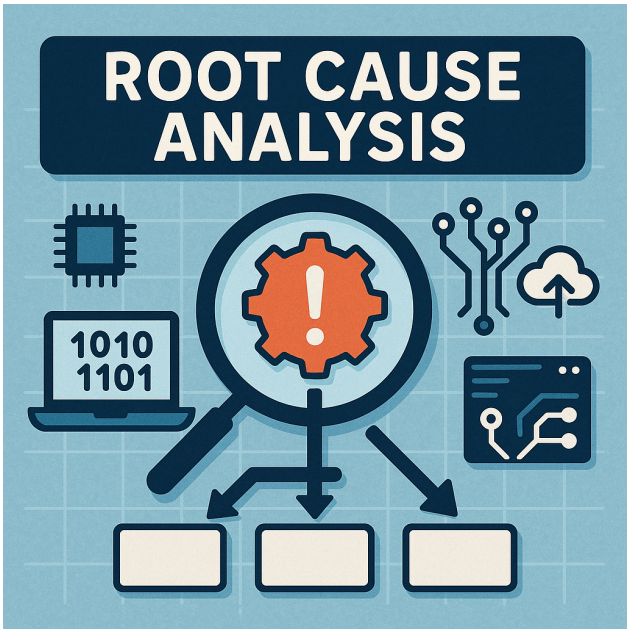
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Effektivisering av problemidentifikation samt grundorsaksanalys med hjälp av optimerad visualisering av telemetri- och loggdata

Kandidatarbete inom Data- och Informationsteknik

David Andersson
William Arkhult
Edvin Bengtsson
Eric Norén
Muhammad Tariq Khalidee
Hampus Åkerlund

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
GÖTEBORG, SVERIGE 2025

David Andersson
William Arkhult
Edvin Bengtsson
Eric Norén
Muhammad Tariq Khalidee
Hampus Åkerlund

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg, Sverige 2025

Effektivisering av problemidentifikation samt grundorsaksanalys med hjälp av optimerad visualisering av telemetri- och loggdata

David Andersson William Arkhult Edvin Bengtsson Eric Norén
Muhammad Tariq Khalidee Hampus Åkerlund

Handledare: Romaric Duvignau, Institutionen för Data- och Informationsteknik
Rådgivare: Johan Fischer, MyGuard
Examinatorer: Patrik Jansson och Arne Linde, Institutionen för Data- och Informationsteknik
Rättande lärare: Magnus Almgren, Institutionen för Data- och Informationsteknik

Kandidatarbete 2025
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola och Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Titelsidans bild: En AI-genererad bild från Sora (OpenAI, 2025), som visar grundorsaksanalys.

Typsats i LaTeX
Göteborg, Sverige 2025

Abstract

With the ever-increasing societal reliance on computer systems comes a need for these systems to meet a high standard of reliability. Even the most robust of systems will have occasional faults, which could be due to internal or external factors. In both cases, the issues need to be identified through data analysis and remedied. Root cause analysis (RCA) is a vital but time-consuming process in large-scale environments. Traditionally performed manually, it requires extensive analysis of telemetry and log data to identify system faults, often delaying resolution and demanding significant human effort. In order to address these challenges, we have developed a system that automates and streamlines RCA.

Our solution is an interactive web application, developed to accommodate the three main aspects of the project, which are data handling, visualisation and automation. Data handling is done using Elasticsearch, which receives log data directly from the server and Prometheus, which collects telemetry data. All collected data is timestamped to enable correlation for RCA. The data is then visualised in a two-part system consisting of both a Grafana dashboard and a React-based application, which serves as the central interface. The application also provides anomaly detection capabilities to find irregularities in the telemetry data stream and in system health. This is achieved using the machine learning algorithms Random Forest and Isolation Forest.

With this system, the necessary data for conducting RCA is presented to the user by giving the ability to look through timestamped anomalies in telemetry data and correlated log data both near-time and historical views. Thus, reducing both the human effort and streamlining the process of RCA.

Sammandrag

När samhället blir mer och mer beroende av datasystem ökar även kraven på att dessa system har en hög standard för pålitlighet. Även de mest robusta systemen drabbas emellanåt av fel, vilka kan ha såväl interna som externa orsaker. I båda fallen krävs att felen identifieras genom dataanalys och åtgärdas. Grundorsaksanalys är en central men tidskrävande process i storskaliga systemmiljöer. Traditionellt genomförs grundorsaksanalys manuellt, vilket innebär omfattande analys av telemetri- och loggdata för att lokalisera systemfel, något som ofta fördröjer felsökningen och kräver betydande mänsklig insats. För att bemöta dessa utmaningar har vi utvecklat ett system som automatiserar och effektiviserar processen.

Vår lösning utgörs av en interaktiv webbapplikation som adresserar tre huvudsakliga aspekter: datahantering, visualisering och automatisering. Datahanteringen sker genom Elasticsearch, som tar emot loggdata direkt från servern, samt Prometheus, som samlar in telemetridata. All insamlad data tidsstämplas för att möjliggöra korrelation vid grundorsaksanalyser. Visualiseringen sker i ett tvådelat system bestående av

en Grafana-dashboard och en React-baserad webbapplikation, som utgör systemets centrala gränssnitt.

Applikationen erbjuder även funktionalitet för anomalidetektering i syfte att identifiera avvikelser i både telemetridata och systemets hälsa. Detta uppnås genom användning av maskininlärningsalgoritmerna Random Forest och Isolation Forest.

Med detta system presenteras relevant data för att genomföra grundorsaksanalys på ett effektivt sätt, genom att användaren får möjlighet att granska tidsstämplade anomalier i telemetridata och relevanta loggdata, både i närtid och i historiskt perspektiv. Därmed minskar den mänskliga arbetsinsatsen och processen effektiviseras avsevärt.

Förord

Detta projekt utfördes som ett kandidatarbete under våren 2025 vid Chalmers Tekniska Högskola. Vi vill tacka Chalmers för möjligheten att göra detta projekt, men framför allt vill vi rikta vårt varmaste tack till vår handledare från Chalmers, Romaric Duvignau, och vår rådgivare Johan Fischer från MyGuard. Dessutom vill vi tacka Hans Malmström från Chalmers för utomordentlig hjälp gällande disposition, struktur och stil inom rapporten.

Vi vill även tacka Dennis Lyberg, Niklas Jarl och Patrik Olsson - Arista samt Mattias Ahnberg - Netnod för att de tog sig tid att hålla presentationer och delta i diskussioner kring telemetri, övervakning av system och tidsförskjutning.

David Andersson, William Arkhult, Edvin Bengtsson, Eric Norén,
Muhammad Tariq Khalidee, Hampus Åkerlund, Göteborg Maj 2025

Innehåll

Figurer

Tabeller

Begreppslista

- **Grundorsaksanalys**: Strukturerad metod för att identifiera den grundläggande orsaken till ett problem eller fel i ett system.
- **RCA:** Root Cause analysis, engelska översättningen av grundorsaksanalys.
- **Telemetridata**: Mätdata från systemkomponenter för övervakning och analys.
- **Loggdata**: Textbaserad, tidstämplad information om händelser och processer i ett system.
- **Prometheus**: Verktyg som används för insamling och lagring av tidsseriedata för systemövervakning.
- **Grafana**: Plattform som används för visualisering och analys av insamlad data från exempelvis Prometheus.
- **Elasticsearch**: Sökmotor och analysverktyg som används för att indexera och söka i stora mängder data, främst loggar.
- **Kibana**: Ett verktyg för att visualisera och analysera data som lagras i Elasticsearch.
- **SSH (Secure Shell)**: Protokoll för säker fjärråtkomst till system över osäkra nätverk.
- **ML (Maskinlärning):** Metod där en dator lär sig mönster i data för att sedan kunna fatta beslut utan explicit programmering.
- **Anomali**: Synonymt med avvikelse. I detta fall innebär en anomali att ett system inte beter sig som förväntat.
- **Isolation Forest**: En ML-algoritm för anomali- och avvikelsedetektering som fungerar genom att isolera observationer.
- **Random Forest**: En ML-algoritm som klassificerar data genom att bygga upp beslutsträd.
- **Accuracy**: Andel av alla förutsägelser från ML-modellen som är korrekta, både positiva och negativa.

- **Precision**: Andel korrekt identifierade normala datainstanser utav totala antalet identifierade normala datainstanser.
- **Recall**: Andel korrekt identifierade normala datainstanser utav totala antalet normala instanser.
- **Specificity:** Andel korrekt angivna anomalier utav totala antalet anomalier.
- **Confusion matrix:** En matris som visualiserar antalet korrekta och antalet felaktiga förutsägelser, av varje klass, en ML-modell har gjort på ett dataset.
- **Klassificering:** Processen när en ML-modell evaluerar data och förutspår dess klass.
- **Överanpassning**: När en ML-modell presterar bra på träningsdata men dåligt på ny, osedd data.
- **Distribuerade system**: System av flera noder eller datorer som samarbetar över nätverk för att uppnå ett gemensamt mål.
- **Docker**: Plattform för containerisering som automatiserar distribution av applikationer i containers.
- **Logganalys**: Analys av insamlade loggfiler för att identifiera avvikelser och fel.
- **Observerbarhet**: Systemets förmåga att möjliggöra insikt i dess interna tillstånd genom externa outputs som loggar, mätvärden och spår.
- **SSH-tunnling**: Teknik för att säkra dataöverföring genom att vidarebefordra nätverkstrafik genom SSH-anslutningar.
- **Villkorsbaserad anomalidetektering**: Manuell avvikelsedetektering där mätvärden kontrolleras mot fördefinierade tröskelvärden.

1
Inledning

Inledningen syftar till att presentera bakgrunden och den teoretiska grund som denna rapports undersökning, problemställning och arbete baseras på. Inledningsvis ger kapitlet en beskrivning av ämnet, relevanta tekniska aspekter inom telemetri- och logganalys samt systemövervakning. Vidare beskrivs de underliggande orsakerna till problemställningen, exempelvis de bekymmer som företag bemöter gällande hantering och visualisering av deras telemetridata, som är mätdata från systemkomponenter. Fortsättningsvis beskrivs syftet med projektet samt de delproblem som arbetet delas upp i och vilka avgränsningar som görs. Avslutningsvis diskuteras de etiska och hållbarhetsmässiga aspekterna av det föreslagna systemet.

1.1 Bakgrund

Många företag har svårt att utnyttja telemetri- och loggdata, vilket är automatiskt insamlade händelseuppgifter från ett system. I dagens samhälle är risken för dataintrång, cyberattacker och systemkrascher ett aktuellt problem för många företag. Exempelvis kostar en DDoS-attack (Distributed Denial-of-Service attack) i genomsnitt över 4,4 miljoner kronor att åtgärda, vilket visar på ett stort behov av effektiva skydd mot cyberattacker [1]. Effektiv grundorsaksanalys, identifiering av orsak till ett fel i ett system, av telemetri- och loggdata kan däremot förebygga och motarbeta detta [2]. Tyvärr är det många företag som saknar de nödvändiga verktygen för analysarbetet, vilket ofta resulterar i många timmar manuellt arbete. Med ett välutvecklat visualiseringsverktyg för telemetri- och loggdata kan processen underlättas avsevärt och således öka ett systems tillförlitlighet, prestanda och säkerhet.

