Høgskulen på Vestlandet

BACHELOROPPGAVE

Verktøy for sentral loggføring i proton CT

Tool for centralized logging in proton CT

Gruppe D45

Anders Johan Grimnes Pedersen, Etkar Hemit, Nichlas William Løneberg

DAT191

Fakultet for ingeniør- og naturvitenskap

Institutt for datateknologi, elektroteknologi og realfag

Veileder: Håvard Helstrup

Innleveringsdato: 23. mai 2022

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Verktøy for sentral loggføring	*Dato:* 23.05.2022
Forfatter(e): Anders Johan Grimnes Pedersen, Etkar Hemit, Nichlas William Løneberg	Antall sider u/vedlegg: 54 Antall sider vedlegg: 89
Studieretning: Dataingeniør	*Antall disketter/CD-er:*
Kontaktperson ved studieretning Håvard Helstrup*:*	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Institutt for fysikk og teknologi ved Universitetet i Bergen	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Matthias Richter	*Telefon:*

Sammendrag:

Et protonterapisenter er under oppbygging ved Haukeland universitetssykehus. I denne sammenheng utvikler Bergen pCT prosjektet et proton CT apparatur som skal installeres på senteret. Apparaturen inneholder flere datamaskiner, som står for ulike oppgaver. Bachelor gruppen har fått i oppgave å lage et verktøy som sentraliserer loggmeldingene disse datamaskinene genererer. Verktøyet skal kunne nås gjennom en web-applikasjon, der det kan filtreres, søkes og visualiseres loggstatistikk.

Stikkord:

Logg-aggregering	Proton CT	Monitorering

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

FORORD

Dette prosjektet er en bacheloroppgave som har tilhørighet til studieprogrammet Dataingeniør på Høgskulen på Vestlandet i Bergen. Prosjektet er gjennomført i samarbeid med The Bergen pCT Project ved UiB. Denne rapporten beskriver arbeidet gjort i prosjektet "Verktøy for sentral loggføring i proton CT" og er skrevet av Etkar Hemit, Anders Johan Grimnes Pedersen og Nichlas William Løneberg.

Gruppen er takknemlig for å kunne ta del i et så viktig og interessant prosjekt som pCT. Gjennom våres bidrag håper vi å føre pCT teamet et steg nærmere et produkt som vil revolusjonere strålebehandling i helsetjenesten. Samtidig ville det gi oss en god mulighet til å tilegne oss nye ferdigheter innen utvikling. Dersom produktet vårt blir brukt som et verktøy, håper vi det vil det skape god nytteverdi for både utviklingen av proton CT og bruken av apparaturen i behandlingssammenheng.

Vi ønsker å takke veileder Håvard Helstrup for gode tilbakemeldinger og relevante diskusjoner under prosjektperioden.

Vi ønsker også å takke Matthias Richter og andre i pCT prosjektet for nyttig kunnskap, hjelp og innspill som har hjulpet å forme systemet.

INNHOLDSFORTEGNELSE

FIGURLISTE

TABELLISTE

ORDLISTE

Ord	**Forklaring/Definisjon**
ALICE	A Large Ion Collider Experiment
API	Programmeringsgrensesnitt
CLI	Command line interface eller terminal
IFT	Instituttet for fysikk og teknologi
IaaS	Infrastructure as a Service
Logg-aggregering	Prosessen som omhandler innsamling av loggdata fra ulike miljøer
pCT	Proton computed tomography
Query	En forespørsel om data fra database. Kan også være innsettelse av data i database.
SSH	Secure Shell Protocol
UiB	Universitetet i Bergen

1 INNLEDNING

I dette kapittelet vil bakgrunnen for prosjektet bli introdusert. Punktene under vil ta for seg blant annet kontekst, målsetting og motivasjon for dette prosjektet, og sammenhengen til det overordnede prosjektet proton CT (pCT).

1.1 Kontekst

Protonterapi er en ny form for strålebehandling, der protoner brukes som strålekilde, i stedet for gammastråler som benyttes i dagens strålebehandling (Aadnevik & Pettersen, 2019). Protonterapi samler stråledosen i et begrenset område og gir lavere stråledoser i omkringliggende vev. Dette er spesielt aktuelt ved kreft i sensitive organer der det er svært viktig med nøyaktig posisjonsbestemmelse.

Et protonterapisenter er under oppbygning ved Haukeland universitetssykehus. I denne sammenheng utvikles det et piksel-basert apparatur for rask og nøyaktig måling av CT-bilder generert av protonstrålen. Bergen pCT prosjektet er et internasjonalt samarbeidsprosjekt med base på Institutt for fysikk og teknologi (IFT) ved Universitetet i Bergen (UIB). Prosjektet utvikler en apparatur som er basert på detektorteknologi utviklet for ALICE-eksperimentet på CERN (IFT, 2021). Den ferdige apparaturen skal stå klart på Haukeland i 2024 og gruppen skal være med å utvikle en del av programvaren som bachelorprosjekt.

Programvaren gruppen har fått i oppgave å utvikle er et verktøy for sentral loggføring. Verktøyet skal samle loggmeldinger som er spredt blant forskjellige moduler i pCT-systemet og presentere de i ett dedikert web grensesnitt. Instituttet for fysikk og teknologi har ikke et slikt verktøy fra før og er heller ikke spesielt kjent med hvordan det skal gjøres. Det er derfor opp til gruppen å gjøre undersøkelser, eksperimentere og finne de beste løsningene underveis i utviklingen. Oppgaven består dermed i stor grad av å sette seg inn i logg-aggregering og hvordan dette fungerer. Programvaren skal inkluderes i et distribuert system, med moduler som kjøres gjennom ferdig konfigurerte dockerkjerner. Dockerkjernene sikrer at loggføringsverktøyet enkelt kan kjøres distribuert på forskjellige maskiner, uten å måtte installere drivere og nødvendig programvare manuelt på hver enkelt maskin.

1.2 Motivasjon

Utlesning fra proton CT apparaturen er viktig for å oppdage feil, kartlegge forbedringspotensial og analysere systemoppførsel. Det ferdige produktet består av et detektorsystem (digital tracking calorimeter (DTC) bestående av 43 lag med sensorer (Grøttvik, 2021)), en partikkelakselerator,

kjølingsanlegg, diverse sensorer og flere datamaskiner. Programmene som kjører på de forskjellige datamaskinene genererer loggmeldinger fra systemhendelser i sanntid. For å lese ut disse loggmeldingene, kreves det at man manuelt går inn i hver enkelt modul man ønsker å inspisere. Dette er tidkrevende, uoversiktlig og unødvendig komplisert.

Loggføringsverktøyet sentraliserer og lagrer loggdata, og gjør det lett tilgjengelig i et brukervennlig grensesnitt. Dette gjør det enklere å overvåke og behandle loggdata effektivt. Det er viktig å overvåke loggdata, slik at potensielle feil og sikkerhetshull blir oppdaget.

Når sluttproduktet er ferdig installert på Haukeland, skal en som ikke er så datakyndig kunne bruke systemet. PCT systemet består av mange spredte moduler, og da er det praktisk at loggmeldingene er sentralisert, der de lett kan monitoreres. Loggføringsverktøyet skal også brukes under utviklingen av pCT systemet for testing og feilsøking.

1.3 Prosjekteier

Proton CT prosjektet er et internasjonalt samarbeidsprosjekt med base på IFT ved Universitetet i Bergen. I tillegg til å tilby både bachelor- og masteroppgaver knyttet til pCT prosjektet, tilbyr instituttet studier innenfor en rekke disipliner i fysikk, teknologi og energi. IFT er ett av syv institutter underlagt Det matematisk-naturvitenskapelige fakultet ved UiB, som til sammen danner grunnlaget for et bredt spekter av forskning, utdanning og innovasjon (IFT, 2014).

Det er mange viktige samarbeidspartnere i forbindelse med pCT prosjektet. I Norge står UiB, HVL og Haukeland Universitetssykehus for utviklingen av et proton CT system med sensorteknologi fra partikkelfysikk-laboratoriet CERN. Helse Bergen er en samarbeidspartner som står for oppbygging av et protonterapisenter ved Haukeland sykehus. Senteret er planlagt å stå klart til å ta imot pasienter i 2024 og vil benytte apparaturen som er under utvikling.

1.4 Problembeskrivelse og mål

Problembeskrivelse

Det tekniske systemet som omfatter pCT prosjektet i sin helhet er spredt blant mange moduler. Feilsøking og monitorering må gjøres manuelt ved å analysere det området av systemet som ønskes å utforske. Som følge vil det ta unødvendig lang tid å undersøke hendelseslogger for diverse komponenter i systemet. Dette fører til følgende problemstilling: Hvordan utvikle et fleksibelt og enkelt kjørbart verktøy som samler, håndterer og presenterer hendelseslogger i et distribuert system?

En vellykket løsning vil være at hendelseslogger fra forskjellige moduler samles i en nettbasert webapplikasjon der systemet kan monitoreres. Slik kan man enkelt finne ut årsaken til at systemet oppførte seg på en viss måte, på et gitt tidspunkt. Ved å samle loggmeldinger og visualisere tilhørende logg-statistikk i et brukervennlig verktøy, vil det være enkelt å ha oversikt over hele systemet. Oppdragsgiver ønsker at det skal være enkelt å søke og filtrere hendelser i systemet på et valgt tidspunkt i historikken til pCT systemet.

Mål

Målet til bachelorprosjektet er å ha et ferdig sluttprodukt som møter kravene til oppdragsgiver. Det innebærer et produkt i form av en online webapplikasjon med et brukervennlig grensesnitt. Produktet skal samle loggmeldinger fra flere moduler til ett system. De viktigste funksjonene til applikasjonen er søking og filtrering i databasen til systemet. Videre er det krav om at loggmeldingshistorikken til systemet skal være tilgjengelig for en lengre periode og være enkelt å finne fram i. Oppdragsgiver ønsker og at atferden til systemet skal vises som grafer i et dashbord i applikasjon.

1.5 Oppbygging av rapporten

Rapporten følger en mal utlevert av HVL og består av ni kapitler med hvert sitt fokusområde. Noen av delkapitlene er justert og tilpasset denne bacheloroppgaven for å oppnå et mer relevant resultat. Fokusområdene er likevel ivaretatt.

Kapittel 1 – Innledning: Prosjektet presenteres og beskriver kontekst og motivasjon. Her uttrykkes også problembeskrivelsen og målet til prosjektet; hvorfor det er viktig, og for hvem.
Kapittel 2 – Prosjektbeskrivelse: Forteller om den praktiske bakgrunnen til prosjektet og hvilke arbeid som er gjort på forhånd. Initielle krav og idéer fra oppdragsgiver blir introdusert samt avgrensninger og ressurser til prosjektet.
Kapittel 3 – Design av prosjektet: Lister og drøfter forskjellige alternativer til løsning og begrunner valgt løsning. Verktøy som benyttes i oppgaven blir forklart her. Beskriver utviklingsmetoder og planlegging av prosjektet, hvordan risikoer er vurdert og hvordan sluttproduktet blir evaluert.
Kapittel 4 – Detaljert løsning: Forklarer i dybden hvordan løsningen trinnvis er satt sammen av de forskjellige systeminstansene.
Kapittel 5 – Resultater: Beskriver hvilke evalueringsmetoder som er brukt og hvordan resultatene bidrar til det overordnede målet.
Kapittel 6 – Diskusjon: De ulike tilnærmingene for å oppnå resultater blir drøftet og utførelsen av prosjektet.

Kapittel 7 – Konklusjon og videre arbeid: Et sammendrag av sluttresultatet presenteres og hvorvidt målet for prosjektet er nådd. Hva som kan gjøres av videre arbeid blir også forklart.
Kapittel 8 – Referanser: Informasjonen som er brukt i prosjektet blir listet i en oversikt i form av kilder.
Kapittel 9 – Vedlegg: Relevante vedlegg til oppgaven

2 PROSJEKTBESKRIVELSE

Dette kapittelet handler om bakgrunnen til prosjektet, hvilke krav som ble stilt og teamets første løsningsidéer. Ressursene som brukes og litteratur om problemstillingen er også beskrevet her. Den tildelte oppgaven er å utvikle et verktøy som samler inn, behandler og viser loggmeldinger fra proton CT.

2.1 Praktisk bakgrunn

2.1.1 Tidligere arbeid

Det eksisterer ikke noe tidligere arbeid på et logg-aggregeringsverktøy i pCT prosjektet. Det er derfor opp til gruppen å komme opp med en løsning.

2.1.2 Initielle krav

Oppgaven som ble tildelt var åpen for forskjellige løsninger, men hadde klare rammer. Det skulle utvikles et nettbasert verktøy for å hente og samle loggmeldinger fra systemet, behandle dem og deretter presentere dem. UiB har bygget pCT systemet i et Linux-miljø. Dermed ble dette et krav til operativsystemet som skal benyttes. Videre stilles det enkelte funksjonelle krav til programmet:

- Lagre historikken til alle logghendelser til systemet
- Må ha søkefunksjon
- Evne til å filtrere logghendelser
- Presentere utvalgte hendelser ved hjelp av grafiske verktøy

2.1.3 Initiell løsnings-idé

Med de initielle kravene festet til oppgaven, var det opp til gruppen hvordan programmet utvikles og hvilke hjelpemidler som benyttes. Etter litt planlegging ble det klart at det skulle utvikles en web applikasjon med et enkelt brukergrensesnitt. De funksjonelle kravene ble fordelt i *backend*- og *frontend*-delen av programmet.

Søke- og filtreringsfunksjonene skulle implementeres i webgrensesnittet (frontend). Det skulle være mulig å søke etter ønskede loggmeldinger ved hjelp av et søkefelt og en søkeknapp på nettsiden. I tillegg ville programmet inneholde et filtreringsfelt som åpnes ved å trykke på en knapp. Her ville det være mulig å huke av krav til loggmeldinger som skulle vises. Til sammen kunne disse funksjonene hente ut enhver loggmelding som hadde blitt opprettet i systemet. Det skulle være mulig å klikke seg inn på en loggmelding og få en utdypet forståelse av systemhendelsen ved hjelp av et visuelt grafverktøy.

For å lagre, behandle og hente ut loggmeldinger trengtes det en strukturert og solid backend. Gruppen var enige om at databaseverktøyet som skulle benyttes måtte være egnet til de funksjonelle kravene. Særlig var det viktig at databasen kunne håndtere loggmeldinger i ulike formater på en rask måte. Dermed ble det hensiktsmessig med et dokument-orientert databaseverktøy som håndterer stor skrivebelastning. Alle loggmeldinger som genereres i systemet skulle bli lagret her og kunne hentes ut ved behov.

2.2 Avgrensninger

I oppgavebeskrivelsen fra oppdragsgiver står det oppført at gruppen kan fokusere på ett eller flere av fire punkter. Ett av punktene omhandler oppgaven Verktøy for sentral loggføring. Etter samtale med oppdragsgiver og kort tidsfrist, ble det naturlig å avgrense prosjektet til dette punktet. I utgangspunktet fikk gruppen stor frihet til å velge hvordan applikasjonen skulle utformes. Hvilke funksjoner programmet skulle ha ble raskt bestemt og godkjent av oppdragsgiver. Brukeren av programmet skal se tre synlige funksjoner; søk, filtrering og presentasjon av loggmeldinger. Siden alle loggmeldinger enkelt kan finnes ved disse funksjonene, var det unødvendig å implementere flere funksjoner i webgrensesnittet.

Det var derimot noen forhåndsbestemte avgrensninger som gruppen måtte forholde seg til. Operativsystemet hvor programmet skulle utvikles ble avgrenset til Linux og det var bestemt at applikasjonen skulle være nettbasert. GitLab er oppbevaringsstedet for all programvare tilknyttet pCT prosjektet, og ble derfor den forbeholdte plattformen for kode- og informasjonsutveksling.

2.3 Ressurser

For å utvikle en programvare for sentral loggføring i pCT, er gruppen utstyrt med egne datamaskiner. Ingen av maskinene hadde operativsystemet Linux, som er nødvendig for å løse oppgaven. Gruppen installerte Linux på ulike måter avhengig av operativsystem og prosessor (CPU) – enten direkte på maskinen eller gjennom en virtuell maskin. All programvare benyttet i prosjektet ble brukt i Linux-miljøet.

Gjennom prosjektet er det tilgang på både interne og eksterne ressurspersoner. Intern veileder er Håvard Helstrup, professor ved HVL, som er involvert i pCT prosjektet. Helstrup bistår med veiledning for rapportskriving samt tekniske kunnskaper innen datateknologi. Eksterne ressurspersoner er Matthias Richter, masterstudenter og doktorgradskandidater som jobber med ulike deler av pCT systemet på UiB. Richter innehar teknisk kompetanse for store deler av prosjektet. Han har skrevet en doktorgrad om utvikling og integrasjon av nettbasert dataanalyse

for ALICE eksperimentet. Teknologien som ble brukt var ny for gruppen, så det var nyttig at det ble opprettet dialog med mer erfarne fagpersoner.

Gruppen vekslet mellom arbeidslokalet på IFT og HVL etter hva slags arbeid som ble utført. Rapportskriving og obligatoriske arbeidskrav egnet seg best på HVL ettersom man kunne sitte på egne rom uforstyrret. IFT ble mest benyttet til møter og utviklingsdelen av prosjektet. Der kunne man kommunisere med ulike fagpersoner for å få innsikt i det større systemet.

Kommunikasjonen innad i gruppen samt mellom gruppen og ressurspersoner ble fordelt blant mange plattformer. Mellom gruppen og intern veileder ble e-post, Zoom og fysisk møte benyttet. Slack ble i tilføyelse til de nevnte plattformene brukt til kommunikasjon med eksterne ressurspersoner. I tillegg ble GitLab brukt til kode- og informasjonsdeling. Gruppen brukte seg imellom:

- Discord til deling av filer, litteratur, arbeid og dialog
- Facebook Messenger for chatting
- Google Drive og Microsoft Word for fildeling
- Grupperom på HVL eller IFT for utvikling, planlegging og rapportskriving

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning logg-aggregering

Oppgaven stiller krav til et verktøy for utlesning og samling av loggdata. Det trengs derfor en backend-løsning som monitorerer og leser ut loggmeldinger. Loggdata må oppbevares et sted, og i den sammenheng er det praktisk med en database. Det er også et krav om visualisering av loggmeldinger i et webgrensesnitt. Her er det flere fremgangsmåter og verktøy som kan være nyttig i prosessen. Videre i dette kapittelet utdypes det mer i detalj rundt valget av logg-aggregeringsverktøy, databaseløsning og fremgangsmåte for visualisering.

