxData Blog* (2016). URL: https://www.influxdata.com/blog/announcing-multi-tenant-grafana-support-for-influxcloud/ (visited on 3rd Apr. 2024).

[75] pydata.org. *Pandas Documentation - IO tools*. URL: https://pandas.pydata.org/docs/user_guide/io.html#io-perf (visited on 9th Apr. 2024).

[Denne siden er blank med hensikt]