1.1.1 Ämnesområdet som undersöks

Orsaken till systemkrascher och avvikelser i ett datasystem kan identifieras genom analys av systemets telemetri- och loggdata. I större datasystem genereras stora mängder sådan data, vilket enligt teknikbolaget Splunk kan användas för att spåra orsaker till systemkrascher och även för övervakning av systemhälsan [3]. Utan rätt verktyg kan denna form av grundorsaksanalys bli resurskrävande då den ofta stora mängden data behöver analyseras manuellt. Oförmågan att genomföra grundorsaksanalys innebär en säkerhetsrisk eftersom bolagen förblir omedvetna om orsaken till sina systemkrascher och avvikelser.

1.1.2 Relevans och målgrupp

Främst är det teknikbolag som gynnas av tillgången till effektiva visualiserings- och analysverktyg för telemetri- och loggdata. Men även företag som hanterar stora mängder data har nytta av effektiv grundorsaksanalys för att motverka systemkrascher. Teknikbranschen växer kontinuerligt, vilket i sin tur ökar efterfrågan på produkter som denna. Samtidigt träder EU:s cybersäkerhetskrav, NIS2-direktiven, i kraft i Sverige sommaren 2025 [4].

Direktiven ställer krav på ett förstärkt skydd mot cyberattacker och förbättrade rutiner för hantering av sådana, särskilt för företag med samhällskritisk infrastruktur [5]. Exempelvis lyfter plattformen Logmanager vikten av en effektiv logghantering för att företag ska kunna följa de nya direktiven [6]. Därav ökar behovet av effektiva verktyg för analys och hantering av telemetri- och loggdata markant.

1.2 Syfte

Projektet syftar till att undersöka hur grundorsaksanalys, identifiering samt diagnostisering av systemproblem kan effektiviseras jämfört med traditionellt manuellt arbete genom att utveckla en interaktiv webbapplikation för användarvänlig visualisering av telemetri- och loggdata.

1.3 Problembeskrivning

Grundproblemet är att visualisera telemetri- och loggdata på ett sätt som underlättar grundorsaksanalys vid systemkrascher och liknande problem. Målet är att lösa detta problem genom att utveckla en interaktiv och användarvänlig webbapplikation. Projektets huvudsakliga problembeskrivning delas upp i delproblem för att underlätta arbetet och möjliggöra avgränsningar. Nedan följer en utförligare beskrivning av delproblemen, nämligen datafiltrering, tidsanalys, visualisering och automatisering.

1.3.1 Datafiltrering för relevans

Telemetri- och loggdata samlas automatiskt in i stora mängder vilket gör det svårt att hantera och visualisera. Ett delproblem är därför att undersöka möjligheten att filtrera ut den data som anses viktigast. Därav undersöks vilken telemetri- och loggdata som är viktigast vid grundorsaksanalys. Om filtrering inte fungerar krävs andra lösningar för att hantera datamängden. En alternativ metod är stickprover för att dela upp data, vilket gör hanteringen lättare men kan försvåra grundorsaksanalys.

Ytterligare en möjlig metod är att bryta ner data. Istället för att hantera de stora mängderna telemetri- och loggdata, skapas statistik över vilka datatyper de består av. För det undersöks olika sätt att kategorisera telemetri- och loggdata för användning i grundorsaksanalys.

En tidslinje av händelser kan förenkla visualiseringen av data men även för att precisera ursprunget av ett fel. Exempelvis kan grundorsaken dölja sig bakom den kedjeeffekt som uppstår i ett system med en felaktig enhet som i sin tur skickar vidare felaktig data.

1.3.2 Tidsanalys

Vid grundorsaksanalys är det viktigt att kunna visa data i kronologisk ordning. Detta försvåras av klockskillnader på servrar, och eftersom data ska hanteras i närtid (inom 10 sekunder av händelsen) kan små tidsskillnader skapa stora problem. Ett delproblem är därför att undersöka hur telemetri- och loggdata kan visas i rätt ordning. Om garanterandet av tidsordningen på data är omöjligt kan alternativ testas. Ett möjligt tillvägagångssätt är framtagningen av ett verktyg som avgör sannolikheten att data är i rätt ordning. Verktyget kan i så fall möjliggöra grundorsaksanalys även vid felaktigt tidsordnad data.

1.3.3 Visualisering

För att visualisera den data som bearbetas skapas en interaktiv webbapplikation. Detta delproblem är därför en stor del av arbetet och slutprodukten. Telemetri- och loggdata behöver visualiseras på ett sådant sätt att det lätt går att utföra grundorsaksanalys på den. Även att visualisera fel och problem i telemetri- och loggdata på ett tydligt sätt är ett mål i delproblemet. Därför undersöks vilka verktyg som applikationen behöver för att tydligt visa var och när saker går fel, men även hur felen hänger ihop med varandra. Webbapplikationen bör även vara användarvänlig på ett sådant sätt att den inte kräver stora förkunskaper för att användas.

1.3.4 Automatisering

Eftersom systemkrascher kan uppstå på relativt kort tid är det viktigt att varningar ges tidigt för att kunna stoppa dem. Att detektera och larma vid onormala händelser som oscillation eller spikar i dataflödet på servern, höjda nivåer av felmeddelanden eller avvikelser i systemhälsan gör det möjligt att förhindra problem innan de uppstår. En möjlighet är anomalidetektering, vilket är processen att identifiera avvikelser från normala beteendemönster i data, med hjälp av ML (maskininlärning). Det kan ge snabba resultat och förbättra chanserna att förhindra allvarliga problem. Nackdelar med ML är att det kräver mycket träningsdata och många tester för att ge en tillräckligt hög tillförlitlighet innan det kan användas. Därav påbörjas utvecklingen av anomalidetekteringsmodellen tidigt i arbetsflödet för att uppnå en tillräckligt hög standard för användning inom grundorsaksanalys.

1.4 Avgränsningar

Eftersom problemidentifiering och grundorsaksanalys är relevant inom många industrier, då det finns en stor mängd olika telemetri- och loggdata, riskerar projektets omfattning att bli alltför bred. Därför avgränsas arbetet till användning av telemetri-

och loggdata från servrar för att hålla projektet på en rimlig storleksnivå. Mjukvaran baseras på den telemetri- och loggdata som erhålls från servern som företaget MyGuard låter oss arbeta med. Detta begränsar den slutgiltiga mjukvaran till den miljö som noderna används inom.

Det huvudsakliga målet med applikationen är endast grundorsaksanalys till problem som uppstår hos ett system, men inte nödvändigtvis att erbjuda lösningar till dessa problem. Detta val motiveras av framtida förbättringspotential där grundorsaken till ett problem står som ett logiskt första steg till en lösning.

Mätning av telemetri- och loggdata kan kategoriseras i tre olika tidsspann vilka är realtid, närtid och historisk data. Applikationen fokuserar enbart på närtid och historisk data, vilket innebär data äldre än 10 millisekunder. Detta motiveras av otillförlitligheten hos realtidsdata som kräver en alltför låg exekveringstid och noggrannhet hos både mjukvara och hårdvara.

Utvecklingen av applikationen fokuserar i huvudsak på att säkerställa en lokalt fungerande programvara utan fokus på kompatibilitet med externa API:er. Elasticsearch, en sökmotor och analysverktyg för främst loggdata, och Kibana, ett visualiseringsverktyg, är verktyg som används i projektet för datahantering. Denna mjukvara används i samband med Prometheus, ett verktyg för datainsamling, och Grafana för bearbetning, visualisering och analys av data. Därav begränsas projektet till vad som är möjligt inom dessa verktygs ramar.

1.5 Etiska aspekter

Då tekniska lösningar forskas fram och implementeras är det alltid viktigt att beakta etiska och samhälleliga aspekter. Detta avsnitt syftar till att diskutera hur dessa frågor relaterar till projektet.

1.5.1 Datasäkerhet och integritet

Då insamling av telemetri- och loggdata sker och analyseras finns det en risk att privat, säkerhetskänslig eller affärskritisk data samlas in. Med detta i åtanke är det viktigt att säkerställa att den data som behandlas inte riskerar att skada varken integriteten hos privatpersoner eller säkerheten för företag. Detta kan åstadkommas genom att anonymisera data samt att personuppgifter och information hanteras lagligt och etiskt. Dessutom går det att verifiera användaren av verktyget med hjälp av exempelvis tvåfaktorsautentisering, vilket är en bra säkerhetsanordning för system liknande det som utvecklas i arbetet.

1.5.2 Ansvarsfull användning

Vid utveckling av ett dataanalysverktyg bör det ses över vilka potentiella organisationer och personer som får tillgång till verktyget. Detta för att säkerställa ansvarsfull användning av verktyget. Risken med dataanalysverktyg är att de kan användas

för att övervaka både anställda i organisationer på ett integritetskränkande sätt, men även individer i samhället som stort. Därför bör tydliga policyer och begränsningar implementeras som gör att det inte går att missbruka verktyget.

1.6 Hållbarhetsaspekter

Utvecklingen av produkten medför möjligheter och utmaningar när det kommer till global och lokal hållbarhet. Projektet kan granskas utifrån de tre dimensionerna av hållbarhet, vilka är ekonomiska, sociala och ekologiska aspekter.

1.6.1 Ekonomisk hållbarhet

Genom att effektivisera problemidentifiering med grundorsaksanalys kan företag och organisationer minska sina driftkostnader genom att snabbt identifiera och åtgärda systemfel innan de påverkar systemet alltför mycket. Detta minskar produktionsbortfall och sparar resurser som annars skulle användas för omfattande felsökningar och reparationer. En snabbare och mer exakt grundorsaksanalys kan även leda till ökad produktivitet, då tekniker och utvecklare kan fokusera på förbättringar snarare än felsökning och systemundersökningar. Vidare kan ett automatiserat analysverktyg minska behovet av externa supporttjänster. Något som i längden bidrar till en mer kostnadseffektiv verksamhet, vilken kan fokusera och lägga mer resurser på tillväxt. Företag som implementerar denna typ av optimerade visualiseringslösningar kan därmed få en konkurrensfördel genom ökad effektivitet och bättre resursanvändning, vilket gynnar den ekonomiska hållbarheten.

1.6.2 Social hållbarhet

En mer effektiv och automatiserad analys av telemetri- och loggdata bidrar till en förbättrad arbetsmiljö genom att minska stressen hos utvecklare och tekniker, som annars behöver hantera systemöversikten samt felsökningsprocesserna manuellt. Med ett intuitivt visualiseringsverktyg blir det lättare för fler på företaget att förstå och använda data, vilket främjar samarbete mellan människor med olika yrken och på olika avdelningar. Dessutom bidrar projektet till kompetensutveckling. Det ger studenter och yrkesverksamma möjlighet att arbeta med avancerade analysverktyg, vilket ökar anställningsbarhet och tekniskt kunnande.

1.6.3 Ekologisk hållbarhet

Genom optimering av systemens prestanda och förminskning av onödiga processer kan energiförbrukningen bli markant lägre, vilket leder till minskade indirekta koldioxidutsläpp och miljöpåverkan. Effektivare systemunderhåll och snabbare grundorsaksanalys kan även förlänga livslängden på datorer och systemutrustning, vilket minskar mängden elektroniskt avfall på lång sikt. Projektet bidrar också till en mer hållbar digitalisering genom att ge organisationer bättre översikt på hur deras digitala infrastruktur ser ut och kan hanteras på ett resurseffektivt sätt. Dessutom kan behovet av fysiska resor för felsökning minska genom att fler problem kan identifieras och lösas från distans, vilket i sin tur minskar koldioxidutsläpp av transportmedel. På så sätt kan projektet hjälpa till att skapa en mer ekologiskt hållbar organisation genom optimerad användning av både hårdvara och energi.