3.1.1 Alternativ løsning for logg-aggregering

ELK Stack

ELK Stack er en kolleksjon bestående av 4 produkter, som til sammen utgjør et verktøy for sentral loggføring (Horovits, 2020). Logstash, Beats, Elasticsearch og Kibana utgjør disse produktene og alle med åpen kildekode utviklet av selskapet Elastic. Logstash er et logg-aggregerings verktøy som med hjelp fra Beats samler data fra diverse kilder, transformerer og tilpasser de, før de blir sendt videre til diverse støttede destinasjoner. Elasticsearch er et søk og analyseverktøy basert på Apache Lucene søkemotoren som indekserer og lagrer data. Kibana er et visualiserings nivå bygget oppå Elastisearch, som tilbyr brukere analyse og visualisering av data, i et brukergrensesnitt. ELK Stack er en av de mest brukte verktøyene for loggmeldinger, og er bredt støttet.

Grafana Loki

Grafana Loki er et horisontalt skalerbart logg-aggregeringssystem designet for å være svært kostnadseffektivt og enkelt å bruke (Grafana, u.å.a). Det er et relativt nytt verktøy utviklet av Grafana Labs i 2018 og er inspirert av Promotheus. I stedet for å indeksere hele loggmeldingen, komprimerer systemet meldingen og indekserer heller attributtene til meldingen for en mer effektiv løsning.

Diskusjon rundt logg-aggregering

Både Grafana Loki og ELK stack er gode løsninger som tilbyr verktøyene som trengs. ELK Stack kan være mer ressurskrevende enn Grafana Loki, men til gjengjeld leverer det raskere søk og filtrering. Begge er åpen kildekode, og krever ikke betalte lisenser. Grafana Loki ble anbefalt av oppdragsgiver, mye grunnet at det er benyttet i ALICE.

3.1.2 Alternativ løsning for database

Valg av databaseløsning avhenger av det valgte Logg-aggregeringsverktøyet. Gruppen endte opp med å velge Grafana Loki som løsning, dette blir drøftet i delkapittel 3.2. Loki trenger to lagringssystemer, en for chunks og en for indekser, derfor blir de gitte databaseløsningene vurdert i dette delkapittelet. Bruken av chunks og indekser blir forklart i detalj i kapittel 4.

Lokal lagring - BoltDB i kombinasjon med filsystem lagring

Lagring av data lokalt på serveren som hoster Loki, er et alternativ gruppen har vurdert. Da kan BoltDB i kombinasjon med filsystemlagring være en god løsning (Grafana, u.å.b).

BoltDB Shipper er Grafana Loki sin løsning for lokal lagring av indekser (Grafana, u.å.c). BoltDB kombineres med filsystem lagring for chunks, og er en lettdreven database skrevet i språket Go (Davidson, 2021). Lagring av data skjer på det lokale disksystemet. BoltDB er en NoSQL nøkkelbasert database uten et query språk. I stedet for å benytte queries for søk, benytter BoltDB indekser. Databasen har form som et binært tre, der hver node inneholder en peker til neste node, og hvert blad er selve objektet som er lagret. Et søk går veldig raskt og er spesielt egnet for read-heavy applikasjoner. Hele BoltDB databasen er en enkel fil.

Som nevnt fungerer BoltDB som en lagringsplass for Loki indekser. Det kreves dermed en annen lagringsform for chunks lokalt på maskinen. For lagring av chunks kan det ordinære filsystemet benyttes. Da sørger Loki for mappestruktur og lagring. Ved lagring blir en chunk tildelt en indeks, som blir lagt til i BoltDB databasen. Loki søker opp indekser i BoltDB databasen og benytter disse indeksene for å lokalisere chunks i mappestrukturen.

Skylagring - Amazon DynamoDB og S3

Gruppen vurderte også lagring av data på skyen, som kan være nyttig i tilfelle det skulle skje noe med host-maskinen. Amazons DynamoDB og S3 var anbefalt av Grafana Labs til skylagring.

Amazon S3 er en skalerbar skylagringstjeneste brukt for lagring av store, ustrukturerte data som blir lagret i “buckets” (Amazon, u.å.a). Fordelen med S3 er at en bucket kan lagre en ubegrenset mengde ustrukturerte data. Data blir lagret i form av objekter og kan være opptil 5TB i filstørrelse. Løsningen her vil da bli å lagre chunks fra Loki i en S3 bucket. Hvert objekt får en nøkkel som blir brukt for å identifisere objektet.

Amazon DynamoDB er en low latency og skalerbar NoSQL database. DynamoDB støtter både vanlige dokumentbaserte modeller i tillegg til nøkkelbaserte modeller (Amazon, u.å.b). DynamoDB blir benyttet til å lagre indeksene generert av Loki, som blir brukt for å referere til en chunk i S3 bucket-en.

Diskusjon rundt databaseløsning

Både BoltDB Shipper og Amazon sine databaseløsninger er bredt støttet av Loki. Dette er hovedgrunnen til at disse løsningene ble tatt i betraktning.

Gruppen har ikke benyttet noen av databaseløsningene tidligere. Dette gjør at kunnskapen må tilegnes, og dette vil gå utover kostnadene (i timer) for å gjøre seg kjent med nye database løsninger. Likevel veier kompatibiliteten mot Loki høyere, enn om det skulle bli valgt en annen databaseløsning gruppen er bedre kjent med.

BoltDB er åpen kildekode og gratis i bruk, det er ikke Amazon sine systemer. For å kunne benytte DynamoDB og S3 trengs en lisens, der det betales for mengden datakapasitet. En bruker kan derimot registrere seg for et års prøveperiode med begrenset datakapasitet.

Når det kommer til de mer tekniske detaljene er det også mye som er likt med Amazon sitt system og BoltDB systemet. I dette kapittelet skal det ikke forklares i detalj og det unnlates derfor å bli for avansert. Både BoltDB og DynamoDB er nøkkelbaserte NoSQL databaser som egner seg svært godt for lese-operasjoner. Når det kommer til chunks, er filsystemlagring og Amazon S3 også i prinsippet veldig like løsninger, men lese og skrive hastigheter kan variere noe. For veldig store mengder data vil et søk gjennom Amazon S3 gå raskere enn i et lokalt filsystem, det motsatte gjelder derimot hvis det ikke er snakk om alt for store mengder data. Det at den ene databaseløsningen er lokal og den andre skybasert vil også ha innvirkning på lese/skrive hastigheter.

Når det kommer til skylagring i forhold til lokal lagring, som er den viktigste forskjellen mellom de to databaseløsningene, er det flere faktorer som kan virke positivt og negativt. Det kan være mer sikkerhetsmessig givende å ha en lokal database på en intern server. Da er all data kontrollert internt i pCT prosjektet og sensitive data blir ikke eksponert til tredjeparts aktører. Velges det en skybasert løsning er det opp til Amazon å stå for sikkerheten, samtidig som de kan tilegne seg sensitiv informasjon.
Amazon sin skyløsning er fordelaktig med tanke på tap av data, da flere backup løsninger er implementert i skyen. Hvis noe skjer med en lokal database kjørende på en server og data på en disk blir slettet eller korrupt, kan data gå tapt. Det kan lages løsninger som f.eks. RAID konfigurasjon, for å lagre backup av data, men dette krever ressurser (Patterson, u.å).

Amazon sine skyløsninger er mindre ressurskrevende enn en lokalt driftet database. Alt av hardware er det Amazon som står for. Dette kan hjelpe med å avlaste UiB sin server, eller gjøre slik at serveren slipper å oppgraderes i lengden. Alt av vedlikeholdsarbeid vil Amazon også stå for. Chunks og indekser kan spise mye diskplass i lengden, her er det og fordelaktig å ha en dynamisk mengde diskplass i en sky.

3.2 Valgt løsning

Når det kommer til logg-aggregeringsverktøy, valgte gruppen Grafana Loki. Dette er mye grunnet anbefalinger fra oppdragsgiver, da det blir benyttet i andre prosjekter som ALICE, som oppdragsgiver er kjent med. Gruppen har også gjort en del undersøkelser i diverse forum og artikler, der Grafana Loki er blitt anbefalt av et stort antall personer, spesielt for å være enkel i bruk og kostnadseffektiv. Utover dette er både Grafana-Loki og ELK-Stack gode løsninger som begge kunne blitt benyttet. Gruppen har som utgangspunkt å benytte Grafana Loki, og eventuelt vurdere bytte til ELK-Stack hvis Grafana-Loki viser seg å underprestere eller ikke virke som planlagt.

Valg av databaseløsning endte på Amazon DynamoDB og S3. Dette er en databaseløsning som er anbefalt for langtidslagring av loggmeldinger i kombinasjon med Loki. Gruppen anså det som praktisk å kunne avlaste serveren med å delegere arbeidet til Amazon sin skyløsning. Det er også praktisk med tanke på backup av data.
Sammen med oppdragsgiver konkluderte gruppen med at det er lurt å teste en skybasert databaseløsning og få dette til og fungere. Det å koble opp mot en skybasert database krever en del mer enn å sette opp en lokal løsning. Da kan løsningen gjenbrukes selv om en annen skybasert databaseløsning skulle bli valgt i ettertid. Sikkerhetsmessig er det ikke et problem å bruke Amazon sin løsning for testing og utvikling av pCT systemet. Når det ferdige systemet skal stå klart på Haukeland i 2024, er det enkelt å konvertere over til f.eks. Azure sin skyløsning, da dette gjøres veldig likt som Amazon sitt oppsett. Selv om den gjeldende versjonen av logg-aggregeringsverktøyet benytter S3 og DynamoDB, har gruppen også valgt å lage en lokal database med BoltDB. Begge versjonene er gjort tilgjengelig, men krever noen små endringer i konfigurasjonen til Loki.

3.3 Valg av verktøy

Det finnes ingen tidligere løsning for en applikasjon som tar for seg sentral loggføring i pCT. Det som finnes fra før, er moduler i et distribuert system som genererer loggmeldinger. Det er derfor viktig at programvare som tas i bruk er kompatible med det eksisterende systemet. Her er de mest vesentlige verktøyene som skal benyttes i prosjektet:

- IDE for programmering av logg-generator
- Promtail - Verktøy for loggscraping
- Grafana Loki - Verktøy for logghåndtering
- Database for lagring av loggmeldinger
- Grafana - Webapplikasjon for visualisering av loggdata

Disse verktøyene blir grundigere forklart i kapittel 4.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Utviklingsmetoden som gruppen har valgt å bruke er den smidige metoden Scrum. Det er et prosessrammeverk som går ut på utvikling og leveranse av komplekse produkter i korte iterasjoner. Disse iterasjonene blir kalt sprints og er tidsintervaller der valgte oppgaver skal bli utført. Dette tillater gruppen å fokusere trinnvis på funksjonene til produktet og korrigere på ting underveis. Prosessen er empirisk og beslutninger som blir tatt er basert på det som er kjent (Schwaber & Sutherland, 2011).

Scrum passer godt til gruppen fordi det legger til rette for ukentlig statusmøter hver tirsdag med oppdragsgiver og resten av pCT-teamet, samt møter med veileder ved behov.

3.4.2 Prosjektplan

Prosjektplanen blir fremstilt i form av et GANTT-diagram som består av tre faser; *forprosjekt*, *systemutvikling* og *rapportering/presentasjon*. Forprosjektfasen gikk i hovedsak ut på å få gruppens visjon av prosjektet på plass, samt å få utviklet en demo som ble vist til oppdragsgiver. Denne fasen ble avsluttet med innlevering og presentasjon av forprosjektrapporten. Utviklingsfasen går ut på å utføre testing og utvikle selve loggføringsverktøyet. Delen av prosjektet som handler om skriving og formidling går under rapportering/presentasjon. GANTT-diagram finnes i prosjekthåndboken som er vedlagt.

3.4.3 Risikovurdering

En risikovurdering sier noe om hva slags hendelser som kan skje, hvorfor de skjer, og hvilke konsekvenser som vil følge (Aven, 2015). Vedlagt i prosjekthåndboken er risikoanalysen som består av hendelser som kan ha en negativ innvirkning for prosjektet. Hver hendelse forklarer sannsynligheten for at det skjer, konsekvenser hvis det skjer og tiltak for å forhindre at det skjer.

Det er kun de mest overordnede risikoene som er tatt for seg i risikovurderingen. De mer detaljerte risikoene, som f.eks. om man får implementert en bestemt funksjon innen estimert tid, føres opp i gruppens agenda. Her følges detaljert tidsbruk og progresjon opp.

3.5 Evalueringsplan

Det er to punkter som tas hensyn til når produktet skal evalueres:

- Verifisering - Er produktet er utviklet riktig og med god nok kvalitet?
- Validering - Har gruppen utviklet riktig produkt?

Verifisering vil foregå gjennom integrasjonstesting og systemtesting. Integrasjonstesting handler om å teste hvordan to eller flere komponenter i systemet fungerer sammen. Systemet består av

mange deler, derfor er det viktig å sikre at samspillet mellom de fungerer slik man har tenkt. Systemtest vil skje etter integrasjonstesting, og da skal hele systemet testes. Dette er nyttig for å avdekke feil som ikke ble fanget opp av integrasjonstestene.

For validering planla gruppen å gjennomføre brukertesting for relevant fagpersonell i pCT prosjektet. Brukertesting vil foregå i to iterasjoner. Først ved at fagpersoner tester en prototype av applikasjonen, deretter ved at de tester sluttproduktet. Etter begge testene vil testpersonene gi en evaluering slik at forbedringer kan gjøres. Produktet må være bygget på en slik måte at det er lett å gjøre endringer for å tilpasse brukernes behov.
Som resultat vil disse testmetodene bidra til å skape et brukertilpasset produkt av god kvalitet. For å implementere eventuelle behov fra brukere underveis i utviklingen, er det viktig å være åpen for endringer og bygge systemet på denne filosofien.

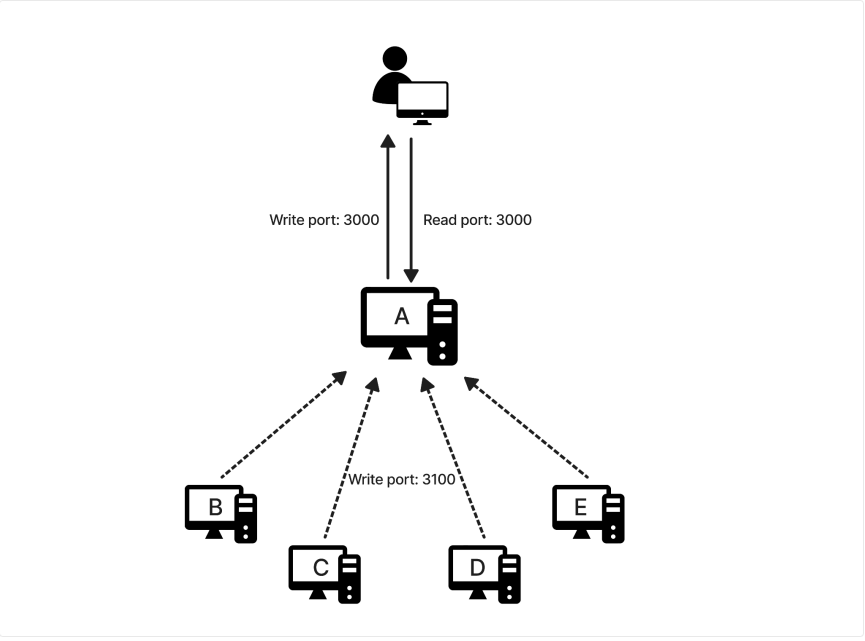
4 DETALJERT LØSNING

Systemet er utviklet med hensyn på at det skal være lettdrevet, enkelt å installere og kompatibelt med alle Linux maskiner. Programvaren kjører i ferdig konfigurerte Docker containere, som kan kjøres i gang med en enkel kommando via Docker Compose. Programvaren er delt inn i to deler; serverside-package og klientside-package. En package består av ett sett med filer, og inneholder alt som er nødvendig for å kjøre verktøyene. Serverside-package er konfigurert til å kjøre på en Linux-server, og inneholder Promtail, Grafana Loki, og Grafana. Klientside-package er konfigurert til å kjøre på en vilkårlig Linux-klient, og inneholder Promtail. Disse pakkene er tilgjengelig på bachelorprosjektets GitLab repository. Ved å laste ned en pakke har du alt som trengs for å kjøre systemet på en maskin (så lenge du har Docker og Docker Compose installert). Dette er en viktig motivasjonsfaktor bak konseptidéen for produktet gruppen har utviklet. Det øker brukervennlighet ved installasjon og gjør det lett å kunne legge til nye noder for monitorering i pCT systemet.

I dette kapittelet kommer en utdypende forklaring av de forskjellige systeminstansene i logg-aggregeringsverktøyet og hvilken rolle de spiller i det distribuerte systemet, samt kommunikasjon på tvers av disse.

4.1 Det distribuerte systemet

Systemet er satt sammen av ulike maskiner med forskjellige funksjoner, som til sammen danner et distribuert system. Dette systemet består av maskiner som genererer loggmeldinger (noder), en servermaskin og en ekstern maskin som representerer brukeren av systemet.



Figur 1: Sammenheng mellom maskinene i det distribuerte systemet

4.1.1 Noder

De ulike modulene i pCT systemet som genererer loggmeldinger er plassert i hver sin maskin. Disse maskinene er kalt noder og representer en komponent i pCT systemet. Hver node har en Docker container hvor Promtail kjører til enhver tid. Promtail er et verktøy som leser loggfiler og sender strømmer med loggdata fra filene til en serverinstanse sammen med et sett med etiketter (labels).