2
Teknisk bakgrund och teori

Detta kapitel presenterar den tekniska bakgrunden och de teoretiska ramverken som ligger till grund för utvecklingen av applikationen. Genom att redogöra för relevanta teknologier och teori ges en fördjupad förståelse för de val som tas under utvecklingsprocessen, samt hur dessa komponenter samverkar inom systemet. I kapitlet introduceras även verktygen som används i projektet.

2.1 Distribuerade system

Ett distribuerat system består av flera självständiga datorer eller noder som är sammankopplade via ett nätverk, vilka samarbetar för att uppnå ett gemensamt mål. Dessa system kännetecknas av sin skalbarhet, feltolerans och parallell bearbetning av data. I utvecklingen av moderna applikationer används distribuerade system för att säkerställa hög tillgänglighet och lastbalansering, särskilt när stora mängder data ska behandlas i närtid [7].

Dessutom möjliggör distribuerade system modularisering, vilket innebär att applikationen delas upp i separata komponenter som var och en ansvarar för en specifik funktionalitet. Fördelen med modularisering är ökad robusthet. Om en enskild komponent kraschar, påverkas inte nödvändigtvis övriga delar av systemet, så länge det inte föreligger ett direkt beroendeförhållande. I system med beroenden mellan komponenter kan ett fel sprida sig och orsaka att även andra komponenter slås ut [7]. För att hantera denna typ av komplexa felkedjor används grundorsaksanalys, vilket möjliggör systematisk spårning av incidentens ursprung.

2.2 Grundorsaksanalys

Grundorsaksanalys är en uppsättning av strukturerade metodologier för att identifiera de underliggande orsakerna till problem eller händelser, snarare än att endast åtgärda symptomen som uppstår [8]. I kontexten av distribuerade system kan grundorsaksanalys definieras som processen att identifiera de primära bakomliggande faktorerna som bidrar till systemfel, prestationsproblem eller avvikelser i systembeteende.

Faktiska fel och avvikelser har inte samma innebörd inom ramen av distribuerade system. Fel avser en tjänsts oförmåga att utföra sina avsedda funktioner, medan avvikelser representerar de observerbara symtom som indikerar underliggande fel. Ex-

empel på avvikelser är försämrad svarstid, minskad genomströmning eller förekomst av felmeddelanden. Detektering av avvikelser kan ske antingen på applikationsnivå, där man betraktar applikationen som helhet, eller på tjänstenivå där fokus ligger på specifika tjänster [9].

Grundorsaksanalys innebär att identifiera de bakomliggande orsakerna till observerade avvikelser på applikations- eller tjänstenivå, med det övergripande målet att klargöra varför ett visst fel inträffade. Detta kan åstadkommas genom att analysera den information som samlats under applikationens faktiska körning. Det ska således inte förväxlas med traditionella felsökningsmetoder, som innebär att man försöker återskapa det observerade felet i en testmiljö genom att omstarta och simulera applikationens beteende. I stället bygger grundorsaksanalys på information från loggar och övervakningsdata som samlats in under systemets normala drift, utan att applikationen behöver köras om i en modifierad miljö [9].

2.2.1 Metodologier för grundorsaksanalys

Wang och Qi [10] presenterar en systematisk översikt över metodologier för grundorsaksanalys inom mikrotjänstarkitekturer. De beskriver fyra huvudsakliga angreppssätt:

- **Spårningsbaserad analys:** Distribuerad spårning möjliggör rekonstruktion av exekveringsflödet för individuella förfrågningar genom systemets olika komponenter, vilket underlättar identifiering av prestandaflaskhalsar och felpunkter. MicroRank [11] är ett representativt exempel som lokaliserar orsaker till fördröjningar i mikrotjänstbaserade miljöer genom att analysera normala och icke-normala spårningar inom systemet.
- **Logganalys:** Strukturerad bearbetning av loggar kan avslöja mönster och anomalier som tyder på underliggande systemproblem. Onion [12] är ett exempel på ett verktyg som automatiserar identifiering och lokalisering av relevanta loggar i molnbaserade system med hög träffsäkerhet och effektivitet.
- **Telemetribaserad analys:** Övervakning av kvantitativa indikatorer såsom CPU-användning, svarstider och minnesförbrukning är centralt för att upptäcka beteendeavvikelser. Verktyget DejaVu [13] använder historisk anomalidata och komponentberoenden för att identifiera felkällor i distribuerade onlinetjänster.
- **Grafisk analys**: Genom att modellera kausala beroenden mellan komponenter i systemets topologi möjliggörs effektiv identifiering av rotorsaker. Ett framstående exempel är Grano [14] som integrerar mätdata, topologi och händelseloggar i en interaktiv grafstruktur. Verktyget visar sig i praktiken kunna reducera felsökningstider dramatiskt, exempelvis i eBays produktionsmiljö.

Wang och Qi [10] konstaterar att dessa metodologier ofta är komplementära snarare än ömsesidigt uteslutande, och att kombinationen av flera angreppssätt kan förbättra såväl noggrannhet som effektivitet i grundorsaksanalyser. Sådana hybridlösningar pekas ut som en central framtidsriktning för att stärka robustheten hos distribuerade system.

2.2.2 Observerbarhet och grundorsaksanalys

Begreppet observerbarhet (observability) etablerar sig som en utvidgning av traditionell övervakning. Medan övervakning fokuserar på kända tillstånd och problem, strävar observerbarhet efter att ge insikt i tidigare okända systemtillstånd [15].
Sridharan i sin bok *Distributed Systems Observability* [16], konstaterar att observerbarhet vilar på tre pelare:

1. **Loggar:** Detaljerade, tidsstämplade och textbaserade poster om händelser i systemet.
2. **Mätvärden**: Numeriska representation av systemets tillstånd och prestanda.
3. **Spårning:** Följandet av förfrågningar genom olika systemkomponenter.

Denna teoretiska uppdelning återspeglas i moderna verktygsuppsättningar för operationell analys.

2.3 Verktyg

Verktygen som används för grundorsaksanalys i projektet är Elasticsearch som hanterar loggar, Prometheus som samlar mätvärden och Grafana som visualiserar data för mänsklig tolkning. Utöver det används ML-modellerna Isolation Forest och Random Forest för anomalidetektering, vilket är processen att identifiera avvikelser från normala beteendemönster i data.

2.3.1 Prometheus

Prometheus är ett kraftfullt och flexibelt verktyg för systemövervakning, byggt på öppen källkod som används för att samla in, lagra och analysera mätvärden från olika källor inom ett datasystem [17]. Verktyget är särskilt populärt inom miljöer där närtidsövervakning är nödvändig, till exempel för att övervaka servrar, databaser eller nätverksenheter. Prometheus använder en så kallad “polling-metod” för att samla in data genom att kontinuerligt skicka förfrågningar till definierade mål i systemet. Dessa mål inkluderar Prometheus Exporters, såsom Node Exporter, vilka övervakar systemets resurser som CPU-, minnes- och diskanvändning. Det är även möjligt att använda applikationsspecifika exporters anpassade efter systemets behov.

De insamlade mätvärdena lagras i en databas som innehåller tidsserier, vilket betyder att varje mätvärde har en associerad tid. Detta gör det lätt att följa utvecklingen av värdena hos specifika komponenter, vilket behövs när en grundorsaksanalys skall utföras. Dessutom innehar Prometheus ett eget “query language” (språk för sökningar i data) som gör att man kan ställa förfrågningar gentemot denna tidsseriedatabas på ett användarvänligt sätt [17].

En fördelen med Prometheus är att den kan behålla stora mängder tidsseriedata som sedan går att vidare analysera och interagera med. Ett sådant interaktivt system kan utvecklas med hjälp av exempelvis Grafana som visualiseringsverktyg.

2.3.2 Grafana

Grafana är ett visualiseringsverktyg som bygger på öppen källkod, där den tar emot data från olika datakällor och sedan kan presentera denna data i olika grafer och strukturer för att på ett användarvänligt sätt visualisera de olika dataströmmarna [18]. Grafana används ofta i kombination med Prometheus för att övervaka serveranvändning hos företag.

Det finns ett antal olika sätt att visualisera data genom Grafana, exempelvis kan effektiva diagram och grafer skapas, men även mer detaljrika visualiseringar som linjediagram och värmekartor. Valfriheten som erhålls genom Grafana gör det möjligt att utveckla system som är användningsområdesspecifika för ens behov, vilket ofta ger ett gott resultat.

Grafana erbjuder ett webbaserat gränssnitt vilket sänker kunskapen som krävs hos användare, eftersom att de inte behöver använda sig av Command Line Interface (CLI). Denna användarvänlighet gör att det krävs mindre resurser för att använda systemet, vilket kan spara organisationer både tid och ekonomiska resurser. Dessa faktorer innebär att Grafana används som ett centralt verktyg inom delproblemet visualisering i projektet.

2.3.3 Elasticsearch

Elasticsearch är en distribuerad och skalbar sökmotor och analysplattform som bygger på Apache Lucene. I projektet tillämpas Elasticsearch som en central komponent för lagring, indexering och sökning av logg- och händelsedata. Pågrund av dess distribuerade arkitektur och höga prestanda möjliggör Elasticsearch närtidsanalys av stora datamängder, vilket är avgörande för effektiv grundorsaksanalys. Programmet ger användaren en överblick av systemstatus, detekterade avvikelser och relaterade loggar, vilket stärker både observerbarheten och felsökningsförmågan.

2.3.4 SSH

SSH, Secure Shell protokollet, är en metod för att säkert kommunicera och skicka instruktioner till servrar och datorer [19]. SSH använder sig av kryptografi för att både säkerställa att kommunikationen inte kan avlyssnas och läsas, samt för att autentisera anslutningar mellan datorer.

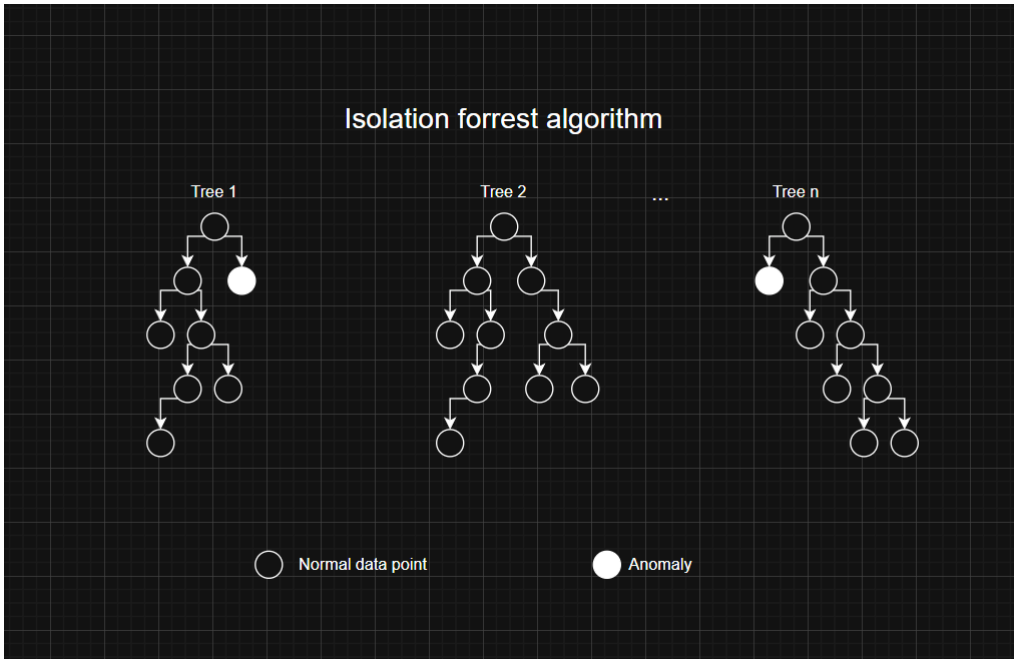
Den typ av kryptografi som gör SSH säkert, vilket leder till dess popularitet, är “public key cryptography” [19]. Sådan kryptografi använder en publik och en privat nyckel för att säkerställa att endast autentiserade anslutningar kan läsa eller skriva data och utföra instruktioner på datorn.