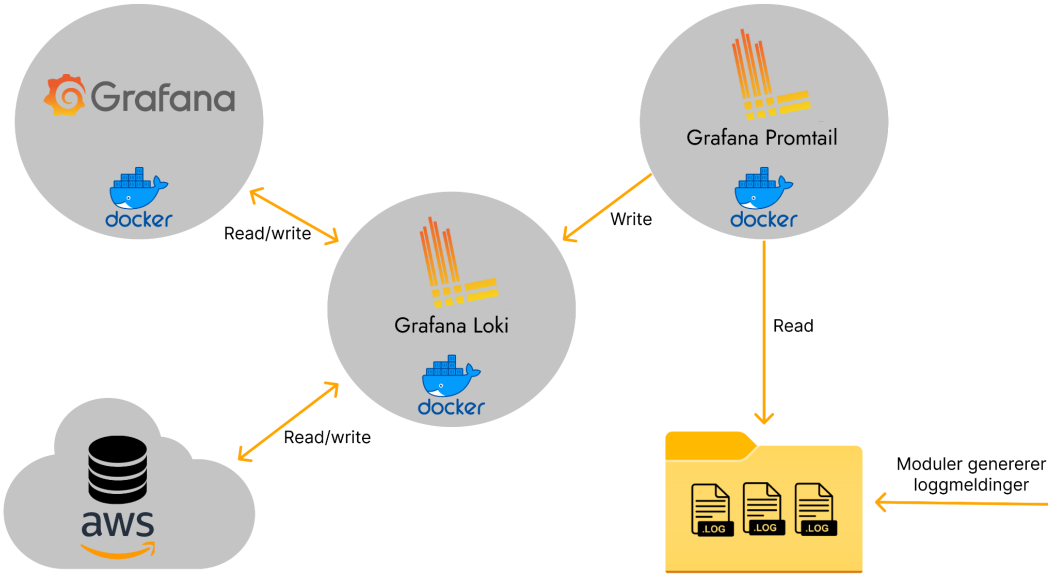
4.1.2 Servermaskin

Servermaskinen i systemet står for sentraliseringen av alle loggmeldinger som genereres. Herfra blir loggmeldingene sendt til databasen for langtidslagring og hentes ut ved behov. På serveren kjører webapplikasjonen Grafana og logghåndteringsverktøyet Grafana Loki til enhver tid. De kan nås fra andre maskiner ved IP adresse og portnummer. Loki prosesserer alle loggmeldingene som er i omløp og Grafana henter ut det som ønskes å monitoreres.

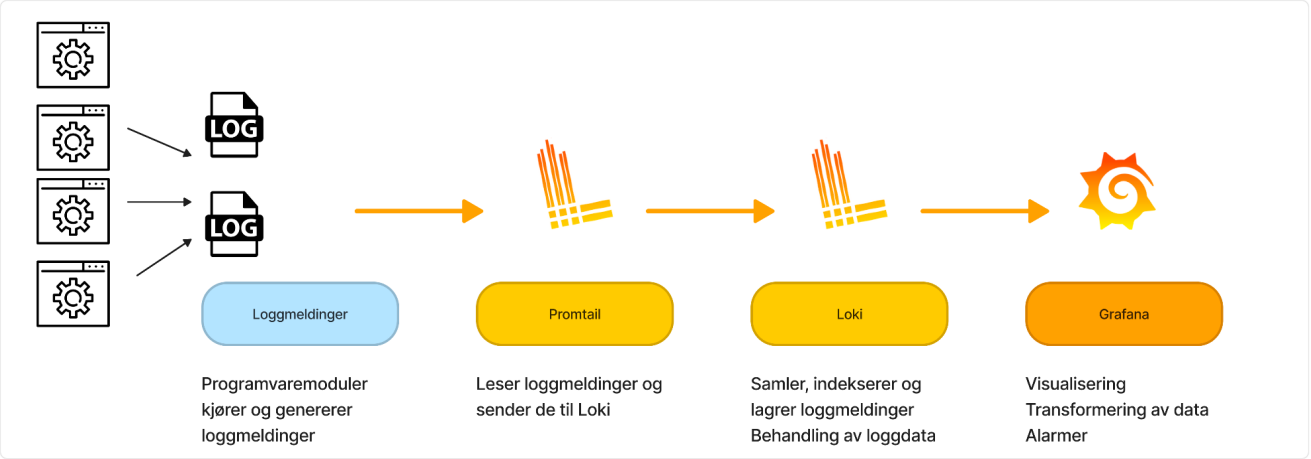
4.1.3 Ekstern maskin

En bruker som vil monitorere systemet kan gå inn i webapplikasjonen Grafana fra en ekstern datamaskin. Det gjøres ved å taste inn serverens IP adresse og porten til Grafana i adressefeltet til en nettleser. Brukeren logger seg inn med brukernavn og passord og kan monitorere systemet gjennom et dashbord.

4.2 Systemarkitektur



Figur 2: Illustrasjon av det overordnede systemet med alle instanser.



Figur 3: illustrerer veien fra en loggmelding blir opprettet til den vises i Grafana Dashboard.

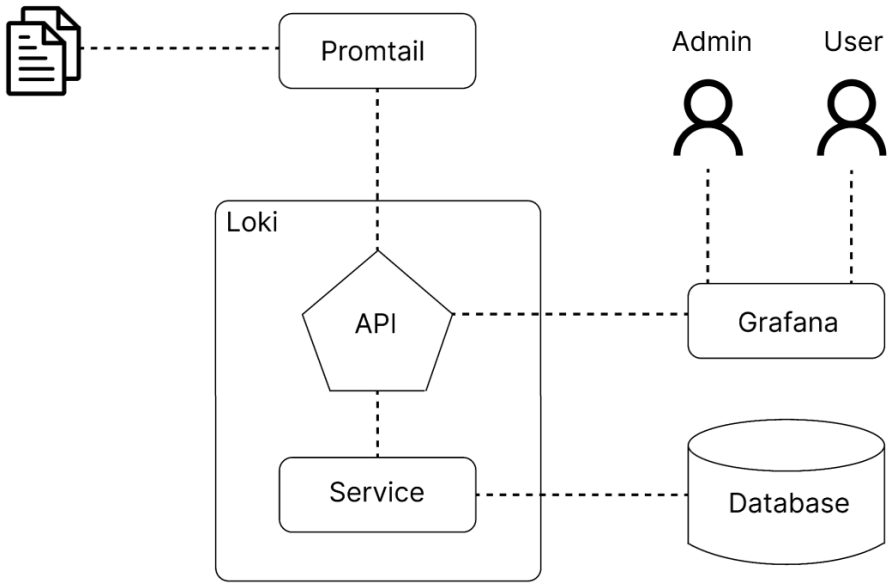
Logg-aggregeringsverktøyet er satt sammen av tre separate programinstanser som utfører hver sin oppgave. Hver instans, Grafana Promtail, Grafana Loki og Grafana, kjører i Docker containere, og kan kjøre uavhengig av hverandre. En Docker container kan på enkelt sett beskrives som en virtuell lettdreven maskin som inneholder alt som trengs for å kjøre et program. Containere isolerer programvare fra omgivelsene, og dermed må all kommunikasjon på tvers av instansene (Loki, Promtail og Grafana) gå via nettverkskommunikasjon på portnumre. Loki fungerer som det sentrale bindeleddet, og all kommunikasjon på tvers av instansene snakker med Loki sitt API på port 3100.

Promtail kjører i en Docker container lokalt på nodene i det distribuerte systemet, samt på serveren, og leser ut loggmeldinger fra spesifiserte filplasseringer på den gjeldende maskinen. Disse sendes videre til Loki sitt API på port 3100.

Loki kjører i en Docker container på serveren og mottar loggmeldingene fra Promtail gjennom sitt API. Loki behandler denne loggdataen, formaterer den og laster opp i databasene. Loki står for databasehåndtering og responderer til queries som mottas gjennom APIet.

På serveren innhenter Promtail loggmeldinger, samt sender query-requests til Loki gjennom Loki sitt API. Grafana er brukergrensesnittet i systemet, og all brukerinteraksjon går gjennom dette UIet. Query-requests fra brukerinput sendes fra Grafana til Loki sitt API, der queries blir behandlet, innhentet fra databasen, og returnert gjennom APIet.

All loggdata fra Loki kan visualiseres og behandles i Grafana sitt webgrensesnitt, som kan nås fra alle eksterne enheter gjennom en nettleser på portnummer 3000. Det kreves innlogging for å komme seg inn på Grafana Dashboardet. Admin-brukere har tilgang til å redigere og konfigurere oppsettet av dashbordet, mens vanlige brukere kun har tilgang til søk, filtrering og visualisering.



Figur 4: Arkitekturskisse som illustrerer kommunikasjon mellom komponenter, og brukermiljø.

4.3 Docker

For å gjøre prosessen av å sette opp verktøyet på hvilken som helst Linux-maskin lettere, ble Docker valgt som plattform. Docker kjører tjenester i en "container", som er et lett operativsystem som kun inneholder det som trengs for å kjøre en tjeneste. Docker sikrer at hver container tildeles tilstrekkelig minne og CPU kapasitet uten å gjøre den overflødig (Jacobsen, 2021).

Her er nøkkelkonseptene som trengs for å bruke Docker plattformen (Kisller, 2021):

4.3.1 Docker Engine

Docker Engine er applikasjonen som må lastes ned på host-maskinen for å kunne bygge, kjøre og håndtere Docker containere. Dette er kjernen av Docker systemet og forener alle komponenter i plattformens på ett sted.

4.3.2 Docker Daemon

Denne komponenten prosesserer API forespørsler for å håndtere de forskjellige aspektene ved installasjonen, som images, containere og lagringsvolumer.

4.3.3 Docker Client

Dette er brukergrensesnittet for kommunikasjon med Docker systemet. Den tar inn kommandoer via command-line interface (CLI) og sender dem til Docker daemon.

4.3.4 Docker Image

En read-only mal som brukes for å lage Docker containere. Et image består av mange lag (layers) som utgjør en sammensatt pakke. Pakken har alt av installasjoner, avhengigheter, biblioteker, prosesser og applikasjonskode nødvendig for å lage et fungerende container miljø.

4.3.5 Docker Container

En kjørende instans av et Docker image. Når en container kjører, blir det lagt til et skrivbart lag på toppen av image-lagene, kjent som et container layer. Dette blir brukt for å lagre endringer gjort i containeren i løpet av kjøretiden.

4.3.6 Docker Registry

Et skybasert distribuert system for lagring av Docker images. Et Docker register organiserer images til oppbevaringssted, kalt repositories. Hvert repository inneholder forskjellige versjoner av et Docker image som deler det samme image-navnet. Docker Hub er det mest brukte registeret.

4.3.7 Docker Compose

Compose er ikke et nødvendig konsept for å bruke Docker plattformen, men er nyttig å bruke når flere containere skal kjøre samtidig. Det er et hjelpeverktøy for å definere et multi-container miljø, og sørger for at alle containere i dette miljøet blir startet opp. Docker imagene er definert i en Docker Compose YAML fil. Ved hjelp av en kommando i CLI kan alle containerne kjøres eller stoppes. Dette er praktisk for å slippe å starte og stoppe hver container manuelt.

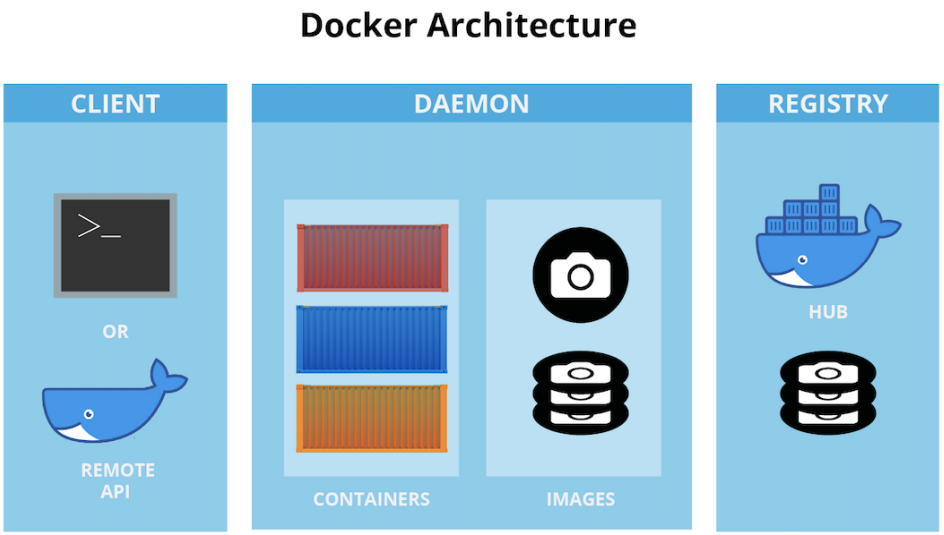
Docker Compose inneholder også andre viktige funksjoner, som ekstra instruksjoner for hvordan containere skal kjøres. Tre av disse er benyttet i løsningen:

Volumes: I Docker Compose kan det defineres shared Volumes. Dette er en delt filmappe mellom Docker containeren og host maskin som konstant holdes synkronisert. Dette er nyttig for at container skal kunne få tilgang til lokale filer på host maskin eller vice versa. Et eksempel er mappen "/var/logs/". For å få lest ut loggmeldingene som er lokalt på host maskin, kan mappen defineres som et shared volume med Docker container. Volumes kan også brukes for lagring av data. Når en container avsluttes, vil all data som ikke er lagret utenfor containeren gå tapt. Derfor kan data som trenger å lagres, bli lagret på host maskin gjennom et Volume.

Network: Ett sett med Docker containere kan defineres i et network. Dette er viktig for kommunikasjon på tvers av containere. Hver container i samme network kan nås og oppdages av andre containere på det samme nettverket (Docker, u.å.a). Containere i et nettverk kjører isolert fra andre nettverk.

Image detection: Ved oppstart av en Docker Compose fil, vil det gjøre et søk etter Docker imaget som ønskes å kjøres. Hvis imaget eksisterer på host maskin, og er av riktig versjon, vil den kjøre det lokale imaget. Eksisterer ikke imaget lokalt på host maskin, vil Compose gjøre et søk etter imaget på Docker Hub. Finnes det her, lastes imaget ned automatisk og kjøres. Dette er nyttig og

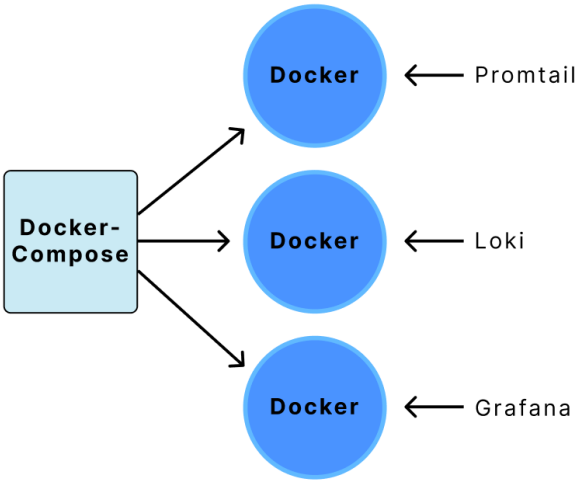
nødvendig i prosjektets løsning, slik at alle nødvendige images blir lastet ned ved førstegangsinstallasjon på et nytt Linux-system.



Figur 5: Docker arkitektur

4.3.8 Docker løsning

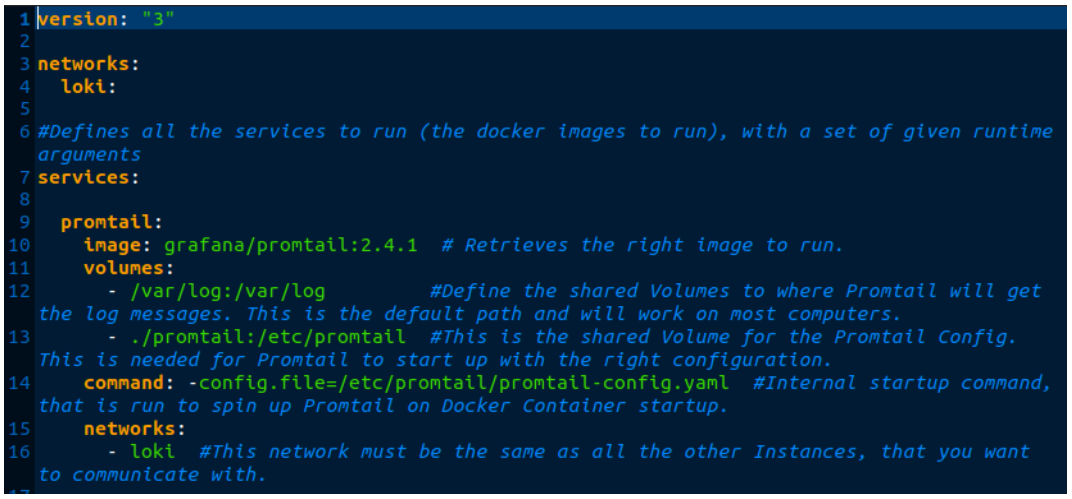
I løsningen har gruppen benyttet Docker Compose for å enkelt kunne kjøre logg-aggregeringsverktøyet gjennom én kommando i CLI. Containerne som kjøres av Compose består av Grafana, Loki og Promtail og blir lastet ned fra Docker Hub registeret, fra Grafana sitt repository. Disse tjenestene er definert i filen docker-compose.yaml. Gruppen har laget en docker-compose fil for server- og klientside, dette er illustrert henholdsvis i Figur 7 og 8. Gruppen opprettet også et Docker image for demoen beskrevet i kapittel 4.7.



Figur 6: Docker compose illustrasjon

```yaml
version: "3"

networks:
  loki:

#Defines all the services to run (the docker images to run), with a set of given runtime arguments
services:
  loki:
    image: grafana/loki:2.4.1  #Defines which image to run
    ports:
      - "3100:3100"  #Mapping a port to the container. Host IP, if not set, MUST bind to all network interfaces. Host and container MUST use equivalent ranges. Either specify both ports (HOST:CONTAINER), or just the container port. In the latter case, the Compose implementation SHOULD automatically allocate any unassigned host port. Always specified as a quoted string.
    volumes:
      - ./loki:/etc/loki  #Volumes are synchronised folders, shared between host and container. Host folder path is defined first, then volume folder path, as such; (HOST:CONTAINER)
    command: -target=all,table-manager
             -config.file=/etc/loki/loki-config.yaml  #overrides the the default command declared by the container image (i.e. by Dockerfile's CMD).
    networks:  #Defines the networks that service containers are attached to. Services communicate with each other through Networks. In this specification, a Network is a platform capability abstraction to establish an IP route between containers within services connected together
      - loki

  promtail:
    image: grafana/promtail:2.4.1
    volumes:
      - /var/log:/var/log
      - ./promtail:/etc/promtail
    command: -config.file=/etc/promtail/promtail-config.yaml
    networks:
      - loki

  grafana:
    image: grafana/grafana:latest
    volumes:
      - grafana-storage:/var/lib/grafana
    ports:
      - "3000:3000"
    networks:
      - loki


volumes:
  grafana-storage:
    external: true
```

Figur 7: docker-compose.yaml filen brukt i serversiden



```yaml
version: "3"

networks:
  loki:

#Defines all the services to run (the docker images to run), with a set of given runtime arguments
services:

  promtail:
    image: grafana/promtail:2.4.1  # Retrieves the right image to run.
    volumes:
      - /var/log:/var/log         #Define the shared Volumes to where Promtail will get the log messages. This is the default path and will work on most computers.
      - ./promtail:/etc/promtail  #This is the shared Volume for the Promtail Config. This is needed for Promtail to start up with the right configuration.
    command: -config.file=/etc/promtail/promtail-config.yaml  #Internal startup command, that is run to spin up Promtail on Docker Container startup.
    networks:
      - loki  #This network must be the same as all the other Instances, that you want to communicate with.
```

Figur 8: docker-compose.yaml filen brukt i klientsiden

4.4 Systeminstanser for logg-aggregering

For å samle informasjon fra applikasjons- og tjenestelogger fra ulike miljøer i pCT, har gruppen benyttet seg av tjenester fra Grafana Labs. Tjenestene er Grafana, Grafana Loki og Promtail, og til sammen utgjør de en logg-aggregator. De er alle instansiert med Docker.