SSH har flera funktioner, som till exempel portforwarding genom okopplade nätverk, komprimering av data och möjligheten att visa applikationer med grafiska gränssnitt utan att fysiskt vara inkopplad. Dessa funktioner, tillsammans med säkerheten som

SSH erbjuder, är anledningen till att det är ett vanligt protokoll för exempelvis serverhantering.

2.3.5 Isolation Forest

Isolation Forest är en av två ML-algoritmer som testas i projektet för den automatiska anomalidetekteringen. Algoritmen bygger på unsupervised learning, det vill säga att träning genomförs på icke klassificerad data. Detta gör den särskilt väl lämpad för att upptäcka anomalier i data, vilket motiverar dess användning inom projektet [20]. I första steget i algoritmen väljs en telemetripunkt, en kategori av ett dataset, slumpmässigt ut. För den väljs ett slumpmässigt delningsvärde rekursivt för att på så sätt bilda träd och finna anomalier. Det för flera träd genomsnittliga antalet delningar som krävs för att isolera ett mätvärde avgör ifall det är en anomali eller inte, varav en lång väg ges för normala värden medan anomalier har korta vägar, se figur 2.1 [21]. Kort sagt delar Isolation Forest upp mätvärdena efter en trädstruktur, för att isolera avvikande värden och därav finna anomalier.



Figur 2.1: Isolation Forest algoritm

2.3.6 Random Forest

Algoritmen Random Forest är en supervised learning ML-algoritm, det vill säga att den tränas på klassificerad data. Random Forest tillhör kategorin ensemblemetoder och bygger på att kombinera ett stort antal beslutsträd för att skapa en mer robust och noggrann modell än vad enskilda träd kan erbjuda.

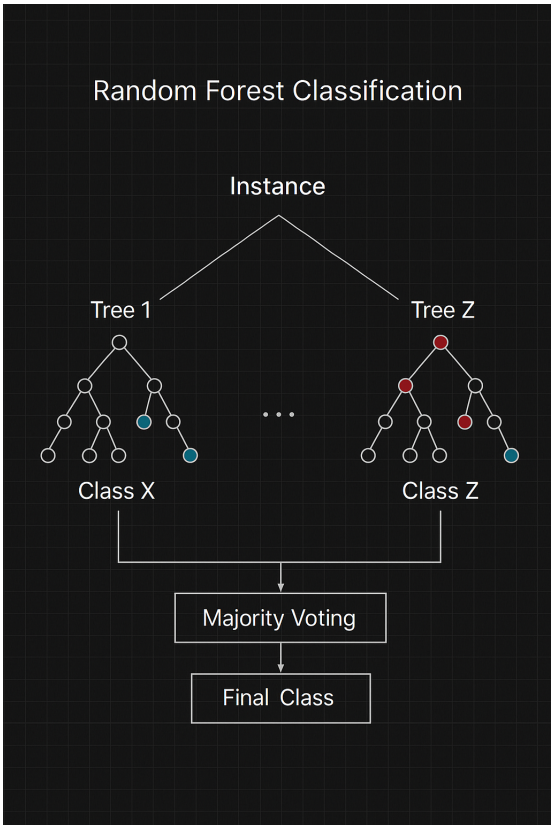
Varje enskilt träd i skogen konstrueras genom att slumpmässigt välja en delmängd av träningsdata med återläggning, en metod som kallas “bootstrap sampling”. Dessutom väljs vid varje delning i trädet en slumpmässig mängd av telemetripunkter

istället för att utvärdera alla tillgängliga. Detta tillvägagångssätt bidrar till att skapa variation mellan träden och minskar risken för korrelation mellan dem.

Klassificering är processen när en ML-modell evaluerar data och förutspår dess klass, vilket utförs genom att varje enskilt träd lämnar ett beslut baserat på sina parametrar, exempelvis huruvida en datapunkt klassificeras som en anomali eller ett normalt värde. Slutligen summeras dessa individuella beslut vid klassificering eller genom att beräkna medelvärdet vid regression, se figur 2.2. Detta innebär att den slutliga prediktionen är baserad på flera träd i modellen.

En av de främsta fördelarna med Random Forest är dess förmåga att reducera problemet med överanpassning, det vill säga när en modell presterar bra på träningsdata men dåligt på ny, osedd data. Även om ett enskilt träd kan överanpassa till sin träningsdata, minskar den totala effekten av detta när beslut från många oberoende träd kombineras. Random Forest är dessutom robust mot brus i data och klarar av att hantera både numeriska och kategoriska variabler utan krav på omfattande dataförbehandling.

Random Forest används i dag inom en rad olika områden såsom medicinsk diagnostik, kreditriskbedömning och rekommendationssystem för aktiehandel [22].



Figur 2.2: Random Forest algoritm

3
Metod

Detta kapitel syftar till att beskriva den metodik och arbetsprocess som tillämpas för att utveckla, testa och utvärdera applikationen. Projektet genomförs i samarbete med en extern handledare, Johan Fischer från MyGuard. Kravspecifikationen är baserad på behov och riktlinjer från den externa handledaren.

3.1 Metodval

I projektet tillämpades Design Science Research Methodology (DSRM)[23], en forskningsmetod som är särskilt lämpad för teknikorienterade utvecklingsprojekt där målet är att skapa och utvärdera tekniska artefakter. DSRM erbjuder en strukturerad och iterativ process för att säkerställa att den tekniska artefakten inte bara fungerar i praktiken, utan även tillgodoser verkliga behov och utvärderas systematiskt .
Följande steg i DSRM har anpassats till projektets omfattning och mål:

1. **Problemanalys och motivering**: Tillsammans med industrihandledaren identifieras behovet av ett verktyg för att förenkla grundorsaksanalys genom att korrelera telemetridata och loggar.
2. **Definition av mål för en lösning**: Kravspecifikationen och framgångskriterier formuleras med fokus på funktionalitet för anomalidetektering, logghantering och visualisering.
3. **Design och utveckling**: En iterativ process för framtagning av en prototyp, där systemets kärnfunktioner implementeras och testas löpande.
4. **Demonstration**: Lösningen testas i både simulerade scenarier och mot verkliga data från industrihandledaren, vilket möjliggör praktisk validering.
5. **Utvärdering**: Systemets prestanda, funktionalitet och användbarhet analyseras.
6. **Kommunikation**: Resultat och slutsatser dokumenteras och presenteras i rapporten för relevanta intressenter [23].

Denna metod möjliggör en balans mellan akademisk stringens och praktisk relevans, vilket framgår av projektets resultat i kapitel 5.

3.2 Arbetsprocess

Genom projektet följdes en iterativ och modulär arbetsprocess, strukturerad för att möjliggöra effektiv utveckling, testning och integration av flera komplexa komponenter inom en distribuerad systemmiljö. Arbetsprocessen formades i nära samarbete

med industrihandledaren och bygger på kontinuerlig återkoppling samt inkrementell utveckling.

- **Kravinsamling och planering**: Arbetet inleddes med en serie workshops och tekniska genomgångar tillsammans med industrihandledaren, där både funktionella och icke-funktionella krav definierades. Mätbara mål och tydliga milstolpar fastställdes, vilket lade grunden för det fortsatta utvecklingsarbetet.
- **Funktionella krav:** De funktionella kraven fokuserade på effektiv anomalidetektering, korrelation och visualisering av telemetri- och loggdata i syfte att underlätta grundorsaksanalys. Centrala krav omfattade:
  - Kontinuerlig och automatiserad anomalidetektering baserad på ML-modeller och villkorsbaserad metodik.
  - Integration och korrelation av telemetri- och loggdata utifrån gemensamma parametrar som tidsstämpel och IP-adress.
  - Användarvänligt gränssnitt med sammanfattande rapporter och möjligheter att länka till detaljerad visualisering av historisk data och närtidsdata.
  - Automatiserade processer för datahantering och analys för att minimera manuell insats.
- **Etablering av utvecklingsmiljö:** En viktig inledande fas omfattade konfiguration av den tekniska miljön. Här ingick uppsättning av kärnverktyg såsom Prometheus med Node Exporter för telemetriinsamling, Grafana för visualisering, samt Elasticsearch-stacken för logghantering. Därtill konfigurerades fjärråtkomst och säker kommunikation via SSH och portforwarding. Denna infrastruktur utgjorde grunden för det efterföljande utvecklingsarbetet.
- **Parallell komponentutveckling:** Med hänsyn till systemets komplexitet och beroenden mellan olika verktyg delades utvecklingsarbetet upp i flera tekniska komponenter, bland annat telemetriinsamling, logghantering, backend-API, ML-modell och gränssnitt. Dessa utvecklades parallellt och avgränsat från varandra för att effektivisera processen och minska integrationsproblem.
- **Kontinuerlig testning och validering:** Varje komponent testades isolerat i syfte att säkerställa funktionalitet och överensstämmelse med kravspecifikationen. Särskilt fokus lades vid verifiering av datakvalitet, API-kommunikation, korrekt loggindexering och visuell representation av mätvärden.
- **Integrationsfas:** När de enskilda modulerna uppfyllde sina respektive krav påbörjades ett stegvis utfört integrationsarbete. Under denna fas sammanfogades komponenterna till en sammanhållen helhet, med särskild fokus på att säkerställa stabil kommunikation mellan API:er, synkronisering av tidsstämplar, visualisering av resultat och kompatibilitet mellan datakällor.
- **Systemets och helhetsutvärdering:** Den slutgiltiga applikationen testades som helhet i en kontrollerad men realistisk miljö. Fokus låg på genomförandet av funktionella tester, medan analys av prestanda lämnades till framtida förbättringar.

Den iterativa och modulära metodiken möjliggjorde hög flexibilitet, snabb återkoppling samt tidig upptäckt av integrationsproblem. Kombination av modulär utveckling, kontinuerlig testning och gradvis integration var särskilt effektiv för att utveckla en applikation som sträcker sig över flera lager av systeminfrastruktur.

3.3 Datainsamling och testning

Datainsamling och testning var centrala delar för validering av systemets funktionalitet och robusthet. Två huvudsakliga typer av data användes:

- **Verkliga data:** Telemetri- och loggdata hämtades från industrihandledarens befintliga driftmiljö, vilket gav ett realistiskt sammanhang för systemets beteende i normal drift.
- **Simulerade data:** För att testa systemets respons vid fel och avvikelser skapades syntetiska scenarier. Dessa scenarier inkluderade vanliga typer av systemstörningar, för att generera avvikande mätvärden och loggar med felliknande mönster.

Testningen genomfördes på både komponentnivå och för den slutgiltiga applikationen:

- **ML-modellen:** Testades med olika konfigurationer och träningsdata för validering av dess förmåga att identifiera avvikande mönster i telemetridata. Fokus låg på prestanda vid detektering av anomalier.
- **Villkorsbaserad anomalidetektering:** Villkorsbaserade tröskelvärden applicerades på mätvärden för att snabbt identifiera uppenbara systemavvikelser.
- **Logghantering:** Elasticsearch-stacken testades för att säkerställa korrekt indexering, filtrering och korrelation av loggar med tidsstämplar och IP-adress.
- **Frontend gränssnittet:** Det grafiska gränssnittet testades för att säkerställa tydlighet i visualisering av anomalier och loggar samt korrekt funktionalitet.
- **Slutgiltiga applikationen:** Efter integrationen genomfördes systemtester för säkerställande av att samtliga komponenter fungerar tillsammans utan förlust av funktionalitet. Det inkluderade API-kommunikation, korrekt flöde av data mellan backend och frontend samt presentation av sammanhängande rapporter.
- **Stresstestning och felscenarier:** Systemet utsattes för medvetet skapade avvikande mätvärden och felaktiga loggar för att utvärdera dess förmåga att identifiera, korrelera och rapportera problem.

Tabell 3.1 ger en samlad översikt över samtliga testfall som behandlas i rapporten. Testfallen är listade i kronologisk ordning utifrån när de utfördes, vilket visar hur systemet har utvecklats genom de olika iterationerna. Varje rad anger kortfattat vilket delsystem som testats, vilken specifik version eller komponent som stod i fokus samt det unika testfalls-ID som används för att referera till testet i efterföljande kapitel. På så vis fungerar tabellen som en karta över rapportens empiriska underlag, läsaren kan snabbt identifiera vilka aspekter som verifierats och sedan fördjupa sig i respektive appendixavsnitt för metod, resultat och analys. Detta strukturerade upplägg säkerställer dessutom att alla kravspårnings- och kvalitetssäkringsaktiviteter tydligt kan kopplas till minst ett definierat testfall.

Tabell 3.1: Översikt av appendix–testfall

Nr	Namn	Testfalls-ID
1	Testning av Anomali Detektering med ML-modell, version 1	ADML1
2	Testning av Anomali Detektering med ML-modell, version 2	ADML2
3	Testning av Datahämtning genom Node Exporter samt Formatering med Prom2json	DH1
4	Testning av Villkorsbaserad Anomali Detektering	ADRB1
5	Hämtning av loggar från Elasticsearch via SSH-tunnel	ELASTIC01
6	Test av Elastic-backend, Filtreringsfunktioner och FastAPI	ELASTIC02
7	Testning av Anomali Detektering med ML-modell, version 4	ADML4
8	Kontinuerlig Anomali Detektering med ML-modell, version 5	ADML5
9	Testning av UI på Centraliserad hemsida	FE1
10	Kombinerad Kontinuerlig Anomali Detektering (villkor + ML)	ADMLRB5
11	Testning av Slutprodukt	SLP

Testfallen täckte ett brett spektrum av situationer och itererades löpande under utvecklingen. Den modulära arkitekturen underlättade isolerad testning av enskilda funktioner, vilket i sin tur påskyndade identifiering av eventuella felkällor och förbättringsmöjligheter.

3.4 Relaterat arbete

Detta avsnitt syftar till att ge en överblick över tidigare forskning och tekniska lösningar som är relevanta för projektet. Fokus ligger på algoritmer för anomalidetektering, hantering av telemetri- och loggdata samt metoder för grundorsaksanalys. Genom att studera befintligt arbete inom dessa områden motiveras de tekniska val som görs i projektets utvecklingsfas.

Relevant vetenskaplig litteratur har hittats genom sökningar i databaser såsom IEEE Xplore, Scopus, Google Scholar och ACM Digital Library med nyckelord som: “root-cause analysis”, “telemetry data”, “log analysis”, “anomaly detection using machine learning”, “observability” och “system monitoring machine learning”. Artiklar valdes ut baserat på vetenskaplig kvalitet, aktualitet och relevans för projektets mål.

Ett centralt moment i projektet var valet av algoritm för identifiering av anomalier i telemetridata. Isolation Forest är en ofta förekommande algoritm, känd för sin linjära tidskomplexitet och låga minnesanvändning, vilket gör den väl lämpad för högdimensionella datamängder. Den fungerar även utan tillgång till märkta träningsdata,

vilket gör den användbar i oövervakade scenarier [24]. Agyemang utvärderar fem oövervakade algoritmer och lyfter fram Isolation Forest som en av de mest träffsäkra, baserat på mått som accuracy, precision, F1-score och recall [25]. Liknande slutsatser dras av Yang et al., där Isolation Forest uppvisar hög effektivitet vid detektering av anomalier i storskaliga system [26].

Även Random Forest har identifierats som en stark kandidat. I en komparativ studie visar Primartha et al. att algoritmen presterar bättre än andra klassificerare, såsom Naive Bayes, neurala nätverk och ensemblemetoder, vid användning av k-korsvalidering [27]. Vidare visar Gupta att Random Forest är särskilt effektiv för anomalidetektering i molnbaserade miljöer, där den uppnår hög noggrannhet och låg falsk alarmfrekvens. Styrkorna inkluderar hantering av stora datamängder, robusthet mot saknade värden och möjlighet till tolkning genom feature importance [28].

Med utgångspunkt i ovanstående arbeten och flera andra relevanta studier [27], [26] beslutades det att både Random Forest och Isolation Forest ska implementeras och utvärderas i projektet, med fokus på prestanda och noggrannhet vid identifiering av anomalier.

För effektiv grundorsaksanalys i distribuerade system betonas i litteraturen vikten av att kombinera loggdata med telemetridata. Utifrån tidigare studier och litteratur som diskuteras i avsnitt 2.2, valdes en korrelationsbaserad metod där analys sker genom att identifiera samband mellan gemensamma attribut såsom tidsstämplar, IP-adresser och enhets-ID:n. Denna metod lade grunden för grundorsaksanalys inom detta projekt.

Sammanfattningsvis ger detta avsnitt en forskningsöversikt som motiverar valen av algoritmer och metodik i projektets utvecklingsarbete. Genom att förankra lösningen i etablerade tekniker säkerställs att implementationen bygger på beprövade principer.

3.5 Användning av AI

Generativ AI användes i projektet som ett stöd i både kodutveckling och dokumentationsarbete. AI-drivna verktyg, såsom kodningsassistenter och språkmodeller, tillämpades för att effektivisera utvecklingsprocessen och lyfta kvaliteten på det producerade materialet. Den AI-genererade informationen granskades alltid kritiskt och validerades mot etablerade vetenskapliga källor, för att säkerställa validitet, relevans och trovärdighet.

ChatGPT, utvecklad av OpenAI, användes vid texthantering och kodutveckling. Därtill användes även OpenAIs modell Sora för att skapa visuellt material, såsom illustrationsbilder i rapporten.

3.6 Versionshantering och dokumentering

Under projektets gång användes flera verktyg för versionshantering, fildelning och dokumentation. GitHub tillämpades för att hantera versionshistorik och samarbete kring kodbasen. Dokumentförfattande och samarbete kring textmaterial skedde huvudsakligen via Overleaf för LaTeX-baserad rapportskrivning, samt Google Drive för delning av kompletterande material. Dessa verktyg möjliggjorde strukturerat samarbete inom gruppen och bidrog till att både kod och dokumentation förblev väldokumenterad, synkroniserad och spårbar under hela projektets gång.

4
Produktutveckling

Detta kapitel belyser den tekniska produktutvecklingen som systemets funktionalitet byggs upp av. Centrala komponenter, såsom insamling och visualisering av data, som utgör infrastrukturen för produkten beskrivs. Vidare bedrivs en djup genomgång av hur insamlad telemetri- och loggdata hanteras, förbereds och visualiseras för vidare analys. Fortsättningsvis diskuteras vilka verktyg och teknologier som används, och hur de sammanhänger i systemets helhet. Avslutningsvis beskrivs även hur ML- och villkorsbaserade algoritmer kan användas för att identifiera systemanomalier.

4.1 Nätverks- och serverinfrastruktur

För att möjliggöra datainsamling i närtid är projektet uppsatt i en distribuerad miljö där mätdata inhämtas från en fjärrserver främst för utveckling och träning, men även för testning. På denna server är Prometheus installerat, som kontinuerligt hämtar systemets exporterade systemdata via definierade punkter. För att skapa en säker och kontrollerad kommunikation mellan klient och server används SSH-tunnling med portforwarding. Uppkopplingen sker via SSH med parametrarna -L och -C, där -L möjliggör att lokala portar kopplas till fjärrportar, och -C aktiverar komprimering för att reducera överföringsvolymen. Denna lösning gör det möjligt att övervaka och extrahera data från fjärrservern på ett effektivt och säkert sätt.

Lokal port	**Fjärrport**	**Tjänst**
3000	-	Frontend för Webbapplikation (lokalt endast)
3100	3000	Grafana Dashboard
5601	5601	Kibana
9090	9090	Prometheus Scraping
9100	9100	Node Exporter (metrics)
9200	9200	Elasticsearch
-	12222	SSH-ingångsport till fjärrserver

Tabell 4.1: Portmappningar via SSH-tunnel

Denna konfiguration möjliggör åtkomst till Prometheus via localhost:9090 på den lokala maskinen, även om tjänsten körs på en annan enhet. Genom att tunnla trafiken via SSH upprätthålls datasekretess, och fjärrservern kan hållas stängd för direkt trafik från externa IP-adresser. På fjärrnätverket finns olika maskiner som kontinuerligt kör olika processer. Dessa maskiner har olika IP-adresser under samma nät och används för att testa olika funktionaliteter av applikationen.

4.2 Centraliserad datainsamling

Detta avsnitt redogör för de två huvudsakliga datatyper som samlats in, telemetri- och loggdata. Dessutom beskrivs hur dessa hanterats genom hela flödet, från generering vid källan till lagring, filtrering och exponerad åtkomst via gränssnitt och API:er.

4.2.1 Telemetridata

För att samla in relevant telemetridata konfigurerades ett system där mätvärden hämtas från en fjärrdator som är ansluten till nätverket. Denna fjärrdator sattes upp i en containeriserad miljö med hjälp av Docker och all kommunikation med den skedde via SSH. Systemet konfigurerades att var femte sekund hämta så kallade metrics, vilket innebär numeriska mätvärden över systemets tillstånd, exempelvis CPU-användning, minnesbelastning, diskåtkomst och nätverksaktivitet. Datahämtningen skedde genom att exekvera kommandon mot en Prometheus-exporter som kördes på fjärrdatorn. För att möjliggöra vidare bearbetning konverterades råa mätdata från servern till JSON-format med hjälp av verktyget prom2json, ett verktyg med öppen källkod som översätter Prometheus-exporterade metrics till JSON. JSON-filerna formaterades därefter med hjälp av ett simpelt formaterings-skript i Python. Varje fil innehöll cirka 300 enskilda mätvärden, vilket gav ett bra underlag för vidare analys och maskininlärning.