4.4.1 Promtail

Promtail er en agent som videresender innholdet av lokale logger til Grafana Loki (Grafana, u.å.d). Den er instansiert på server og hver klient i pCT systemet som trenger overvåkning. Promtail har primært tre oppgaver:

- Utlese og oppdage loggfiler
- Feste etiketter til logg-strømmer
- Sende dem til Loki-instansen.

Promtail kommer med en YAML fil, promtail-config.yaml (se Figur 9), som er plassert lokalt på maskinen den kjører på. Her spesifiseres instruksjoner for hvordan Promtail skal kjøres. Filen blir lest inn når Docker Containeren med Promtail instansen startes opp.

Oppdage loggfiler

Før Promtail kan sende loggdata til Loki, må den ha informasjon om omgivelsene sine. Dette er spesifisert manuelt i promtail-config filen, i seksjonen scrape_configs. Det innebærer å definere hvilken filplassering, og hvilken filer som skal monitoreres for loggmeldinger. Promtail oppdager automatisk filer i den spesifiserte filplasseringen av den spesifiserte filtypen (f.eks .txt, .log etc). Promtail begynner å scrape nye loggmeldinger som dukker opp i disse filene. Scraping kan forklares som en teknikk i et dataprogram, der data generert av et annet program blir innhentet (Cloudflare, u.å.).

Feste etiketter til logg-strømmer

Mens Promtail oppdager loggfiler som genereres, blir tilhørende metadata bestemt (filnavn, modulinformasjon, etc.). Dette festes på utvalgte loggmeldinger som etiketter (labels) for identifikasjon og sortering når de blir prosessert i Loki. Promtail kan også feste etiketter basert på innholdet i hver logglinje for videre filtrering.

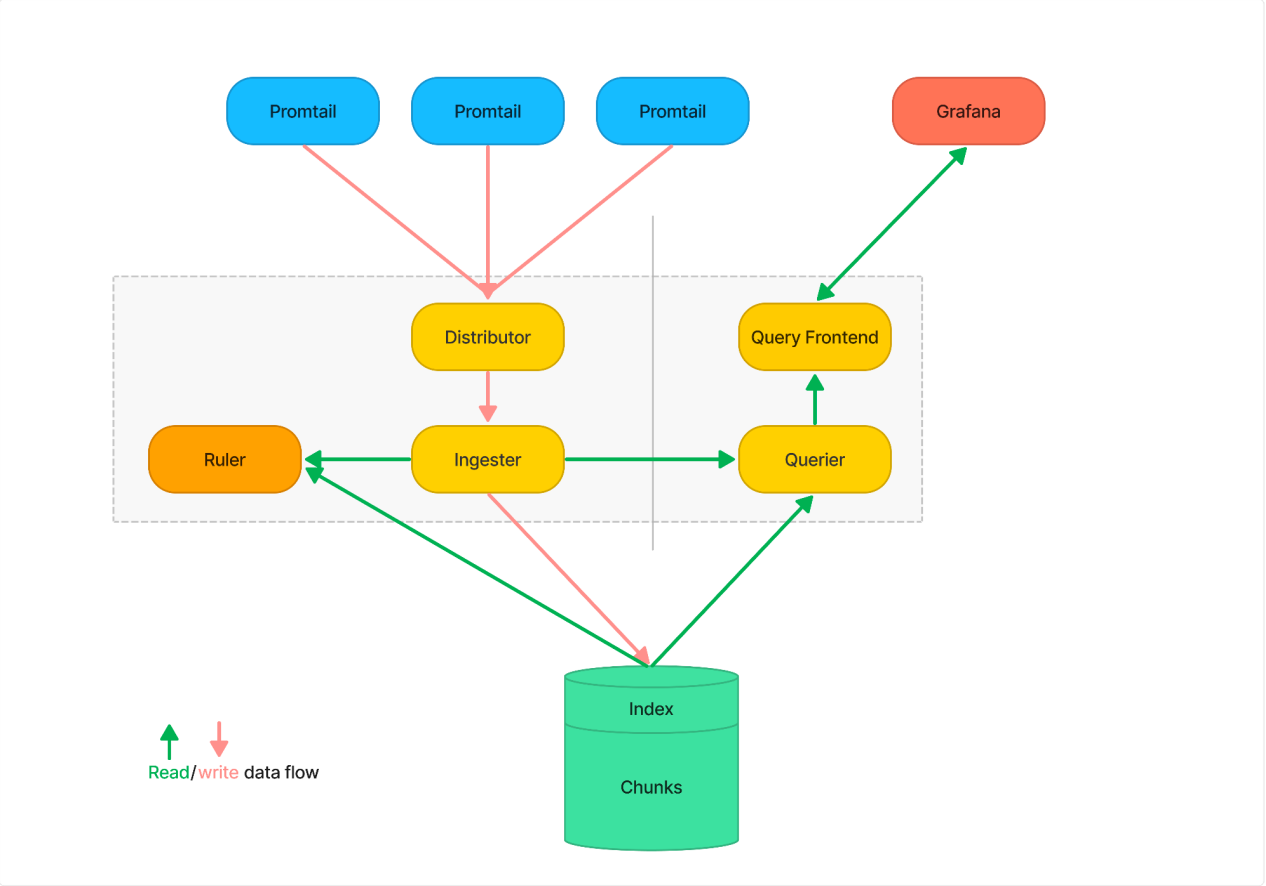
Sende loggdata til Loki

Når Promtail har et sett med loggfiler og alle etiketter er riktig plassert, vil den starte tailing (kontinuerlig lesing) av loggene. Etter nok data er lest til minne eller etter et konfigurerbart tidsavbrudd, blir det sendt som ett parti til Loki sitt API. Deretter vil den fortsette lesingen fra der den stoppet.

```
#Define the ports for promatil. These are at default.
server:
  http_listen_port: 9080
  grpc_listen_port: 0

#Describes where to save read file offset to disk.
#Promtail will save a file indicating how far it has read into a file. It is needed for when Promtail
#is restarted to allow it to continue from where it left off.
positions:
  filename: /tmp/positions.yaml

#Define the url of the Loki instance you want promtail to forward the logs to. You can define multiple instances. You always connect to the Loki push API.
clients:
  - url: http://158.39.77.99:3100/loki/api/v1/push


#The scrape_configs block configures how Promtail can scrape logs from a series of targets using a specified discovery method
scrape_configs:
- job_name: system  #Name of this scrape config in the Promtail UI. This is at default
  static_configs:
  - targets:         # 'targets' can be discluded, it will automatically default to localhost, and
      - localhost    # should ONLY be defined as localhost if you chose to define it.
    labels:
      job: varlogs             #Define the label for a job. All logs retrieved from the '__path__' are labeled with 'varlogs' in this case.#In Grafana you can access all the log messages from this job if you query '{job="varlogs"}'. Every Logstream under this job will retrieve a label containing this job.
      __path__: /var/log/*log  #This is where you define the path to where you want to retrieve the log messages.
```

Figur 9: Konfigurasjonsfilen til Promtail, promtail-config.yaml

4.4.2 Loki



Figur 10: Read/write data flow mellom de interne komponentene i Loki

Grafana Loki er verktøyet som tar seg av all behandling av innkommende loggdata på Loki APIet og all behandling av queries fra brukere. Av instansene i logg-aggregeringsverktøyet er Loki "hjernen" som prosesserer, delegerer, returnerer og innhenter data. Loki er delt inn i flere interne komponenter, med hver sine oppgaver, disse er: Distributor, Ingester, Chunk Store, Query Frontend og Querier (Kumar, 2021).

Distributor

Distributor er det første stoppet i write-pathen for loggdata (illustrert i figur 10)(Grafana, u.å.e). Distributor behandler innkommende logger sendt fra diverse klienter. I den gjeldende løsningen er det flere instanser av Promtail som utgjør disse klientene. Loggstrømmen som mottas fra Promtail valideres først for korrekthet, splittes opp i kategorier og sendes deretter til forskjellige ingesters. Valideringen går ut på å sjekke om all innkommende data oppfyller de satte spesifikasjonene, som at labels er gyldige Loki labels, sjekke at tidsstemplet ikke er for gammelt eller for nytt, og at loggmeldingene ikke er for lange. Kategoriseringen av loggstrømmer, som er et sett med logger knyttet til et unikt label-sett, skjer ved at Distributor benytter det tilknyttede label-settet for å finne hvilken Ingester som skal tilsendes loggstrømmen. En loki instans kan ha

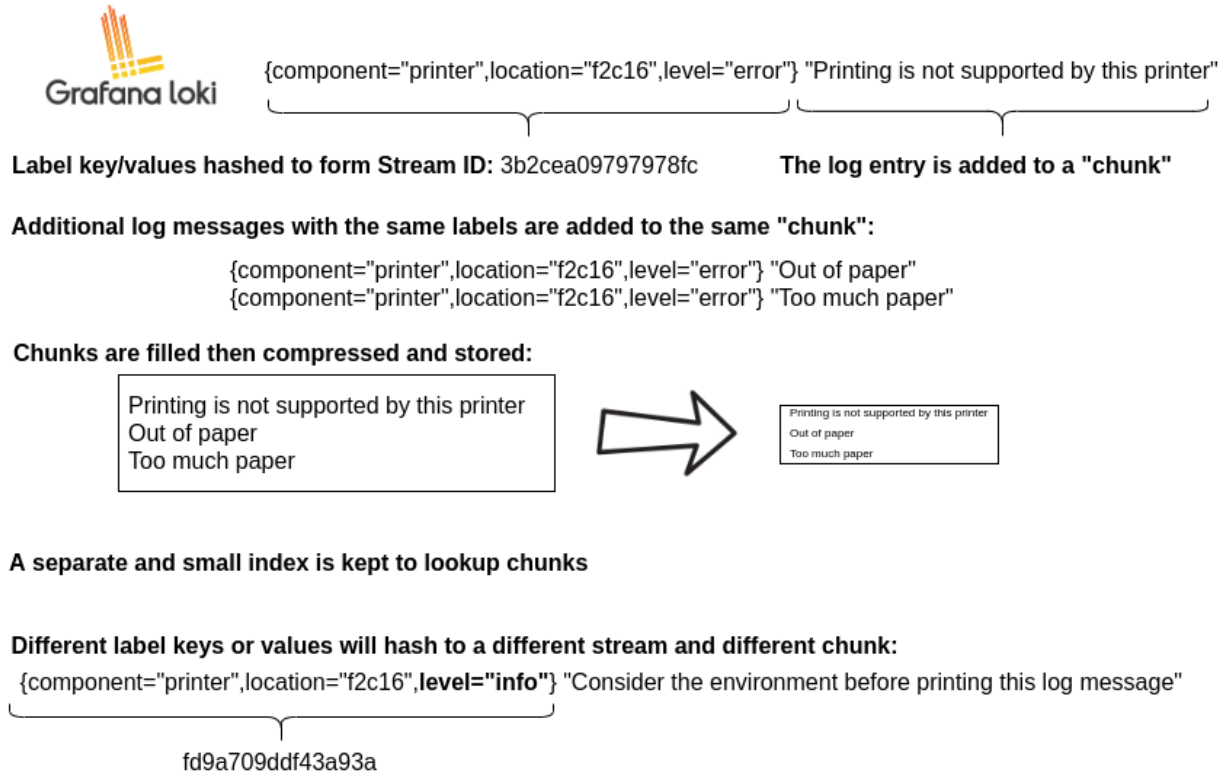
mange Distributors, avhengig av hvor mange enheter (Promtail-instanser i dette tilfellet) som har knyttet seg til push APIet til Loki. Mengden av Distributors er skalerbart, og justeres opp og ned med hensyn på dette. At antallet Distributors kan tilpasses, hjelper for å opprettholde høy nok throughput, slik at Ingesterne aktivt blir tilført data, uten at data forsinkes eller forsvinner i prosessen.

Ingester

Ingester-tjenesten tar seg av skriving av loggdata til lagringsmedium, og returnering av loggdata basert på queries fra lagringsmedium. Ingestoren er dermed knyttet til både write path og read path (illustrert på figur 10) for datatransmisjon i Loki.

Write path

Som nevnt tidligere, er Loki bygget rundt ideen om å kun indeksere metadata om loggmeldinger, med et label (Grafana u.å.f). Dette er det Ingesteren som utfører. Hver loggstrøm Ingesteren mottar fra en Distributor, bygges til et sett av mange "chunks", og lagres midlertidig i minnet. Loggmeldinger fra hvert unike sett med labels, bygges inn i chunks. Det vil si, en chunk består av flere loggmeldinger som deler det samme label-settet. Etter en viss tid, som kan konfigureres, komprimerer Ingesteren chunksene fra minnet og sender de til en database. Finnes en chunk som tilfredstiller label-settet, og som enda ligger i minnet til ingesteren, fortsetter ingesteren å pushe loggmeldinger inn i denne chunken, før den blir komprimert og sendt til databasen.
Ingesteren produserer en indeks basert på loggmeldingens innhold og label-sett. Dette skjer samtidig som loggmeldinger blir plassert i chunks. Indeksen fungerer som en referanse til hvilken chunk en loggmelding befinner seg i databasen. Indeksen i seg selv, blir plassert i en ekstern database, atskilt fra databasen med chunks. I databasekapittelet kommer en grundigere forklaring på dette.



Figur 11: Illustrasjon av hvordan Loki behandler write-data (Grafana, u.å.g)

Read path

På Read path er Query Frontend det første stoppet for innkommende query requests i Loki. Disse query requests blir behandlet og sendt videre til Querier, som igjen sender forespørsel til Ingester. Når Ingester mottar en read-request fra Querier, søker den gjennom det lokale minnet sitt, og returnerer matchende logg-data.

Querier

Querier tjenesten behandler queries mot de interne Loki komponentene, med Grafana Loki sitt eget logg-query språk LogQL. Ved å bruke LogQL innhenter Querier loggmeldinger fra Ingester tjenestene sitt minne, samt fra fra databasen. Query forespørsler kommer fra Querier Frontend. Når Querier har mottatt en forespørsel, sender den først Ingesterene en query, og spør om de har gjeldende logg-data liggende i minnet, hvis ja, returneres logg-dataen. Om Ingesterne ikke har den forespurte logg-dataen i minnet, vender Querier seg mot databasen, og gjør et søk der. All logg-data som er funnet, returneres til Query-Frontend.

Query Frontend

Query Frontend er tjenesten som forsørger Querier sine API endepunkter. Eksterne kilder som ønsker å gjøre en query mot logg-databasen, snakker med query APIet til loki. Der er det Query

Frontend som tar imot de innkommende queries. Her gjøres det noen tilpasninger slik at queryen stemmer med LogQL syntaxen. De innkommende queries blir så lagret i en kø, for å deretter bli delegert til Querier, som gjør selve jobben. For å øke hastigheten på innkommende forespørsler, splittes større queries til flere mindre, og kjøres deretter parallelt på flere Queriers. Query frontend kan beskrives som "sjefen" som tar imot forespørsler, tilpasser de, og delegerer de til forskjellige "arbeidere" (Queriers).
Når en Querier har ferdigbehandlet en query, returnerer den loggdata til Query Frontend, som tilgjengeliggjør loggmeldingene på APIet. I dette prosjektet er det kun Grafana som kommuniserer mot query APIet.

API

Loki eksponerer et HTTP API for pushing, querying og tailing av loggdata (Grafana, u.å.h). Her nevnes de mest vesentlige endepunktene eksponert av de interne komponentene, og Loki selv.

Distributor eksponerer dette endepunktet:

`POST /loki/api/v1/push`

Push endepunktet blir brukt til å sende logg-data til Loki. Dette er det APIet Promtail sender logg-data til.

Query Frontend eksponerer disse endepunktene:

`GET /loki/api/v1/query`

Query endepunktet blir brukt til å gjøre queries på ett enkelt tidspunkt.

`GET /loki/api/v1/query_range`

Query _range endepunktet blir brukt til å gjøre queries over et gitt tidsrom.

`GET /loki/api/v1/tail`

Tail endepunktet blir brukt for å returnere en strøm av loggmeldinger fortløpende, basert på en query.

`GET /loki/api/v1/labels`

Labels endepunktet blir brukt for å innhente en gitt liste av labels, innen et gitt tidsrom.

De overnevnte API endepunktene blir aktivt brukt av Grafana for queries og innhenting av loggmeldinger.

Disse endepunktene er eksponert uavhengig:

`GET /ready`

Ready endepunktet er brukt for å sjekke om Ingesteren er klar for å ta imot data. Den returnerer "Ready" hvis klar.

Oppsett til Loki

Grafana Loki kjører i en dedikert Docker container, som er plassert på serveren. Dockereren startes opp via en ferdig konfigurert Docker Compose fil, sammen med Grafana og Promtail.