4.2.2 Loggdata

Samtliga noder i systemet konfigurerades för att överföra loggdata till en central nod, Monitor, med hjälp av logghanteringsverktyget Rsyslog. Loggöverföringen skedde via TCP till port 5140, där varje inkommande datapaket märktes med nodens käll-IP som identifierande tag. Detta inkluderar loggarna den centrala noden själv genererade. Den centrala noden körde en containeriserad instans av Elasticsearch-stack i en Docker-miljö, med uppgift att aggregera, indexera och långsiktigt lagra inkommande loggar.

För att säkerställa att loggdata bevaras över tid konfigurerades en så kallad "retention policy" i Elasticsearch, för att lagra data i upp till sex månader. Ett nytt index skapas automatiskt varje dygn, vilket möjliggjorde effektiv uppdelning och åtkomst av loggar baserat på datum. Denna struktur gjorde det dessutom möjligt att genomföra historisk logghantering med hjälp av Elasticsearch Query DSL (Domain Specific Language), som erbjöd ett kraftfullt och flexibelt sätt att filtrera och söka i loggdata.

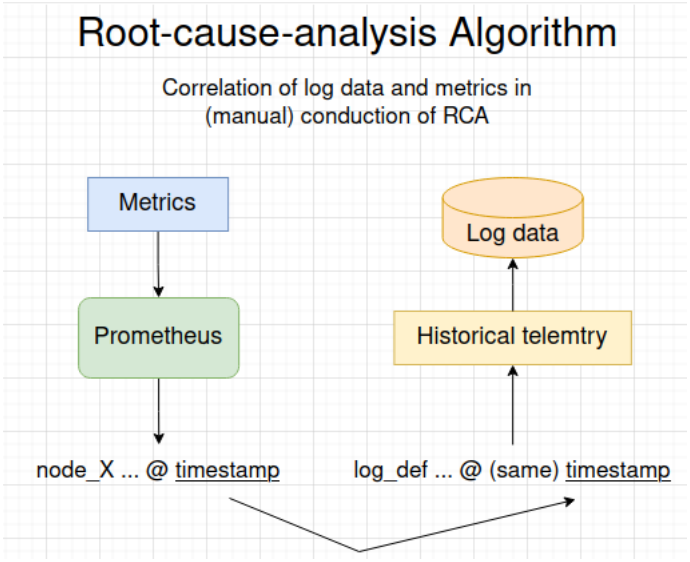
Metodiken för att inhämta loggdata liknar den strategi som även tillämpades för insamling av telemetridata. Genom SSH-tunnlar och portforwarding exponerades Elasticsearch och Kibana lokalt, vilket möjliggjorde åtkomst via den lokala utvecklingsmiljön. Backend-logiken implementerades i Python, där ramverket FastAPI i kombination med Uvicorn användes för att bygga en skalbar och responsiv backendserver.

Servern kommunicerade med Elasticsearch för att hämta loggar, som därefter exponerades lokalt via port 8000. Backend-logiken utökades med funktionalitet för att filtrera loggar baserat på specifika parametrar såsom IP-adress, tidsstämplar, samt nyckelord såsom “Error”, “Timeout” osv.

4.3 Grundorsaksanalysalgoritm

Projektets centrala algoritm för grundorsaksanalys bygger på korrelationen mellan närtidsinsamlad systemdata (metrics) och historiska loggdata. Figur 4.1 visar den övergripande arkitekturen för hur grundorsaksanalysprocessen är strukturerad. Processen inleds med att systemets metrics samlas in i närtid genom Prometheus. Dessa data, exempelvis CPU-belastning, minnesanvändning, virtuellt minne samt Input/Output (I/O), analyseras därefter i närtid av en ML-modell tränad för att identifiera anomalier, det vill säga avvikelser som signalerar att något inte står rätt till i systemet. När en anomali detekteras sparas den tillhörande tidpunkten (timestamp) automatiskt.

Denna tidpunkt fungerar sedan som en nyckel för vidare analys. Algoritmen går tillbaka i tiden och hämtar motsvarande loggdata från samma tidpunkt via historisk telemetridata. På så sätt kan systemets beteende vid just detta tillfälle granskas mer i detalj, vilket möjliggör en djupare förståelse av vad som orsakat felet eller problemet. Genom att koppla ihop mätvärden och loggar vid en gemensam tidpunkt kan algoritmen hjälpa till att identifiera rotorsaken till ett problem snarare än att bara peka på symptom. Metoden efterliknar en manuell grundorsaksanalys, men effektiviseras genom automatisering, vilket gör det möjligt att snabbt isolera felkällor även i stora och komplexa datasystem. Denna korrelation mellan metrics och loggdata utgör därmed kärnan i den grundorsaksanalys som projektet fokuserar på att utveckla och visualisera.



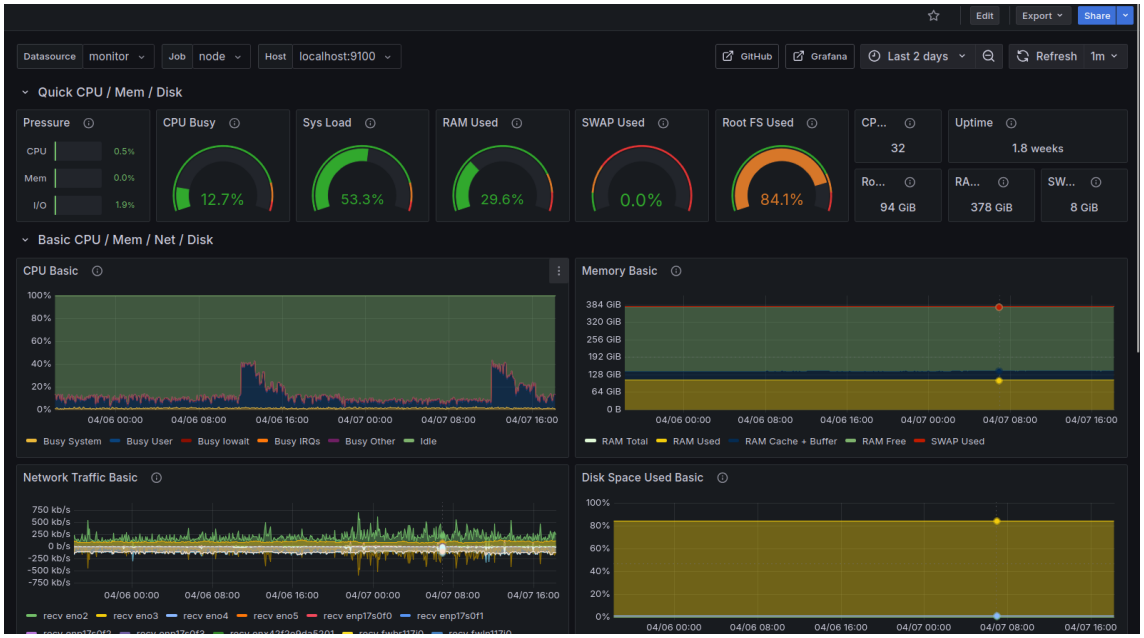
Figur 4.1: Flödesschema av algoritm för grundorsaksanalys

4.4 Visualisering

Visualisering av insamlad data är ett viktigt steg för integrationen mellan användare och system. Projektet nyttjar två verktyg för visualisering, en dashboard i Grafana och en hemsida anpassad för att sammanställa anomalidetekteringen med relevant loggdata.

4.4.1 Visualisering med Grafana

Grafana, ett kraftfullt verktyg för datavisualisering, användes för att presentera den insamlade informationen, exempelvis gällande CPU-temperatur eller minnesanvändning. Verktyget konfigurerades genom att sätta upp dataflöden från datainsamlingsagenten Prometheus, vilket möjliggjorde visualisering av dessa data. För detta projekt har en dashboard i Grafana skapats, som visas nedan i figur 4.2.



Figur 4.2: Grafana dashboard

Från Grafanas dashboard kan man tydligt utläsa de olika mätvärdenas nivåer vid specifika tidpunkter. Denna visualisering är ett kraftfullt verktyg för att genomföra manuella grundorsaksanalyser och få en detaljerad förståelse för systemets beteende och eventuella avvikelser.

4.4.2 Centraliserad hemsida

Detta projekt består av en bred omfattning mjukvara som använder en webbmiljö, där bearbetad data sammanställs och presenteras för användaren i syfte att förenkla problemidentifieringsprocessen. Användaren får en överblick av de detekterade anomalierna och de relevanta loggarna som korrelerar med den estimerade tidsstämpeln. Med denna information ges en helhetssyn över potentiella felkällor och

förenklar därmed grundorsaksanalysen. Hemsidan kopplas ihop med komponenter i projektet genom en API där exempelvis data om anomalier kontinuerligt kan laddas upp. Elasticsearch, som används till logghantering, är tillgängligt genom sin egen API och kan därigenom bidra med lämpliga loggar vid en tid då en anomali detekterats.

Applikationen använder NextJS som är ett React-framework. Detta i syfte att förenkla utvecklingsprocessen med hjälp av funktionalitet som vanligtvis tar erfarenhet och tid att implementera. Ett bra exempel på detta är ett verktyg kallat hooks från React, som bland annat tillåter att enstaka komponenter uppdateras utan att uppdatera hela sidan. Då ny data flödar in när nya anomalier upptäcks vill detta reflekteras i listan av anomalier. Genom användningen av dessa verktyg underlättas hela processen både under utveckling och testning.

4.5 Containerisering

Den slutprodukt som utvecklats i detta projekt består av flera olika delar, bland annat visualisering i Grafana och hantering av loggar via databaser. Eftersom systemet är beroende av många olika program och verktyg, är applikationen containeriserad. Detta görs med hjälp av Docker och Docker Compose.

Docker och Docker Compose är verktyg som används för att skapa och hantera isolerade miljöer, så kallade containrar, utifrån så kallade Docker-images. Dessa containrar innehåller all nödvändig programvara och alla beroenden som krävs för att systemet ska fungera korrekt. Detta förenklar installation och distribution, eftersom det minimerar risken för problem med saknade paket eller versionskonflikter på olika maskiner.

4.6 Automatisering

Automatisering är en central del i systemets design och syftar till att förenkla drift, övervakning och anomalidetektering. Detta uppnås dels genom skript för att initiera systemkomponenter som SSH-tunnlar, Grafana och Prometheus, och dels genom automatiserade mekanismer för att upptäcka avvikelser i telemetridata.

4.6.1 Automatisering genom skript

För att underlätta initialiseringen av SSH-tunnlar, Grafana och Prometheus utvecklades ett skript som automatiskt startar dessa delar av systemet.

En av fördelarna med att använda ett enkelt körbart skript för denna process är att det möjliggör för användare att enkelt starta systemet utan att behöva hantera avancerade kommandoradsinstruktioner. Detta förenklar och effektiviserar arbetsflödet, vilket frigör tid som istället kan ägnas åt mer värdeskapande aktiviteter såsom

användning, vidareutveckling och felsökning.

Utöver detta utvecklades ett ytterligare skript för att stoppa de tjänster som startats av det föregående skriptet. Denna funktion ger samma fördelar, genom att minska behovet av att hantera systemadministration manuellt och därigenom tillåta användare att fokusera på arbetsuppgifter som ger större nytta för projektets mål.