Loki-config.yaml filen setter de ønskede konfigurasjonene for Loki, når docker containeren starter opp. Konfigurasjonsfilen er vist i figur 12.

```
#Enables authentication through the X-Scope-OrgID header, which must be present if true.
#If false, the OrgID will always be set to "fake".
auth_enabled: false

#Defines the listening ports for Loki.
server:
  http_listen_port: 3100
  grpc_listen_port: 9096

#Sets common definitions to be shared by different components. This way, one doesn't have to replicate configs in multiple places.
common:
  path_prefix: /tmp/loki  #When this is defined this will be present in front of the endpoint paths.
  storage:
    filesystem:  #Configures storing the chunks on the local filesystem. Required fields only required when filesystem is present in config.
      rules_directory: /tmp/loki/rules
  replication_factor: 1  #How many times incoming data should be replicated to the ingester component.
  ring:  #A common ring configuration to be used by all Loki rings.
    instance_addr: 127.0.0.1 #IP address to advertise in the ring.
    kvstore:  #The key-value store used to share the hash ring across multiple instances.
      store: inmemory  #Backend storage to use for the ring. Supported values are; consul, etcd, inmemory, memberlist, multi.

#Configures the storage solution for the index store and the chunk store. These are seperate stores, unless defined different.
#Can be configured to use database storage, local file storage, or a combination
schema_config:
  configs:
    - from: 2021-04-26
      store: aws     #This is the store used for indexes. Indexes are used to access the chunk storage in the right storage space.
      object_store: s3  #This is the store used for chunks. Chunks are compressed log data.
      schema: v11
      index:
        prefix: loki_index
        period: 24h       #How long to store the indexes before deletion.

#This part is to tell Loki how and where to store indexes and chunks
storage_config:
  aws:
    s3: s3://********************** #Chunk storage access key for Amazon S3.
    dynamodb:
      dynamodb_url: dynamodb://****************** #Index storage access key for Amazon DynamoDB.
```

```
#Data transmission settings for database
table_manager:
  chunk_tables_provisioning:
    inactive_read_throughput: 1
    inactive_write_throughput: 1
    provisioned_read_throughput: 5
    provisioned_write_throughput: 5
  index_tables_provisioning:
    inactive_read_throughput: 1
    inactive_write_throughput: 1
    provisioned_read_throughput: 5
    provisioned_write_throughput: 5
  retention_deletes_enabled: false
  retention_period: 48h

#Ingester settings
ingester:
  chunk_idle_period: 5m #Define the idle period before a chunk is compressed and flushed to the database.
  chunk_retain_period: 30s #Define how long after the chunk is flushed, the chunk should be retained in memory.

ruler:
  alertmanager_url: http://localhost:9093 #Comma-separated list of Alertmanager URLs to send notifications to. Each URL is treated as a separate group in the configuration.
```

Figur 12: Konfigurasjonsfilen til Loki, loki-config.yaml

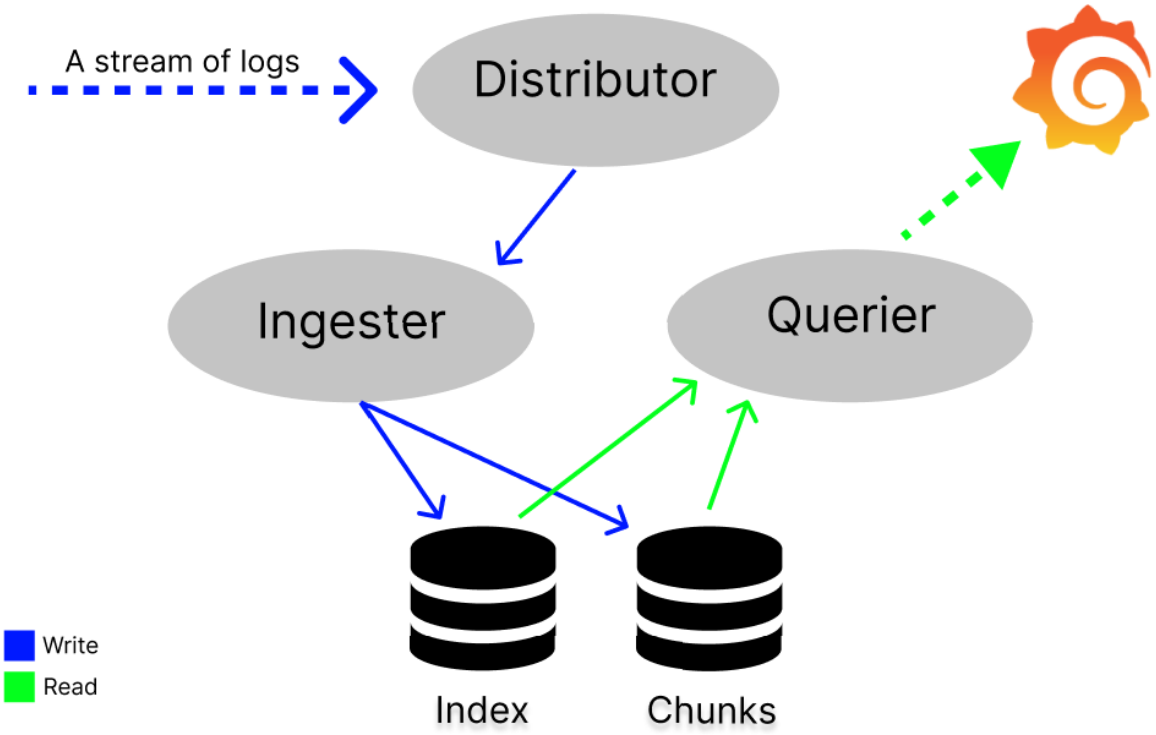
4.4.3 Grafana

Grafana er en åpen kildekode-plattform for monitorering og datavisualisering. Det er en webapplikasjon som omgjør kompleks data til visuelle grafer, diagrammer eller lignende. Hensikten er å få brukerne i stand til å forstå sammenhenger og mønstre i store datamengder. Et dashbord i applikasjonen tillater brukere å velge hvordan de ønsker loggdataen visualisert. Dashbordet kan ha mange paneler for å analysere flere logginstanser samtidig.

For å utforske loggdata som har blitt sentralisert på serveren, må Loki-instansen legges til som en datakilde i Grafana. Det gjøres ved å spesifisere en URL med IP-adresse til serveren og portnummeret til Loki (http://158.39.77.99:3100). Kun brukere med admin-rolle kan legge til datakilder.

Grafana interagerer med brukerne via portnummer 3000. Gjennom applikasjonen kan brukeren sende queries til Loki og få resultatet returnert i form av loggmeldinger og visuelle grafer. All monitorering av pCT kan gjøres i Grafana. Siden dette er brukergrensesnittet i løsningen, vil Grafana bli nærmere forklart fra brukerens perspektiv i kapittel 4.6.

4.5 Databaseløsning for langtidslagring



Figur 13: Illustrasjon av hvordan loggmeldingene blir lagt inn i databasen

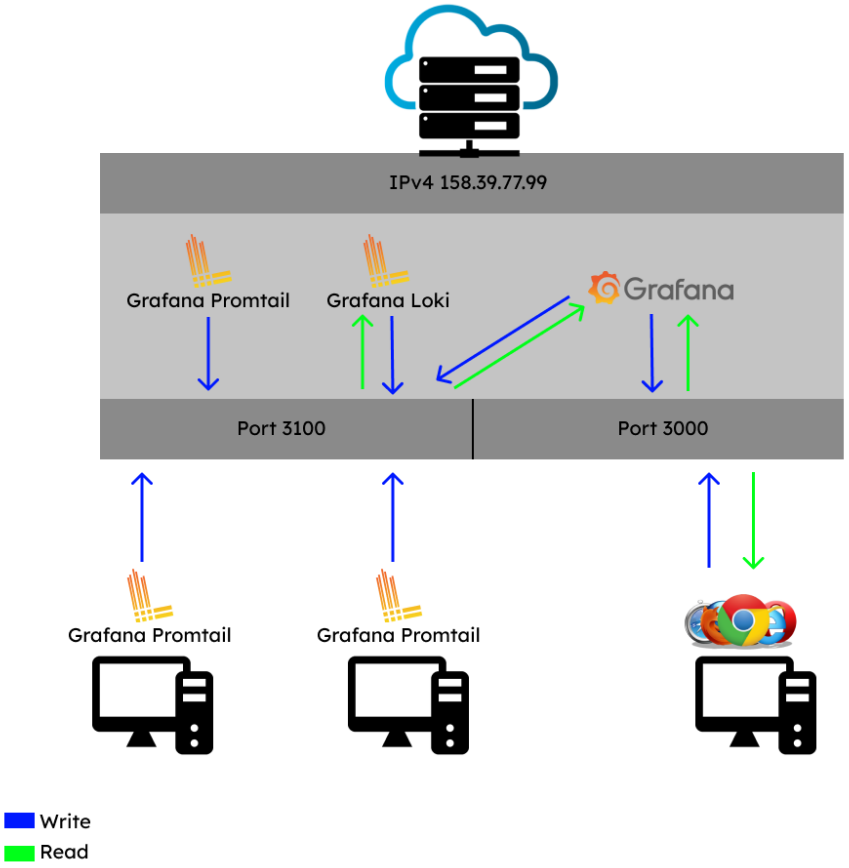
Grafana Loki lagrer loggmeldinger i form av indekser og chunks. Loki tar inn loggmeldingene i separate streams der hvert stream blir tildelt en stream ID og et sett med loggmeldinger. Disse streamene blir komprimert og lagret som chunk-objekter. En label blir opprettet for den chunken og lagret som en indeks. Det er Lokis ingester som er ansvarlig for pushingen av indeksene og chunkene (Grafana, u.å.b).

For langtidslagring ble Amazons DynamoDB og S3 valgt for lagring av henholdsvis indekser og chunks. Begge er skybaserte tjenester som Grafana selv har anbefalt til bruk for langtidslagring. Amazon S3 ble brukt ettersom den er egnet for å lagre stor og ustrukturert data (som et sett med komprimerte loggmeldinger). Databasene blir knyttet mot Loki instansen i serveren via tilgangsnøkler for hver tjeneste inne i konfigurasjonsfilen til Loki (se figur 14). Av sikkerhetsmessige grunner har tilgangsnøklene blitt sensurert.

```
storage_config:
  aws:
    s3: s3://****************************************
    dynamodb:
      dynamodb_url: dynamodb://****************************************
```

Figur 14: Delen av konfigurasjonsfilen til Loki som kobler opp mot skytjenestene

4.6 Server og det distribuerte systemet



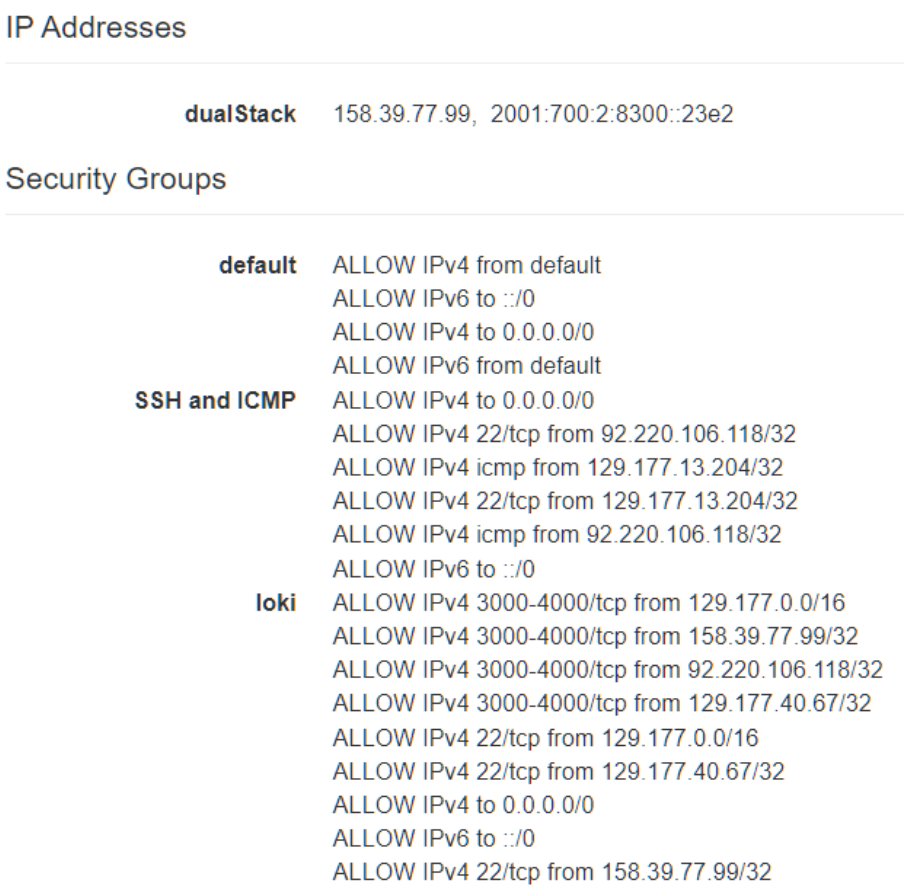
Figur 15: Kommunikasjon mellom instanser gjennom portnumre på server

For testing og kommunikasjon på tvers av instanser i det distribuerte systemet er det satt opp en virtuell maskin kjørende på en NREC server. NREC tilbyr infrastruktur som tjeneste (IaaS), også kalt skyinfrastruktur (NREC, u.å.a). NREC står for nettverket, serveren, lagringen og virtualiseringen, men gir brukeren tilgang til, samt håndteringen, av disse instansene via skyen. Server tjenesten med den virtuelle maskinen gruppen har benyttet, settes opp i NREC sitt dashbord, på deres nettside. Her settes det også opp sikkerhetsgrupper og SSH-nøkkel godkjenninger for å begrense tilgang.

Den virtuelle maskinen kjører i et Linux miljø på Ubuntu versjon 20.04, som ble installert på serveren via et standard Ubuntu ISO image. Tilgang for kontroll og styring av den ferdig konfigurerte serveren opprettes gjennom Linux terminalen (CLI). Tilgang gis kun hvis IP adressen er godkjent i en Security Group, samt at det er opprettet et SSH-nøkkelpar. Hvis det kobles på gjennom et offentlig nettverk, som HVL eller UiB sine skole-nett, oppdager NREC dette, og det kreves en UiB innlogging i tillegg, som en ekstra sikkerhet.

En sikkerhetsgruppe inneholder regler som definerer hvilken trafikk som er lov inn (ingress) og ut (egress) fra den virtuelle maskinen på serveren. Det følger med en standard sikkerhetsgruppe, men det kreves oppretting av noen ekstra sikkerhetsgrupper for at alle instanser i

loggføringsverktøyet skal kunne kommunisere med serveren. "SSH and ICMP" og "loki" er sikkerhetsgrupper som er definert utenom standard gruppen "default".



Figur 16: Sikkerhetsgrupper for serveren med IPv4 158.39.77.99

NREC serveren har en IPv4 adresse på 158.39.77.99 og portene i intervallet 3000 til 4000 står åpne for innkommende trafikk fra godkjente IP adresser. Her trengs det ikke et SSH nøkkelpar. De godkjente IP adressene kan også lese informasjon fra klienter koblet opp på portnumre i intervallet 3000-4000.

På den virtuelle maskinen som kjører på serveren er det installert de mest grunnleggende komponentene som kreves for at Loki, Grafana og Promtail skal kunne kjøre. Dette innebærer Docker og Docker Compose samt nødvendige Ubuntu systemoppdateringer.

Promtail, Loki og Grafana er installert i hver sin Docker på serveren. Kommunikasjon via server er illustrert i figur 15. Det distribuerte systemet kommuniserer via serveren på IPv4 adresse 158.39.77.99. Serveren kjører en Promtail instans, som sender loggmeldinger til Grafana Loki sitt push API på port 3100 internt på serveren. Serveren hoster Grafana Loki og Grafana. Grafana Loki kjører på port 3100 og leser alt av loggmeldinger som kommer inn på APIet, både fra sin interne instans av Promtail og alle Promtail instanser kjørende på klienter koblet til serveren. Loki

prosesserer disse loggmeldingene og pusher de ut igjen på get-APIet på port 3100, som respons til query forespørsler.

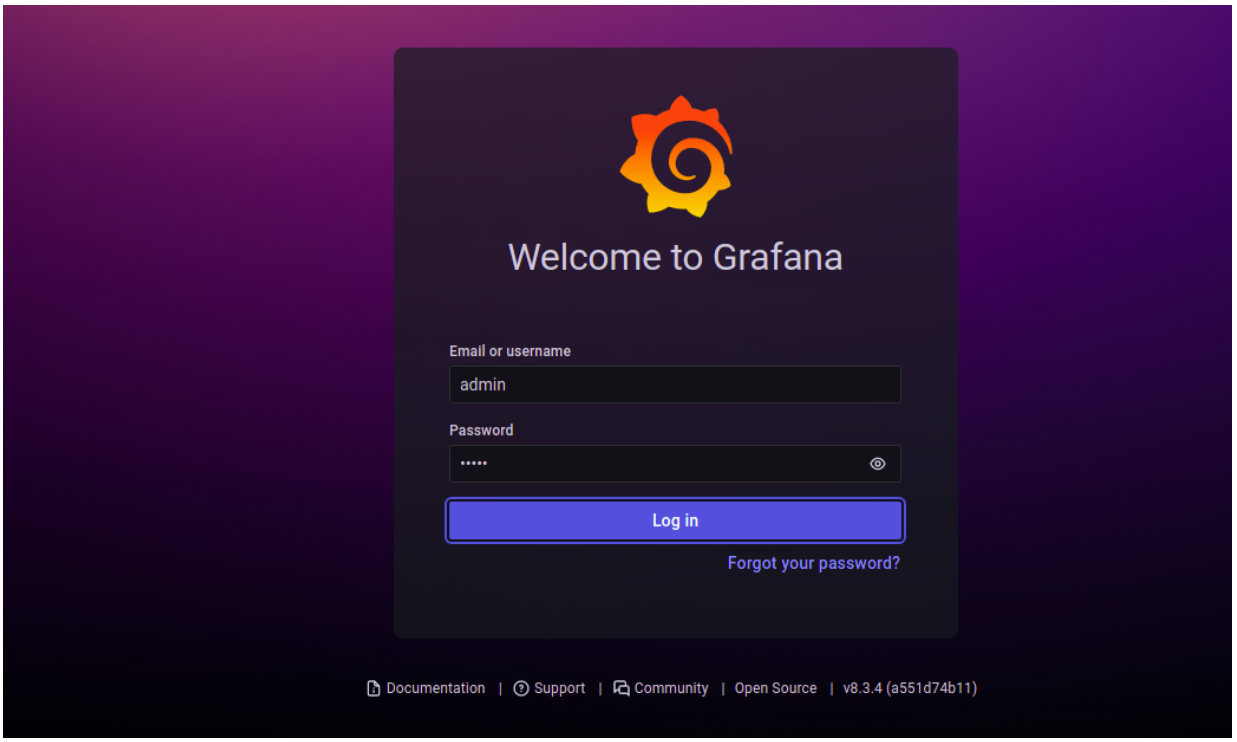
Grafana kjører på port 3000 på serveren. Grafana innhenter de ferdig prosesserte loggmeldingene fra Loki sitt API på port 3100, gjennom queries. Eksterne datamaskiner kan koble til serveren på port 3000 og visualisere dataene i Grafana, samt sende queries i grafana sitt brukergrensesnitt, som videreformidler disse queryene til Loki på port 3100.

4.6 Brukergrensesnitt

Brukergrensesnittet foregår i webapplikasjonen Grafana. De viktigste interaksjonene vil bli introdusert nedenfor.

4.6.1 Innlogging

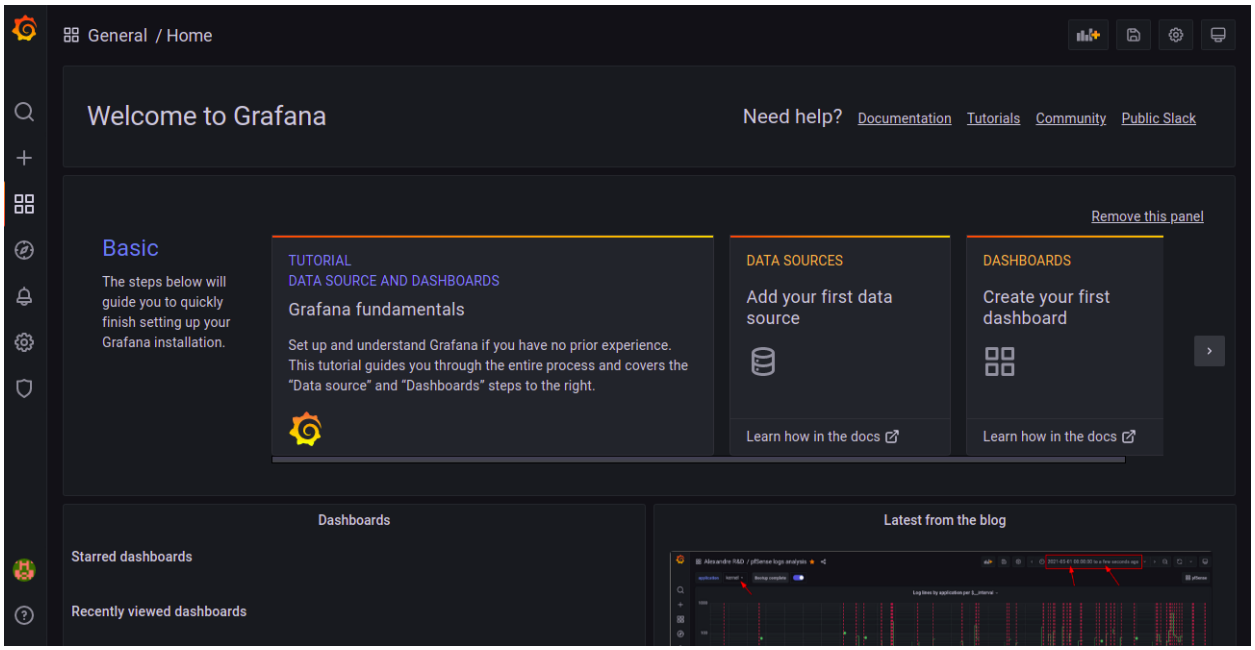
En bruker som vil monitorere systemet møter en innloggingsside. Brukeren logger seg enten inn som admin eller som vanlig bruker.



Figur 17: Grafana innlogging

4.6.2 Hjem

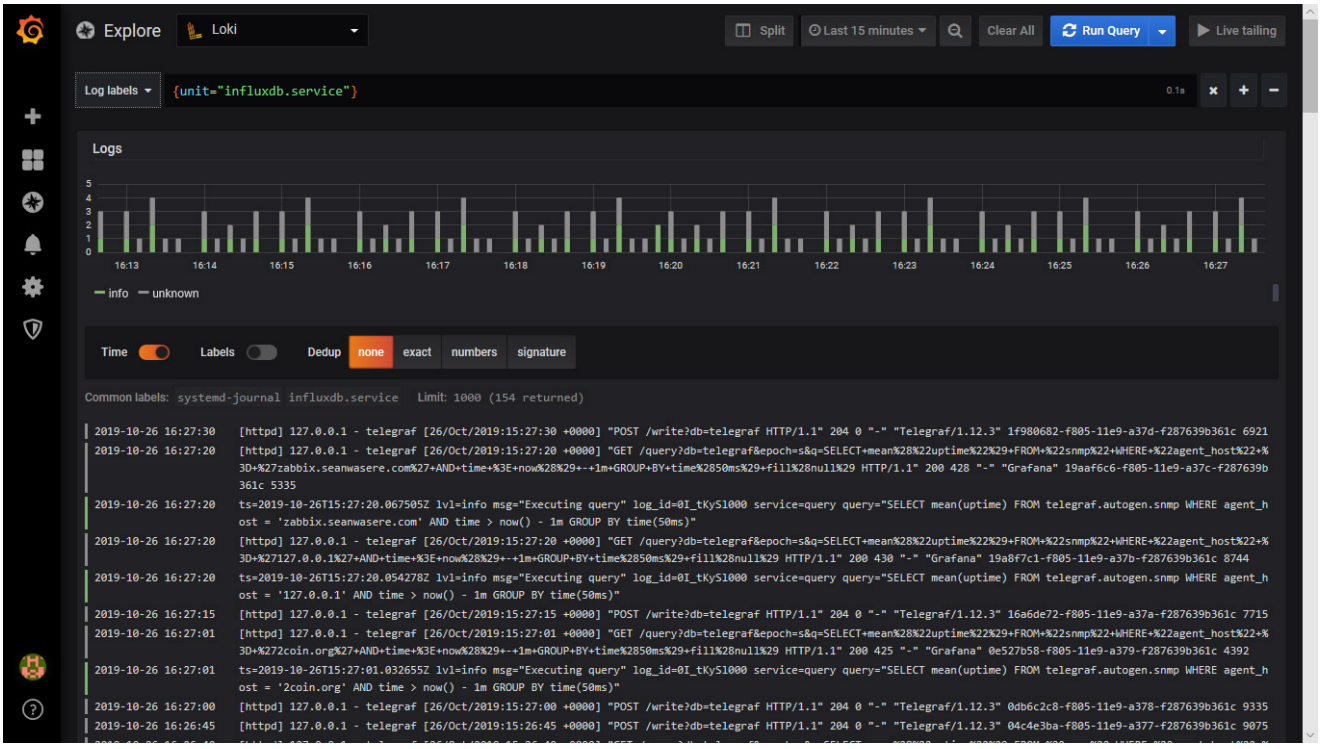
Etter en vellykket innlogging kommer brukeren til hjemmesiden. Denne siden kan tilpasses slik at den passer brukerens behov, ved å for eksempel implementere et dashbord. Grensesnittet har en meny med ulike sider og funksjoner man benytte seg av.



Figur 18: Grafana hjemmeside

4.6.3 Explore

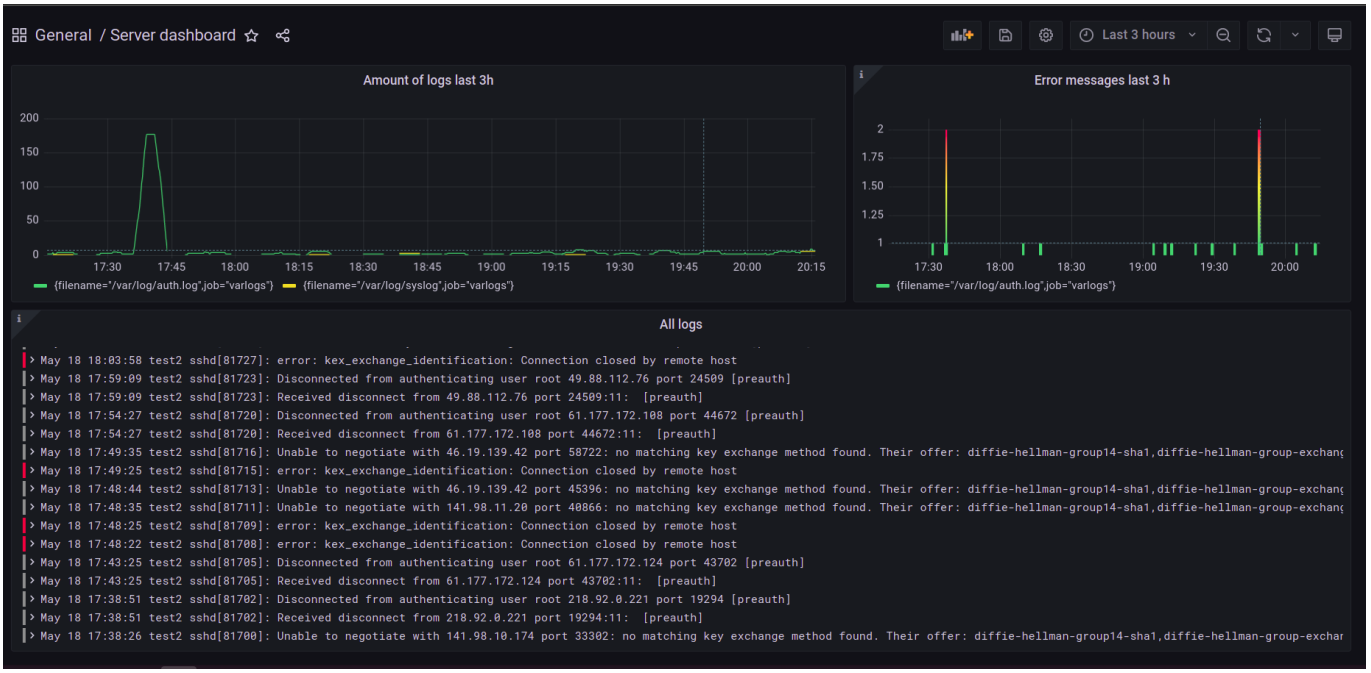
På denne siden kan brukeren sende queries ved å benytte datakilden Loki. Explore-siden har ikke noe dashbord eller panel, bruken er forbehold til å sende queries og til å utforske dataen. Resultatet av logg-queries blir vist som histogrammer og individuelle loggmeldinger. Det er Explore-siden som besitter funksjoner for søk og filtrering. Filtrering skjer enten ved å filtrere etikettene som er festet til logginstanser eller gjennom queries. Søking skjer gjennom queries.



Figur 19: Grafana Explore

4.6.4 Dashboards

Loggdata som skal monitoreres vises i et brukerdefinert dashbord som kan endres ved behov. Et dashbord er et sett av en eller flere paneler organisert i en eller flere rader. Det er her dataen blir visualisert ved hjelp av grafiske verktøy fra Grafana. De mange ulike panelene gjør det lett å konstruere riktige queries, og tilpasse visualiseringen til egnet behov. Hvert panel i dashbordet interagerer med data fra Loki gjennom queries.



Figur 20: Grafana Dashboard

4.7 Demo

For å demonstrere hvordan verktøyet kan fungere i praksis, ble det utviklet et demoprosjekt. Siden gruppen hadde planlagt å bruke MongoDB som database på dette tidspunktet, ble dette brukt som databaseløsning i demoprosjektet. Prosjektet besto av et Java-program med to samkjørende tråder. Den ene tråden genererte tilfeldige loggmeldinger og skrev dem til en fil. Den andre tråden leste loggmeldingene fra den samme filen og prosesserte dem før de ble lagt inn i en lokal database via MongoDB APIet. Tråden delte i tillegg meldingene inn i attributter som dato, klokkeslett, kilde, type og alvorlighetsgrad for å gjøre det lettere å sortere meldingene i databasen.

4.8 Prototyper

Logg-aggregeringsverktøyet kunne ikke testes mot modulene i pCT før slutten av prosjektperioden. Dermed var det av høy prioritet å få en prototype som fungerte så nære sluttproduktet som mulig. Prototypene beskrevet nedenfor ble laget i iterasjoner og introduserte en ny komponent i hver iterasjon.

Prototype 1.0

Den første prototypen benyttet logg-generatoren i demoprosjektet beskrevet i avsnitt 4.7. Målet var å få loggmeldingene vist i grensesnittet til Grafana ved bruk av Promtail og Loki. Prototypen bestod av:

- Et program som genererte loggmeldinger og skrev dem til en fil
- Promtail som ekstraherte loggmeldingene og sendte de til Loki sitt API
- Loki som prosesserte loggmeldingene og lagret de lokalt
- Grafana som sendte queries til Loki for å få loggmeldingene vist i et brukergrensesnitt

Grafana, Loki og Promtail kjørte i hver sine Docker containere på en datamaskin, og ble instansiert av Docker Compose. Denne maskinen opererte som både server og klient i det distribuerte systemet. Konfigurasjonsfilene til Loki og Promtail ble plassert lokalt på samme maskin. I Promtail konfigurasjonsfilen ble det spesifisert at loggmeldingene skulle utleses fra filplasseringen der logg-generatoren produserte loggmeldinger. Det ble også definert hvilken Loki instans den skulle sende til. Loki komprimerte og fordelte loggmeldingene i indekser og chunks før de ble lagret lokalt på maskinen.

Da kommunikasjonen ble opprettet, kunne loggmeldingene vises i terminalen og systemets metrics kunne vises på http://localhost:3100/metrics. Metrics brukes til å monitorere systemets ytelse og gjenkjenne viktige hendelser for gitte tidsintervaller (Berman, 2017). En adminbruker ble opprettet i Grafana og Loki ble lagt til som datakilde i applikasjonen. En ekstern bruker på en annen datamaskin kunne deretter sende queries og få vist loggmeldingene i grensesnittet til Grafana.

Figur 21: Prototype 1.0 vist fra brukergrensesnittet

Prototype 2.0

I neste iterasjon var målet å lage et distribuert system med en servermaskin som håndterte loggmeldingene og en node som genererte loggmeldingene og sendte de til serveren. I tillegg ble det opprettet en skybasert server som ble en del av sluttproduktet. Maskinen som genererte loggmeldinger (klienten), sendte det til Loki-instansen på servermaskinen ved å spesifisere IP-adressen til serveren og portnummeret til Loki i konfigurasjonsfilen til Promtail. Ved å installere Docker imaget til logg-generatoren og Promtail på en maskin, kunne systemet legge til flere noder. Prototypen var vellykket og målet ble nådd.

Prototype 3.0

I siste prototype skulle verktøyet fungere med alle komponenter korrekt installert og med en fungerende kommunikasjon mellom instansene. Prototypen bygget videre på de tidligere versjonene, men introduserte databaseløsningen som brukes i sluttproduktet. I stedet for at Loki lagret loggmeldingene lokalt, ble indekser og chunks fordelt i to skybaserte tjenester, DynamoDB og S3. Prototypen fungerte slik den skulle, og ble utgangspunktet for sluttproduktet. Det eneste som manglet var å bytte ut nodene som genererte testloggmeldinger med nodene i pCT.

5 RESULTATER

5.1 Evalueringsmetode

Som nevnt i delkapittel 3.5 er evalueringsmetodene som ble brukt integrasjonstesting, systemtesting og brukertesting. Disse metodene ble valgt for å verifisere at systemet fungerte som forventet og for å validere produktideen.

Integrasjonstester ble gjort på ulike deler av systemet under utviklingen. De handlet om å få to eller flere komponenter i systemet å fungere sammen. Hver integrasjon ble verifisert gjennom testing og bygging av systemet på gruppens GitLab repository. Dette er en utviklingspraksis tilnærmet Continuous Integration, men tar ikke i bruk automatisk bygging og testing av koden (Amazon, u.å.c). Demoprosjektet beskrevet i delkapittel 4.7 var med på alle testene siden det var disse loggmeldingene som skulle sendes mellom komponentene i systemet. En vellykket test ble gjort da loggmeldingene kom frem dit de skulle.

Etter integrasjonstestene ble det gjennomført en systemtest. Det var viktig å gjøre testene i denne rekkefølgen slik at systemtesten kunne avdekke feil som integrasjonstestene ikke fanget opp. Testen gikk ut på å kjøre hele systemet for å verifisere at alle komponenter fungerte feilfritt sammen.

For å sikre at produktet stod til oppdragsgivers forventninger, ble brukertesting gjennomført i to iterasjoner. Grensesnittet ble evaluert i første iterasjon mens hele systemet ble evaluert i andre iterasjon. De tok sted på Instituttet for fysikk og teknologi ved UiB og ble gjennomført av oppdragsgiver og andre deltakere i pCT prosjektet. Det som ble evaluert var brukervennlighet og funksjonalitet.

I første iterasjon ble prototype 3.0 testet. Testpersonene skulle via brukergrensesnittet visualisere loggmeldingene generert av logg-generatoren fra demoprosjektet. Ved å bruke søk og filtrering skulle brukerne hente loggmeldingene fra databasen og se utskriften samt systemoppførsel ved hjelp av dashbordet i grensesnittet. Siden ingen av testpersonene visste hvordan loggene skulle ekstraheres, måtte gruppen demonstrere hvordan queries skulle settes opp. Etter testgjennomgang ble prototypen evaluert ved at testpersonene ga tilbakemelding på programmets funksjonalitet og brukervennlighet.

I andre iterasjon ble produktet testet med reelle loggmeldinger fra det distribuerte systemet pCT. Dette ble gjennomført av oppdragsgiver ved at han lastet ned de nødvendige pakkene fra gruppens repository på GitLab, og distribuerte dem til server og klienter. Da systemet var satt opp, testet han systemet fra brukeren sitt perspektiv på en ekstern datamaskin. Formålet med denne brukertesten var å evaluere funksjonalitet og brukervennlighet til systemet når det hentet inn loggmeldinger fra modulene i pCT. Da brukertesten var gjennomført ga oppdragsgiver en tilbakemelding på testen. I tillegg hadde gruppen forberedt et evalueringsskjema med påstander

om brukervennlighet og funksjonalitet som skulle besvares. Hver påstand skulle besvares med et tall mellom 1 og 5 som insinuerte hvor enig man var i påstanden.

5.2 Evalueringsresultat

Resultatene fra testmetodene ble evaluert og tatt i betraktning for videre utvikling av systemet. Evalueringen bestod av utfallet fra integrasjonstester, en systemtest, samt respons fra personene som utførte brukertestene. Besvarelsen fra evalueringsskjemaet ble et godt grunnlag for å bedømme i hvilken grad produktet er riktig utviklet.