4.6.2 ML-Baserad anomalidetektering

För att automatiskt detektera anomalier i närtid i ett systems telemetridata, övervakades och klassificerades denna data av en ML-modell. Fem versioner av ML-baserad anomalidetektering utvecklades, varav fyra av versionerna testades utförligt för att evaluera prestanda samt finna eventuella förbättringar. Beroende på version av ML-modell mättes prestandan i fyra parametrar accuracy, precision, recall och specificity. Prestandan utvärderades även med hjälp av en confusion matrix, en matris som visualiserar antalet korrekta och antalet felaktiga förutsägelser av varje klass som en ML-modell har gjort på ett dataset. I version 3, 4 och 5 tränades modellerna med optimerade hyperparametrar (variabler som bestäms utifrån karaktären och distributionen av data), för att förbättra prestandan, se test ADML4, appendix A.7, för specificering av optimeringen.

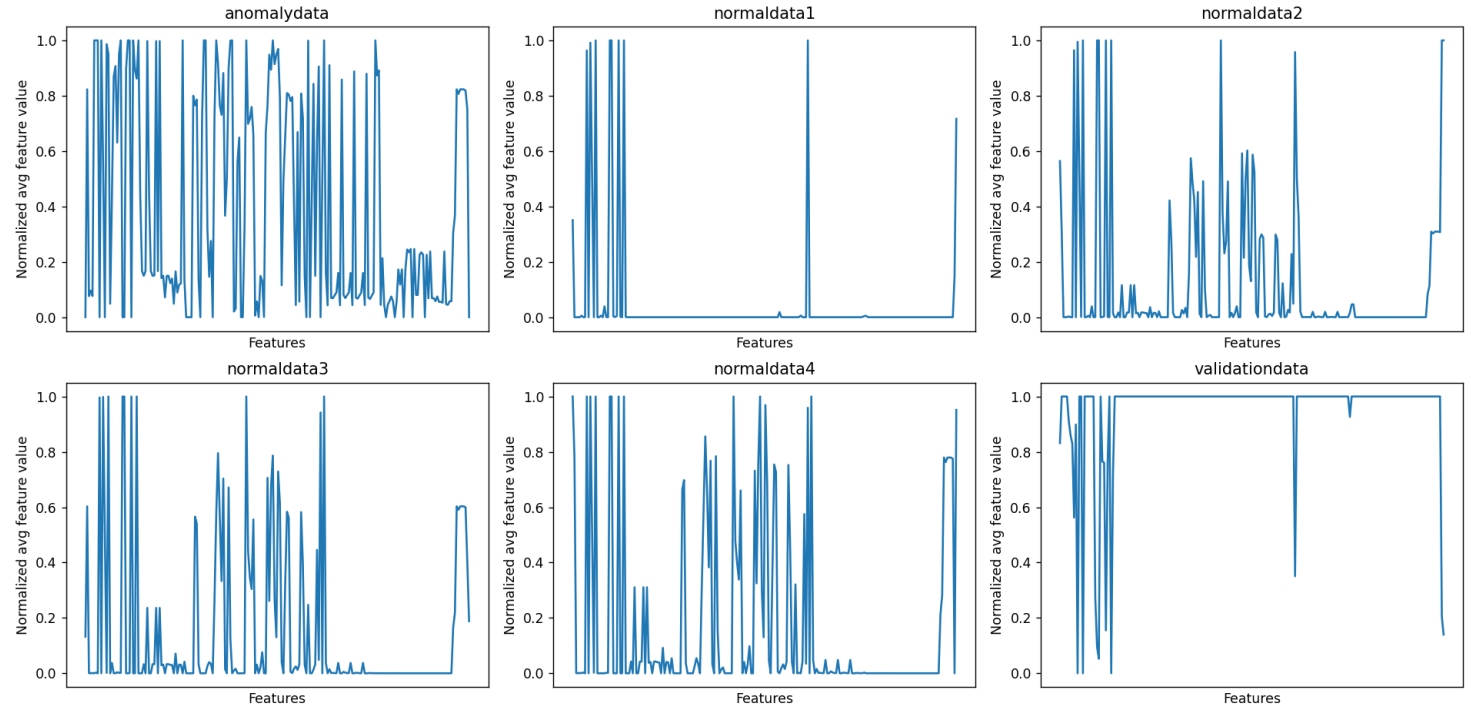
En grundorsaksanalys genomfördes även i version 4 och 5 då en anomali detekterades. Vid detektionen genomfördes en SHAP (SHapley Additive exPlanations) på avvikande data för att identifiera de tio mest avvikande telemetripunkterna. Dessa telemetripunkter med tillhörande anomalivärden skrevs ut av systemet i rangordning med den mest avvikande telemetripunkten först, se figur 4.3.

```
SHAP values for the first detected anomaly:
node_forks: 0.0296
node_disk_sectors_writtendm-0: 0.0294
node_disk_io_time_msdm-1: 0.0293
node_netstat_Icmp_OutEchoReps: 0.0198
node_netstat_TcpExt_TCPSackRecovery: 0.0198
node_netstat_TcpExt_TCPDSACKIgnoredOld: 0.0198
node_disk_writes_completeddm-1: 0.0198
node_memory_Active_anon: 0.0197
node_memory_SwapCached: 0.0197
node_netstat_TcpExt_DelayedACKLocked: 0.0197
```

Figur 4.3: Grundorsaksanalys med SHAP

Alla versionerna av ML-modellerna är kompatibla med Isolation Forest. Version 4 och 5 är dessutom kompatibla med Random Forest. ML-algoritmerna importerades från Python biblioteket scikit learns. Version 1 och 2 tränades samt valideras på syntetisk, genererad data. Resterande versioner tränades på fyra olika dataset, varav version 4 och 5 validerades på två dataset. Varje dataset bestod av 2000 till

5000 mätinstanser, varav vardera mätinstans består av ∼400 mätpunkter. Distributionen av denna data illustreras i figur A.4, med telemetripunkterna i x-axeln och det normaliserade medelvärdet för vardera telemetripunkt i y-axeln. I figuren visas sex olika dataset, där anomalydata, normaldata1, normaldata2 och normaldata3 använts som träningsdata, medan validationdata och normaldata4 använts som valideringsdata varav validationdata utgjorde anomalierna vid validering. Dessutom består validationdata utav datainstans där endast enstaka komponenter av servern är avvikande, till skillnad från anomalydata som dessutom består av datainstanser där flera komponenter av servern är avvikande samtidigt.



Figur 4.4: Grafer över normaliserade medelvärdet för alla telemetripunkter för vardera dataset.

Modellen bygger på algoritmen Random Forest, från Python-biblioteket scikit-learn. Fortsättningsvis är systemet generiskt byggt då det enkelt går att implementera en Isolation Forest modell istället för Random Forest. Random Forest modellen är tränad på tidigare insamlad telemetridata som består både av normal telemetridata och av anomalier. Anomalierna har samlats in genom att den observerade servern stressats på olika sätt, för att på så sätt simulera dataattacker, utan att skada servern. Vidare användes endast ett urval av de telemetripunkter som ansågs vara viktigast att övervaka. När en anomali detekterades av ML-modellen skickades tidsstämpeln för den data som undersöktes samt de tio telemetripunkter som ML-modellen ansåg avvek mest från de förväntade värdena till visualiseringsapplikationen, för att således varna användaren.

4.6.3 Villkorsbaserad anomalidetektering

Parallellt med utvecklingen av den ML-baserade applikationen för automatisk anomalidetektering utvecklades även ett anomalidetekteringsprogram som grundar sig i villkor. När specifika mätvärden, hämtade i närtid, över- eller understiger bestämda tröskelvärden, avgör programmet om det är en anomali eller ej. Det villkorsbaserade anomalidetekteringsprogrammet erbjuder en robust och förutsägbar metod för att snabbt identifiera avvikelser i systemets beteende. Programmet utformades för att vara flexibelt och för att tröskelvärden och övervakade parametrar enkelt kan anpassas utifrån förändrade krav eller nya insikter från driftmiljön. Detta gör att systemet kan fortsätta upprätthålla en hög nivå av tillförlitlighet och driftsäkerhet. Nackdelen med denna typ av implementation är att alla mätvärden och trösklar behöver manuellt implementeras av användare.

De telemetripunkter som analyseras i den villkorsbaserade anomalidetekteringen baseras på principen att maximera detekteringsgraden av anomalier i systemet med minimalt antal telemetripunkter. Alltså att största möjliga del av servern täcks med ett fåtal telemetripunkter. Utifrån dessa kriterier så väljs följande:

- `node_load1`: Ger ett sammantaget mått på systemets belastning.
- `node_memory_MemAvailable` och `node_memory_SwapFree`: Tillsammans ger dessa telemetripunkter en bild av tillgängligt minne och eventuella minnesproblem.
- `node_disk_io_time_weightedvda`: Representerar I/O-belastning på disken.
- `process_cpu_seconds_total` och `process_resident_memory_bytes`: Fokuserar på applikationsnivån och fångar avvikande resursförbrukning hos specifika processer.
- `node_procs_running`: Övervakar antalet aktiva processer.
- `http_requests_total`: Täcker applikationens exponerade HTTP-endpoints och används för att identifiera trafikrelaterade anomalier.
- `node_netstat_Tcp_RetransSegs`: Identifierar potentiella nätverksproblem som retransmissioner.

Således övervakas CPU, minne, disk, nätverk, processer och applikationslagret.

Vardera telemetripunkt har ett tröskelvärde som skiljer en anomali från ett normalt mätvärde. Initialt fastställs tröskelvärdena som antingen ett övre eller nedre gränsvärde, beroende på vilken typ av avvikelse som skulle detekteras. Värdena är satta approximativt till 20% från det, utav de normala dataseten, mest extrema medelvärdet för respektive telemetripunkt. Därefter finjusteras vardera tröskelvärde för att förbättra förmågan att särskilja anomalier från normala mätvärden. Se tabell 4.2 för de slutgiltiga tröskelvärdena.

Tabell 4.2: Telemetripunkter och tillhörande tröskelvärden som används i den villkorsbaserade komponenten av det kombinerade ML- och villkorsbaserade anomalidetekteringssystemet.

Telemetripunkt	Tröskelvärde
node_load1	4.0×10^{-2}
node_memory_MemAvailable	9.5×10^{9}
node_memory_SwapFree	1.3×10^{9}
node_disk_io_time_weightedvda	9.0×10^{7}
process_cpu_seconds_total	1.3×10^{3}
process_resident_memory_bytes	1.75×10^{7}
node_procs_running	5.0×10^{0}
http_requests_total	8.5×10^{4}
node_netstat_Tcp_RetransSegs	4.0×10^{4}

4.6.4 Kombinerad ML- och villkorsbaserad anomalidetektering

De två ovan beskrivna tillvägagångssätten av att implementera anomalidetektering testas även att kombineras till en gemensam applikation. Då vissa telemetripunkter i ett dataset väger tyngre än andra vid detektering av anomalier, testas en implementation av både villkor och ML. Villkoren utgör en första nivå och ML-modellen en andra nivå i klassificeringen av data. I första steget klassificeras data utifrån villkor. Ifall det då anses vara en anomali skickas denna data vidare till ML-modellen. Modellen avgör då ifall denna specifika data faktiskt är en anomali eller ej. Om villkoren däremot klassificerar data som normal avslutas klassificeringen utan att denna data passerar genom ML-modellen.

5

Resultat

Detta kapitel presenterar de huvudsakliga resultaten från arbetets tre centrala komponenter: anomalidetektering, logghantering samt visualisering. Dessa delar möjliggör tillsammans en integrerad lösning för att identifiera avvikelser i systemdata och genomföra en grundläggande grundorsaksanalys.