Integrasjonstester

Følgende er testresultatene av integrasjonstester gjort i prosjektet. For ordens skyld er logg-generatoren fra demoprosjektet med på alle tester, men vil ikke bli nevnt her.

Test	Komponenter som testes	Formål med testen	Resultat i iterasjoner
1	Promtail og Loki	Få Promtail til å sende loggmeldinger til Loki, og visualisere loggmeldingene i terminalen.	1) Vellykket test
2	Promtail, Loki og Grafana	Utvide test 1 til at Grafana henter loggmeldinger fra Loki og viser dem i grensesnittet.	1) Utfordringer med kommunikasjonen mellom Loki og Grafana 2) Vellykket test. Løst ved å endre konfigurasjonsfilen til Loki og datakildeinnstillinger i Grafana
3	Promtail, Loki, Grafana og NREC serveren	Utvide test 2 til at Loki og Grafana kjører på NREC i stedet for lokalt på maskinen.	1) Problemer med forbindelse til server 2) Feil i Sikkerhetsnøkler 3) Vellykket test. Forbindelsesproblemet løste seg med korrigering i sikkerhetsnøkler.
4	Promtail, Loki, Grafana og databaseløsningen	Utvide test 3 til at Loki benytter DynamoDB og S3 for lagring av loggmeldinger i stedet for lokal lagring.	1) Indekser ville ikke bli lagt inn i DynamoDB. 2) Vellykket test. Var feil med konfigurasjonen av indekser i konfigurasjonsfilen til Loki. Funket etter dette ble endret.

Tabell 1: Integrasjonstester

Test 1: Første steg for å sentralisere loggmeldingene. Promtail henter loggmeldinger fra en modul og sender de til Loki for delvis håndtering. Presentasjonen av loggene er begrenset da de kun blir vist i terminalen som rullende tekst.

Test 2: Systemet får et brukergrensesnitt og kan presentere loggene i ønsket format ved hjelp av visuelle verktøy i Grafana. Loki er lagt til som en datakilde i Grafana og blir komponenten som håndterer alle logger etter brukerens behov.

Test 3: Systemet går fra å være et lokalt system på én maskin til en nettbasert løsning, tilgjengelig for alle brukere med tilgang.

Test 4: Systemet kan nå langtidslagre loggmeldinger i skybaserte databaser.

Ved å opprette flere instanser av Promtail med lik konfigurasjon på flere maskiner, danner det et distribuert system som samler, håndterer og presenterer hendelseslogger.

Systemtest

En test ble gjort av hele systemet da alle komponenter var plass. Testen var vellykket om samspillet mellom alle komponenter fungerte som forventet. Slik var systemet satt opp:

- To datamaskiner (noder) som begge kjørte to Docker containere
  - Demoprosjektet som genererte loggmeldinger
  - Promtail med tilhørende konfigurasjonsfil
- En NREC server som kjørte tre Docker containere via Docker Compose
  - Promtail med tilhørende konfigurasjonsfil
  - Loki med tilhørende konfigurasjonsfil
  - Grafana
- En ekstern datamaskin med tilgang til Grafana (brukergrensesnitt)
- En todelt database, Amazons DynamoDB og S3, som ble håndtert av Loki

Det distribuerte systemet førte logger fra begge nodene til databaseinstansene via Promtail og Loki. Sentraliseringen var gjennomført. Loggmeldingene ble på veien komprimert til indekser og chunks. I Grafana ble en query sendt til Loki, som behandlet forespørselen ved å hente relevante logger fra databasen. De innhentede loggmeldingene ble presentert i brukergrensesnittet via visuelle verktøy i Grafana.

Testen resulterte i at alle komponenter samhandlet riktig og førte loggmeldingene dit de skulle. Det var ingen avvik som ble oppdaget. Resultatet bidrar til å løse problembeskrivelsen og oppnå målene som nevnes i kapittel 1.

Brukertest 1

Etter brukertest av prototypen ga testpersonene tilbakemeldinger som ble brukt for evaluering.

Dette er et kort sammendrag:

Brukerne i teamet var fornøyd med brukergrensesnittet i Grafana. De var tilfreds med løsningene for dashbordet og hvor fleksibelt det var å tilpasse dette. De var litt skeptiske til det første møte med queries, grunnet at det var nytt og forvirrende. Etter en demonstrasjon fra gruppen ble det klart hvordan queries skulle settes opp. Søk virket oversiktlig for teamet etter demonstrasjonen. Filtrering var noe testpersonene tok lett og ytret at de var fornøyd med brukervennligheten.

Brukertest 2

Det var essensielt at oppdragsgiver utførte denne brukertesten siden han skulle implementere systemet og var hovedansvarlig for pCT prosjektet. Han hadde kun én anledning til dette i slutten av prosjektfasen grunnet at han ikke var i Norge. Dette gjorde at gruppen fikk minimalt med tid til å forbedre systemet i ettertid. Brukertesten blir derfor et mål for hvor godt systemet er utviklet.

Resultatet av systemet ble evaluert gjennom en tilbakemelding på produktet samt besvarelsen av evalueringsskjemaet. Tilbakemeldingen er et sammendrag på engelsk og legges til som en del av prosjektbeskrivelsen i pCT sitt delte filsystem på GitLab. I tillegg har oppdragsgiver kommentert noen av påstandene.

Tabell 2 viser evalueringsskjemaet med besvarelse. Påstand nummer 4 og 9 ble ikke besvart, men begrunnes i kommentarene som etterfølges.

Evalueringsskjemaet besvares med tall mellom 1 og 5 (1 er “helt uenig” og 5 er “helt enig”)

Nr.	Påstand	Svar
	Installasjon	
1	Installasjonsinstruksene var tydelige og lett å følge.	5
2	Prosessen med å laste ned pakkene fra GitLab gikk fint.	5
3	Koden i konfigurasjonsfilene var godt dokumentert og lett å gjøre tilpasninger i.	5
4	Det måtte gjøres ekstra tilpasninger utenom det som var påkrevd for at systemet skulle fungere.	
5	Jeg støtte på problemer ved installasjon.	1
6	Det var vanskelig å sette opp det distribuerte systemet.	2
	UI	
7	Navigasjon i Grafana var oversiktlig og det var lett å finne frem.	5
8	Søk og filtrering av loggmeldinger i Grafana var lett å gjennomføre.	5
9	Det var tungvint å sette opp et dashbord.	
10	Jeg synes visualiseringen av loggmeldingene gjorde det oversiktlig å monitorere systemet.	5
11	Jeg var fornøyd med brukervennligheten i grensesnittet.	5
12	Jeg var fornøyd med funksjonaliteten i grensesnittet.	5
	Generelt	
13	Jeg tror en ny bruker vil lære å bruke systemet raskt.	5
14	Jeg ser potensial for videreutvikling av systemet.	5

Tabell 2: Evalueringsskjema for brukertest 2

Matthias Richter, oppdragsgiver, kommenterte følgende av påstandene:

Nr. 2:

Serverpakken har blitt nedlastet som en del av prosjekt repository men har ikke blitt testet ordentlig med Docker Compose.

Nr. 4:

Ikke besvart.

Dette er relatert til at Docker compose ikke var i bruk på PCen. Docker imaget ble kjørt ved 'docker run' kommandoen:

```
docker run -it --rm -v $HOME/pct/logs:/pct/logs -v
$PWD/promtail:/etc/promtail --net=host grafana/promtail:2.4.1
```

Selv om dette vil kjøre containeren med standard parametere, vil Promtail søke i /etc/promtail etter config.yml, en soft link har blitt opprettet i mappen: config.yml -> promtail-config.yaml

Nr. 6:

Det oppsto et problem med Promtail som ikke fant noen loggfiler i den konfigurerte filplasseringen. Dette viste seg å være en brukerfeil ved innskriving i command line argumentet for å starte Promtail Docker containeren. På grunn av denne feilen henviste containeren til en filplassering uten noen loggfiler. Derfor ble ingen loggmeldinger sendt til serveren. Etter litt debugging ble feilen funnet.

Nr. 9:

Ikke besvart. Ikke gjort i dette systemet enda, men Grafana dashboards har blir satt opp veldig lett i andre instanser.

Nr. 11 & 12:

Ikke testet i detaljnivå

Nr. 14:

Det virker som både klient- og serverpakken tilbyr den nødvendige funksjonaliteten og en enkel og rett fram installasjon. Det meste av videreutviklingen vil være å sette opp dashbord for forskjellige subsystemer. Dette er forventet å være en standard oppgave i Grafana systemet. Systemet er veldig fleksibelt og vil tillate å koble til en passende database backend.

Tilbakemelding fra brukertest:

> Summary from commissioning test of central logging system for the Bergen pCT project
>
> Setup: The data readout client and processing prototype is one of the subsystems of the pCT project. It is running on a dedicated PC with connection to the hardware. This setup is located at the Microelectronics lab at IFT/UiB. On the PC, a docker environment without the 'compose' functionality is installed.
>
> On this PC, the client package has been installed and configured to scrape the logs of the data readout client.
>
> On the server side, the central server provided on NREC has been used. The server package has not yet been tested standalone. The logger utility has been tested in a distributed system with the local lab PC and the cloud based server.
>
> Log message were successfully pushed to the server and displayed in the Grafana dashboard. (Richter, 2022)

I testen ble ikke Docker Compose benyttet grunnet at det ikke var installert på maskinen som ble brukt. Det førte til flere tilpasninger enn nødvendig i implementasjonen, men påvirket ikke testresultatene. Resultatet av brukertesten huket av alle målsettingene for prosjektet, nevnt i kapittel 1.

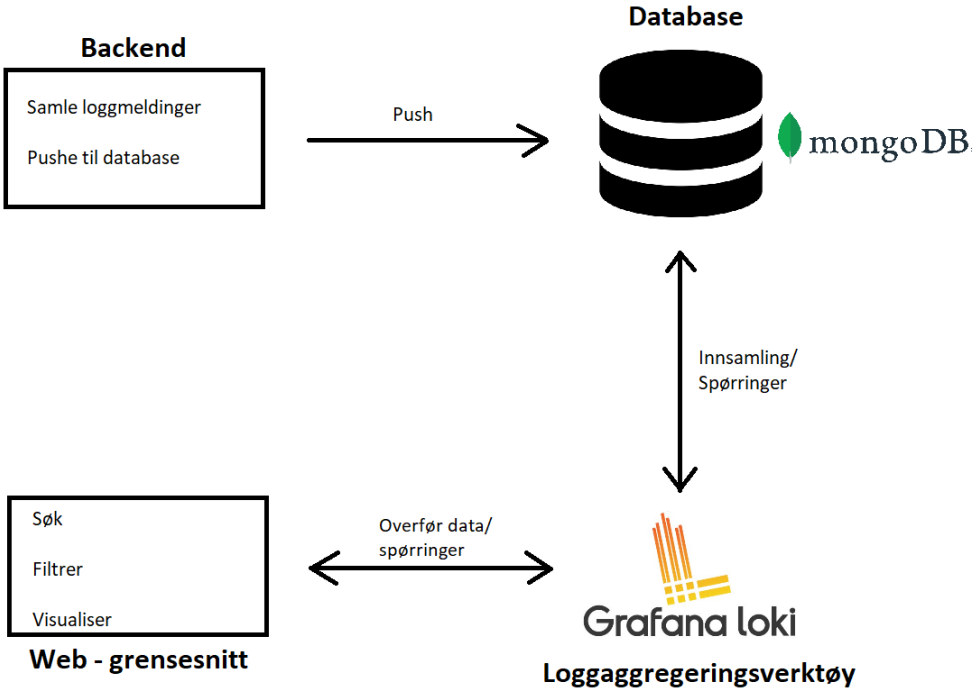
5.3 Prosjektresultat

Evalueringsresultatene bidro i stor grad til å løse problemstillingen for prosjektet. Systemet sentraliserer loggmeldingene, håndterer dem etter brukerens behov og presenterer ønsket data i brukergrensesnittet for monitorering. Målsettingene for prosjektet er oppnådd ved å ferdigstille et sluttprodukt som møter kravene til oppdragsgiver. Systemet har en online webapplikasjon med et brukervennlig grensesnitt som kan søke etter og filtrere loggmeldinger i databasen. Loggmeldinger kan visualiseres som grafer i et dashbord for å se overvåke systemet på en oversiktlig måte.

Resultater fra integrasjonstestene og systemtesten viser at produktet har blitt utviklet riktig og at det er av høy kvalitet. Fra brukertestene kom det frem at det har blitt utviklet riktig produkt og at det har potensial for videre utvikling.

6 DISKUSJON

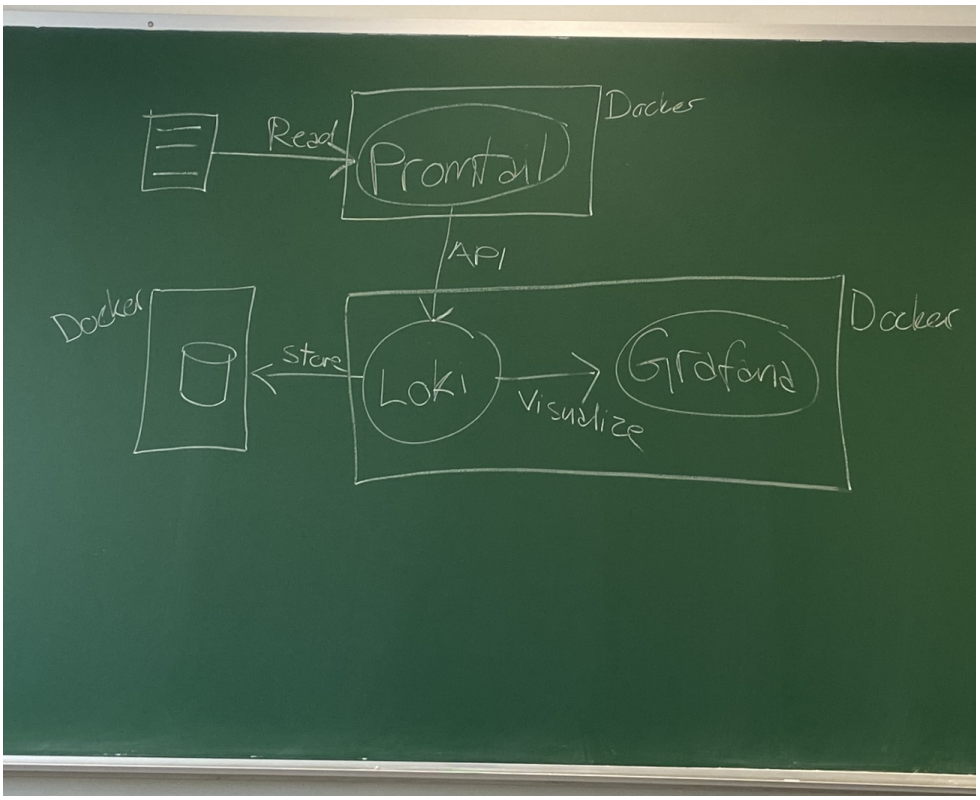
6.1 Avvik fra initiell løsnings idé



Figur 22: Skisse av initiell løsningsidé

Tidlig i prosjektet laget gruppen en initiell løsnings idé som skulle svare på problemstillingen. Den avviker i stor grad fra hva den endelige løsningen kom til å bli. Initielt tenkte vi at: Grafana Loki skulle lese loggmeldinger ut fra en database, der databasen fikk loggmeldinger fra backend-systemet. Backendsystemet var planlagt å utvikles i et Java program, som scraper loggmeldinger fra en filplassering og pusher disse opp i databasen. Denne databasen skulle være en NoSQL database av typen MongoDB, som kunne ta imot store mengder ustrukturert data. Det var viktig at databasen kunne ta imot ustrukturert data fordi loggmeldingene fra de ulike modulene i pCT systemet kunne produsere loggmeldinger i ulike format. Her var MongoDB perfekt fordi den lagrer data i JSON liknende dokumenter, som ikke er bundet til forhåndsdefinerte kolonner og datatyper som i en SQL-database.
Videre var det tenkt at web-grensesnittet skulle utvikles av gruppen fra bunn av. Web-grensesnittet skulle ha en søke og filtreringsfunksjon, samt en graffunksjon for visualisering. Det skulle i tillegg snakke med Grafana Loki som henter loggdata fra databasen.

Gruppen lagde det første utkastet av programvaren i form av en demo. Demoen hentet inn loggmeldinger fra en fil, sorterte og kategoriserte loggmeldingene og pushet de til databasen. Etter fremvisning av demoen på et møte, samt videre undersøkelser og diskusjoner med oppdragsgiver ble det klart at det fantes bedre løsninger, som kunne være langt mer praktisk i kombinasjon med Loki.



Figur 23: Arkitekturkladd, noen steg nærmere dagens løsning

6.1.1 Database

Databaseløsningen var det første som ble diskutert og gjort om på. Planen var at MongoDB skulle være langtidslageret til Loki, der Loki hentet ut meldinger som ikke var lagret i hurtigbufferet. Problemet var at to aktører var ansvarlig for databasekontroll. Det ga lite mening å sortere og laste opp loggdata til en database fra et backendsystem, for at Loki deretter henter ut loggmeldingene fra databasen for og så lagre data i et hurtigbuffer. Loki sin jobb er blant annet å behandle og lagre loggdata, og bør også være ansvarlig for dette. I stedet for å ha to ledd i løsningen som tar seg av databasekontroll, sentraliserte vi dette i Loki. Den initielle løsningen med MongoDB som databaseløsning ble dermed droppet. Loggdata ble nå sendt direkte til Loki uten det ekstra mellomleddet.

Videre undersøkelser viste også at MongoDB i seg selv ikke hadde vært nok som langtidslager, da Loki trenger to databaseløsninger. En for chunks og en for indekser. Dette førte igjen til et veiskille, der gruppen måtte bestemme seg for en ny databaseløsning. Denne diskusjonen er tatt opp i kapittel 3.1. Gruppen endte opp med å velge Amazon S3 og DynamoDB.

6.1.2 Logg-scraper

Initielt hadde gruppen planlagt at backend verktøyet for logg-scraping skulle programmeres i Java. Dette var tenkt å være en applikasjon gruppen skulle utvikle fra bunnen av. Videre undersøkelser førte oss frem til verktøyet Promtail, som er en del av Grafana stacken og anbefalt av mange. Det viste seg at Promtail er spesielt egnet i kombinasjon med Loki. Promtail er også godt egnet for kjøring i Docker containere, og var derfor svært aktuelt. Promtail er satt opp slik at

konfigurasjonene enkelt kan endres på i en YAML-fil utenfor Docker containeren, og kan kalles på ved oppstart. Promtail krever at noen småjusteringer gjøres ved førstegangsoppsettet på en ny maskin. Dette innebærer å spesifisere hvilke loggfiler som skal leses, og fra hvor. At dette kan enkelt endres på utenfor Docker containeren er en stor fordel. Promtail er totalt sett et veldig fleksibelt program som kan skreddersys med utallige funksjoner.

Utover dette er Promtail laget for å feste labels til loggmeldinger før de blir sendt til Loki, og opererer med en syntaks Loki lett forstår. Dette øker effektiviteten av programmet, samt risikoen for uventede feil blir redusert.

6.1.3 Web-grensesnitt

I stedet for å kode et web-grensesnitt fra bunn av, undersøkte gruppen Grafana som web-grensesnitt. Grafana grensesnittet er laget for å kunne koble opp mot Grafana Labs sine tjenester. Det som går igjen her, som med de andre endringene gruppen gjorde, er kompabilitet, samt et mye høyere utvalg av funksjoner. Skulle gruppen laget et web-grensesnitt fra bunn, måtte hver minste funksjon blitt implementert, og funksjonene hadde heller ikke vært like tilbøyelige for tilpasninger. Grafana derimot har et stort utvalg av funksjonalitet for søk, filtrering og grafer som kan spesialtilpasses forskjellige behov. Tilpasninger av dashboard for nye loggutlesninger som kommer i ettertid av gruppens prosjektoppgave, er dermed veldig fint å justere.

6.1.4 Docker Containere

Grafana og Loki var planlagt å kjøre i en og samme Docker container, databasen var også planlagt å kjøre i en egen container (se figur 23). Ettersom databasen ble til en skybasert database, kunne containeren som skulle inneholde databasen droppes. Det ble bestemt at Loki og Grafana skulle kjøres i hver sin Docker container i stedet for i en og samme. Grunnen er at en Docker container burde tildeles ett ansvarsområde, for å oppnå “the single responsibility principle”, og burde også innholde alt ansvar for dette området for å oppnå “high cohession”. Dette kan igjen tilgjengeliggjøre for å oppnå “low coupling”, der enkelte Docker containere kan endres, uten å måtte tilpasse andre kjørende containere.

6.2 Drøfting

Oppdragsgiver stilte krav til et nettbasert verktøy for å samle loggmeldinger fra systemet, behandle dem og deretter presentere dem. Verktøyet skulle utvikles i et Linux miljø, og skulle ha et sett med funksjonelle egenskaper:

- Lagre historikken til alle logghendelser til systemet
- Søkefunksjon
- Evne til å filtrere logghendelser
- Presentere utvalgte hendelser ved hjelp av grafiske verktøy

Det ferdigstilte produktet gruppen leverte oppfylte alle kravene til oppdragsgiver på en god måte. Ved fremvising av verktøyet for sentral loggføring, i kombinasjon med en brukertest, kunne gruppen konkludere med dette. Gjennom hele prosjektperioden har pCT teamet hatt en ukentlig SCRUM, der progresjon i arbeidet har blitt presentert. Der har det blitt diskutert, gjort tilpasninger og endringer underveis. Dette har vært til stor nytte for å holde utviklingsprosessen på riktig spor, og skreddersy applikasjonen for oppdragsgiver.

Verktøyet har en databaseløsning for langtidslagring av logghendelser, Amazon S3 og DynamoDB.

Grafana dashboard innehar både søke og filtreringsfunksjoner for logghendelser. Grafana har også funksjonene for å presentere logghendelser med grafiske verktøy.

Loki tar seg av innsamling av loggmeldinger og behandler disse. Promtail sikrer at loggmeldinger blir sendt til Loki.

Med dette implementert, har gruppen oppfylt kravene til oppdragsgiver.

Problemstillingen for prosjektet er “Hvordan utvikle et fleksibelt og enkelt kjørbart verktøy som samler, håndterer og presenterer hendelseslogger i et distribuert system?”.

Med de overnevnte funksjonalitetene i verktøyet, kan det konkluderes med at verktøyet besvarer det andre leddet i problemstillingen; “verktøy som samler, håndterer og presenterer hendelseslogger i et distribuert system”. Verktøyet fungerer i et distribuert system, der Promtail leser og sender loggmeldingene fra hver klient, Loki mottar loggene fra Promtail, samler og håndterer disse og lagrer de i en database. Grafana presenterer loggmeldingene fra Loki.

Når det kommer til den første delen av problemstillingen; “Hvordan utvikle et fleksibelt og enkelt kjørbart verktøy”, kan dette besvares med hvordan gruppen har implementert verktøyet i ferdig konfigurerte Docker containere. Docker containerne spinnes i gang med Docker Compose, og kan kjøres på enhver Linux maskin som har driverne for Docker og Docker Compose installert. Gruppen har opprettet to pakker, en server-side og en klient-side pakke. Disse pakkene inneholder alt som er nødvendig å kjøre på klient eller server. Alt som trengs er å navigere til den nedlastede pakken, skrive “docker-compose up” i CLI og så er verktøyet oppe og kjører.

Med dette kan det konkluderes med at verktøyet er fleksibelt og enkelt kjørbart. Det samler, håndterer og presenterer loggmeldinger, og besvarer dermed problemstillingen.

Gruppen er fornøyd med resultatet og kan konkludere sammen med oppdragsgiver at målet er nådd. Verktøyet er brukervennlig, lett og tilpasse og vil komme til god nytte under utviklingen av pCT apparaturet. Det vil også kunne tilpasses og brukes for overvåking av systemhendelser når det ferdige systemet installeres på Haukeland.

6.3 Positive og negative sider

Tidlig i prosessen var mye uklart, ting kunne virke overveldende og det tok en stund før hele oppgaven falt oss inn. En av de største spørsmålene til gruppen tidlig i prosjektet var "Hvor kommer disse loggmeldingene fra, hvilke moduler genererer disse?". Det tok en stund før dette kom tydelig frem. I mellomtiden mens gruppen ventet på svar, opprettet vi fiktive loggmeldinger som ble kjørt i den første demoen, og det ble gjort noen antagelser på hvilket format disse loggmeldingene kom til å ha. Senere kom det frem at oppdragsgiver selv ikke var helt sikker på hvilket format loggmeldingene kom til å ha. Svaret fikk vi først etter at siste prototype var utviklet. Hittil var hele systemet bygget på fiktive meldinger produsert av den første demo-versjonen. Formatet på loggmeldingene skal ikke ha en innvirkning på logg-aggregerings verktøyet, men det hadde vært greit å testet likevel. På siste brukertest fikk gruppen lest ut de ekte loggmeldingene fra pCT apparaturet. Det var heldigvis slik at formatet ikke hadde en innvirkning på verktøyet, og resultatet var at alt fungerte bra. Dette er likevel noe som hadde vært fordelaktig å få redegjort for i starten av prosjektet.

Det var en utfordring at oppdragsgiver var stasjonert i Tyskland. Det meste av kommunikasjon var via Zoom, men enkelte arbeidsoppgaver hadde vært best å drøftet og løst med fysisk oppmøte. Oppdragsgiver var svært teknisk dyktig og de fleste problemene ble løst på en alternativ måte sammen over Zoom. Oppdragsgiver var også i Bergen ved to anledninger, der gruppen fikk utløp for mange ideer og spørsmål. Totalt sett ble tilgjengeligheten til oppdragsgiver løst på en grei måte, men det hadde vært en fordel om han var tilgjengelig på UiB. Gruppen tenker at enkelte problemstillinger kunne vært løst raskere da.

De ukentlige tirsdagsmøtene var et veldig bra bidrag. Ved å løse oppgavene med en SCRUM, kunne gruppen jobbe i iterasjoner, drøfte resultater og planlegge veien videre. På møtene var veileder, oppdragsgiver, noen lærere og masterstudenter som også jobbet på pCT prosjektet. Dette ga også et helhetlig overblikk over fremgangen i det overordnede prosjektet. Her kunne personer komme med innspill og idéer som kom til god nytte.

I eksamensperioden fikk ikke gruppen jobbet like mye på oppgaven. Dette var en periode der en del arbeid gikk tapt, som måtte tas igjen senere. Det ble derfor behov for økt arbeidsmengde de siste ukene før innlevering. Planleggingen rundt eksamensperioden kunne vært gjort bedre, og arbeidet fordelt utover en lengre periode. Gruppen fikk likevel tatt igjen det tapte arbeidet.

7 KONKLUSJON OG VIDERE ARBEID

7.1 Konklusjon

Sluttproduktet består av et distribuert system i et Linux miljø som sentraliserer og behandler loggdata fra moduler slik at pCT kan monitoreres. En bruker kan via en web applikasjon selektere subsystemer som skal analyseres i et gitt tidsintervall. Verktøyet klarer å ta imot store mengder data over tid på en god måte.

Målet for prosjektet har vært å utvikle et verktøy som kan overvåke et distribuert system gjennom sentralisert logganalyse i en online web applikasjon. Ved å integrere verktøyet bestående av teknologi for logg-aggregering i Proton CT, har prosjektet nådd dette målet.

Evalueringsresultatene som er beskrevet i kapittel 5, er en god indikasjon på hvordan de opprinnelige målene i kapittel 1 er nådd. Systemtesten beviste at verktøyet kunne sentralisere loggmeldinger fra flere noder til ett dedikert system. Den viste også at atferden til systemet kunne vises gjennom visuelle verktøy i et dashbord i en online web applikasjon. Resultatene fra brukertestene ble brukt til å vurdere hvor brukervennlig og funksjonelt systemet er fra perspektivet til en ny bruker. Evalueringsskjemaet illustrert i tabell 2 gir et positivt bilde på nettopp denne vurderingen. Alle kravene fra oppdragsgiver er oppfylt og tilbakemeldingene på sluttproduktet har vært svært positive. Vi kan med dette konkludere at målet for oppgaven er oppnådd.

Etter å ha utviklet et system som møter kravene til oppdragsgiver, kan problemstillingen besvares: Hvordan utvikle et fleksibelt og enkelt kjørbart verktøy som samler, håndterer og presenterer hendelseslogger i et distribuert system? Gjennom grundig undersøkelser av ulike komponenter ble det bestemt hva som skulle implementeres i verktøyet, og hva som skulle kastes. Svaret på problemstillingen er grundig beskrevet i rapporten. Her står det om hva slags teknologi som er benyttet, hvordan de fungerer og hvordan de er implementert. Rapporten forklarer og hvordan alle komponenter i det distribuerte systemet samhandler for å forme et verktøy for sentral loggføring i proton CT.

7.2 Videre arbeid

Produktet har gjennom prosjektet blitt utviklet på en måte som tilrettelegger for videreutvikling. Gruppen har en delt filplassering på GitLab sammen med resten av pCT systemet, der prosjektet enkelt kan lastes ned. Instruksjoner for nedlastning og bruk, samt kodedokumentasjon er godt forklart på engelsk slik at alle som jobber med prosjektet kan gjøre tilpasninger om nødvendig. Klientpakken kan lastes ned på flere maskiner om det blir behov ytterligere overvåking.

Som forklart i Brukertest 2 i delkapittel 5.2, var oppdragsgiver innstilt på å gjøre videre arbeid på systemet. Påstand 14 i evalueringsskjemaet, illustrert i tabell 2, viser om han ser potensial for videreutvikling av systemet. Her svarte han "helt enig", og begrunnet det med:

> Det virker som både klient- og serverpakken tilbyr den nødvendige funksjonaliteten og en enkel og rett fram installasjon. Det meste av videreutviklingen vil være å sette opp dashbord for forskjellige subsystemer. Dette er forventet å være en standard oppgave i Grafana systemet. Systemet er veldig fleksibelt og vil tillate å koble til en passende database backend. (Richter, brukertest 2, kap. 5.2)

Videreutviklingen vil være å implementere de resterende subsystemene til pCT i vår løsning og sette opp et dashbord for hver av disse systemene i brukergrensesnittet. Databaseløsningen er planlagt å byttes ut til noe som passer systemene til Haukeland sykehus, der pCT skal stå klart i 2024. Det er tilrettelagt for dette i konfigurasjonen i dagens løsning på en måte som gjør det lett for en ny bruker å forstå.

8 REFERANSER

Aadnevik, D. og Pettersen, H. (2019) *Stråling og protonterapi.* Tilgjengelig fra: https://helse-bergen.no/fotnoten/fotnoten-forsking/straling-og-protonterapi (Hentet 04. februar 2022)

Amazon (u.å.a) *What is Amazon S3?* Tilgjengelig fra: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (Hentet 20. mai 2022)

Amazon (u.å.b) *What is Amazon DynamoDB?* Tilgjengelig fra: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html (Hentet 20. mai 2022)

Amazon (u.å.c) *What is continuous integration?* Tilgjengelig fra: https://aws.amazon.com/devops/continuous-integration/ (Hentet 10. mai 2022)

Aven, T. (2015) Risikoanalyse, *Store Norske Leksikon.* Tilgjengelig fra: https://snl.no/risikoanalyse (Hentet: 08. mars 2022)

Berman, D. (2017) *Logs or Metrics - A conseptual decision.* Tilgjengelig fra: https://logz.io/blog/logs-or-metrics/ (Hentet 09. mai 2022)

Cloudflare (u.å.) *What is data scraping?* Tilgjengelig fra: https://www.cloudflare.com/learning/bots/what-is-data-scraping/ (Hentet 27. mars 2022)

Davidson, A. (2021) *Databases Behind the Scenes: How BoltDB Saves Data To the Disk.*Tilgjengelig fra:https://itnext.io/databases-behind-the-scenes-how-boltdb-saves-data-to-the-disk-de8f1e3feda6 (Hentet 12. mars 2022)

Docker (u.å.a) *Networking in Compose.* Tilgjengelig fra: https://docs.docker.com/compose/networking/ (hentet:

Grafana (u.å.a) *Grafana Loki.* Tilgjengelig fra: https://grafana.com/oss/loki/ (Hentet: 09. mars 2022)

Grafana (u.å.b) *Storage.* Tilgjengelig fra: https://grafana.com/docs/loki/latest/storage/ (Hentet 12. mars 2022)

Grafana (u.å.c) *Single Store Loki (boltdb-shipper index type).* Tilgjengelig fra: https://grafana.com/docs/loki/latest/operations/storage/boltdb-shipper/ (Hentet 12. mars 2022)

Grafana (u.å.d) *Promtail.* Tilgjengelig fra: https://grafana.com/docs/loki/latest/clients/promtail/ (Hentet: 26. mars 2022)

Grafana (u.å.e) *Components* . Tilgjengelig fra: https://grafana.com/docs/loki/latest/fundamentals/architecture/components/?src=blog&camp=timeshift_53 (Hentet 1. mai 2022)

Grafana (u.å.f) *Grafana Loki Documentation*. Tilgjengelig fra: https://grafana.com/docs/loki/latest/ (Hentet 1. mai 2022)

Grafana (u.å.g) *Grafana Loki's Architecture*. Tilgjengelig fra: https://grafana.com/docs/loki/latest/fundamentals/architecture/ (Hentet 1. mai 2022)

Grafana (u.å.h) *Grafana Loki's API*. Tilgjengelig fra: https://grafana.com/docs/loki/latest/api/ (Hentet 3. mai 2022)

Grøttvik, O. S. (2021) *Design and Implementation of a High-Speed ReadOut and Control System for a Digital Tracking Calorimeter for proton CT.* (PhD thesis 02/2021). Bergen: UIB. Tilgjengelig fra: https://bora.uib.no/bora-xmlui/handle/11250/2725567 (Hentet: 07. mars 2022).

Horovits, D. (2020) *The complete guide to the ELK stack.* Tilgjengelig fra: https://logz.io/learn/complete-guide-elk-stack/#intro (Hentet: 09. mars 2022)

IBM Cloud Education (2019) *What is PostgreSQL?* Tilgjengelig fra: https://www.ibm.com/cloud/learn/postgresql (Hentet: 08. mars 2022)

Institutt for fysikk og teknologi, UIB (2021) *Medical Physics: The Bergen pCT project.* Tilgjengelig fra: https://www.uib.no/en/ift/142356/medical-physics-bergen-pct-project (Hentet: 07. mars 2022).

Institutt for fysikk og teknologi, UIB (2014) Institutt for *fysikk og teknologi, Universitetet i Bergen.* Tilgjengelig fra: https://www.facebook.com/fysikkogteknologi/ (Hentet: 09. mars 2022).

IntelliJ IDEA (2022) *IntelliJ IDEA overview.* Tilgjengelig fra: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (Hentet: 09. mars 2022)

Jacobsen, J. (2021) *Hva er greia med Docker?* Tilgjengelig fra: https://grafana.com/docs/loki/latest/fundamentals/architecture/ (Hentet: 16. mai 2022)

Kisller, E. (2021) *The basics: A beginner's guide to Docker.* Tilgjengelig fra: What Is Docker? A Beginner's Guide | JFrog (Hentet 24. april 2022)

Kumar, P. (2021) *Logging at Scale in Kubernetes using Grafana Loki.* Tilgjengelig fra: https://medium.com/nerd-for-tech/logging-at-scale-in-kubernetes-using-grafana-loki-3bb2eb0c0872 (Hentet 01. mai 2022)

MongoDB (n.d) *What is MongoDB?* Tilgjengelig fra: https://www.mongodb.com/what-is-mongodb (Hentet: 08. mars 2022)

NREC (u.å.a) *What is NREC?* Tilgjengelig fra: https://nrec.no/ (Hentet 22. april 2022)

Patterson (u.å) *A case for redundant arrays of Inexpensive disks .* Tilgjengelig fra: http://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf (Hentet: 19. april 2022)

PostgreSQL (n.d) *What is PostgreSQL?* Tilgjengelig fra: https://www.postgresql.org/about/ (Hentet: 08. mars 2022)

Schaefer, L. (n.d) *What is NoSQL?* Tilgjengelig fra: https://www.mongodb.com/nosql-explained (Hentet: 07. mars 2022)

Schwaber, K. og Sutherland, J. (2011) *Scrumguiden.* Tilgjengelig fra: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-NO.pdf (Hentet: 08. mars 2022)

ThinkAutomation (n.d) *The history of databases.* Tilgjengelig fra: https://www.thinkautomation.com/histories/the-history-of-databases/ (Hentet: 08. mars 2022)

9 VEDLEGG

Følgende dokumenter er vedlagt:

- Prosjekthåndbok
- Visjonsdokument
- Kravdokumentasjon
- Systemdokumentasjon