Resultaten redovisas utan tolkning eller värdering, med fokus på vad som utvecklas i praktiken. Diagram, figurer och visualiseringar används för att förtydliga utfallet av respektive komponent.

5.1 Anomalidetektering i mätvärden

I utvecklingen av anomalidetektering används tre olika tillvägagångssätt, som nämns i tidigare avsnitt. Vardera implementation utvärderas nedan utifrån funktionalitet med huvudfokus på prestandan.

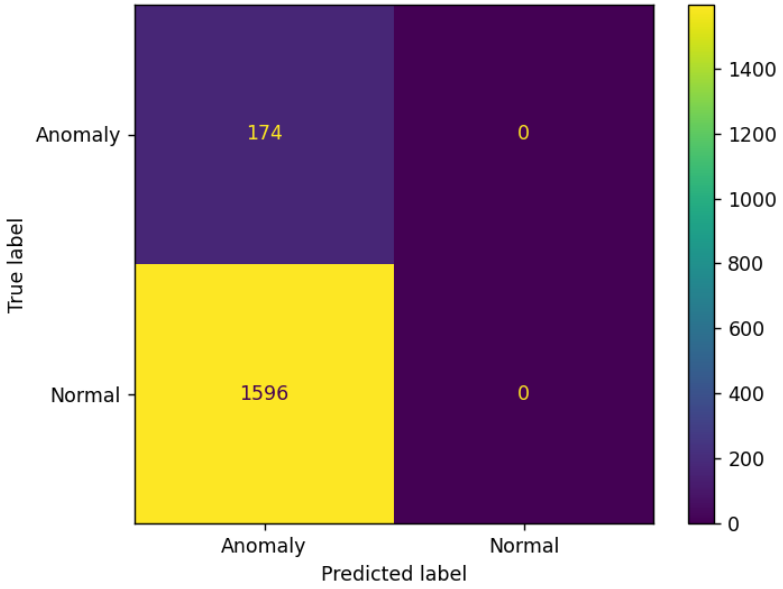
5.1.1 ML-baserad

Första och andra versionen av ML-modellen implementerar Isolation Forest. Modellen tränas samt valideras i båda fallen på syntetisk, genererad, data. I version 1 presterar modellen med en accuracy mellan 98% och 99.79%, vilket verifieras i test ADML1, se appendix A.1. I version 2 implementeras ML med ett nytt tillvägagångssätt där en enskild modell tränas för vardera telemetripunkt. Den data som genereras består av tre olika telemetripunkter, således består systemet av tre ML-modeller. I test ADML2, se appendix A.2, verifieras en accuracy på 99.88%, 100% och 100% för respektive ML-modell. Som framgår ovan presterar ML-modellerna i de två första versionerna med en hög accuracy, dock vid klassificering av syntetisk data.

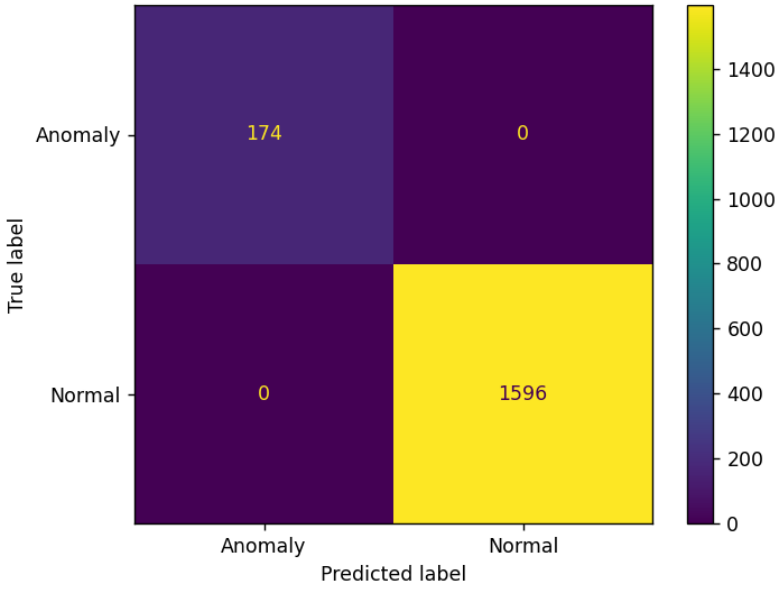
I den fjärde versionen presterar Random Forest bättre än Isolation Forest med högre värden på accuracy, precision, recall och specificity, se tabell 5.1, vilket verifieras i test ADML4, se appendix A.7. Dessutom visar Isolation Forest modellen på en sämre förmåga att urskilja normal data från anomalier, till skillnad från Random Forest modellen, se figur 5.1 och 5.2.

Tabell 5.1: Prestandaevaluering av Isolation Forest samt Random Forest modellerna, version 4.

ML-modell	Accuracy	Precision	Recall	Specificity
Isolation Forest	10%	1%	10%	100%
Random Forest Classifier	100%	100%	100%	100%



Figur 5.1: Isolation Forest confusion matrix



Figur 5.2: Random Forest confusion matrix

I version 5 bibehåller Isolation Forest samma prestanda som i version 4, men Random Forest presterar sämre vid klassificering på närtidsdatan, se tabell 5.2, verifieras i test ADML5, se appendix A.8.

Tabell 5.2: Prestandaevaluering av Isolation Forest samt Random Forest ML-modellerna version 5, utifrån test på närtidsdata.

ML-modell	Accuracy	Precision	Recall	Specificity
Isolation Forest	10%	1%	10%	100%
Random Forest Classifier	10%	1%	10%	100%

5.1.2 Villkorsbaserad

Den villkorsbaserade applikationen för anomalidetektion testas genom stresstester i en kontrollerad miljö för att bestämma dess förmåga att identifiera avvikelser när definierade tröskelvärden nås. Vid tester där olika systemkomponenter belastas, inklusive CPU, arbetsminne, virtuellt minne och I/O visar applikationen en möjlighet att kontinuerligt korrekt detektera flera typer av anomalier baserat på de angivna gränsvärdena. Programmet avläser även när systemet inte utsätts för stress och fortsätter att söka efter en bestämd tid.

```
Metric file timestamp: 2025-05-19 19:58:23
No anomalies detected.
Sleeping for 2 seconds...

Metric file timestamp: 2025-05-19 19:58:25
No anomalies detected.
Sleeping for 2 seconds...

Metric file timestamp: 2025-05-19 19:58:27
No anomalies detected.
Sleeping for 2 seconds...
```

Figur 5.3: Normaldata detektering med villkorsbaserad metod

```
Metric file timestamp: 2025-05-19 19:59:26
Anomalies detected:
  High CPU (iowait) on cpu0: 3.88s in 0.22s
  High CPU (iowait) on cpu1: 5.10s in 0.22s
  High CPU (iowait) on cpu2: 3.33s in 0.22s
  High CPU (iowait) on cpu3: 3.45s in 0.22s
  Load average spiked by 0.55 in 0.22s
  High disk I/O time on dm-0: 10216 ms
  High disk I/O time on vda: 10216 ms
  High disk I/O time on vda5: 9888 ms
Sleeping for 10 seconds...
```

Figur 5.4: Anomalidetektering med villkorsbaserad metod

Utifrån definerade tröskelvärden, identifieras följande typer av avvikelser:

- **CPU-användning (iowait):** Samtliga fyra CPU-kärnor (cpu0–cpu3) visar höga nivåer av fördröjning, Detta tyder på att CPU:erna till stor del var inaktiva i väntan på I/O-operationer.
- **Systembelastning:** Ett abrupt ökat load average observerades, vilket indikerar att flera processer samtidigt väntade på resurser.
- **Disk I/O:** Onormalt långa diskåtkomsttider registrerades för enheter dm-0, vda och vda5 komponenterna med svarstider som överskrider de förväntade tröskelvärdena för diskaktivitet och flaggas därför som avvikelser.

Varje upptäckt anomali loggas med tillhörande tidsstämpel och skickas vidare. Detta möjliggör att användaren snabbt kan identifiera när och var ett mätvärde i systemet avviker från det normala. Sammantaget visar resultaten att den villkorsbaserade metoden är effektiv för att detektera flera typer av anomalier i närtid, särskilt i miljöer där tydliga trösklar kan definieras. Det identifieras även begränsningar med metoden eftersom vissa avvikelser inte visar några varningar. Alla trösklar måste även bestämmas och definieras manuellt i programmet.

5.1.3 Kombinerad ML- och villkorsbaserad

Prestandan av den ML- och villkorsbaserade anomalidetekteringen mäts i termer av recall och specificity, se tabell 5.3, vilket verifieras av test ADMLRB5, se appendix A.10. Accuracy noteras inte eftersom måttet påverkas av andelen anomalier i den data som används vid validering av systemet, vilket i detta fall riskerar att ge ett missvisande resultat.

Tabell 5.3: Prestandaevaluering av kombinerad villkor- och ML-baserad anomalidetektering, version 5.

ML-modell	Recall	Specificity
Isolation Forest	100%	15%
Random Forest Classifier	100%	20%

Efter att en anomali detekterats med den kombinerade modellen sparas anomalimätvärdena framtagna med SHAP-analys korrekt. Mätvärdena sparas i frontend delen för att möjliggöra en korrekt grundorsaksanalys.

5.2 Logghantering


```
[
  {
    "message": "[192.158.0.192]  pam_unix(sudo:session): session opened for user root by kandidatadmin(uid=0)",
    "type": "syslog-tcp",
    "@version": "1",
    "@timestamp": "2025-05-19T14:17:30.997683Z",
    "event": {
      "original": "[192.158.0.192]  pam_unix(sudo:session): session opened for user root by kandidatadmin(uid=0)"
    }
  },
  {
    "message": "[192.158.0.192]  kandidatadmin : TTY=pts/0 ; PWD=/etc ; USER=root ; COMMAND=/bin/nano rsyslog.conf",
    "type": "syslog-tcp",
    "@version": "1",
    "@timestamp": "2025-05-19T14:17:30.994119Z",
    "event": {
      "original": "[192.158.0.192]  kandidatadmin : TTY=pts/0 ; PWD=/etc ; USER=root ; COMMAND=/bin/nano rsyslog.conf"
    }
  },
  {
    "message": "[192.158.0.192]  pam_unix(sudo:session): session closed for user root",
    "type": "syslog-tcp",
    "@version": "1",
    "@timestamp": "2025-05-19T14:17:28.276177Z",
    "event": {
      "original": "[192.158.0.192]  pam_unix(sudo:session): session closed for user root"
    }
```


Figur 5.5: Loggar från Elasticsearch

Komponenten kan skicka förfrågningar till Elasticsearch från det lokala API:et via en etablerad SSH-tunnel och erhålla loggdata från fjärrservern. Testerna visar att loggdata kan hämtas baserat på olika sökparametrar, inklusive filtrering på IP-adress, tidsintervall, typ av logg samt kombinerade sökvillkor. Loggarna exponeras via en FastAPI-endpoint på port 8000.

Efter genomförda tester bekräftas att sökfunktionerna fungerar. Funktionerna möjliggjorde filtrering av loggar utifrån specifik IP-adress (se figur 5.5), tidsintervall samt nyckelord som "error", "timeout" eller liknande.
Vid tester som inkluderade automatiskt genererade felloggar via ett stress-skript kunde komponenten identifiera och returnera dessa loggar korrekt. Den kan även hämta loggar som tidsmässigt sammanfaller med anomalier som detekteras av maskininlärningsmodulen.

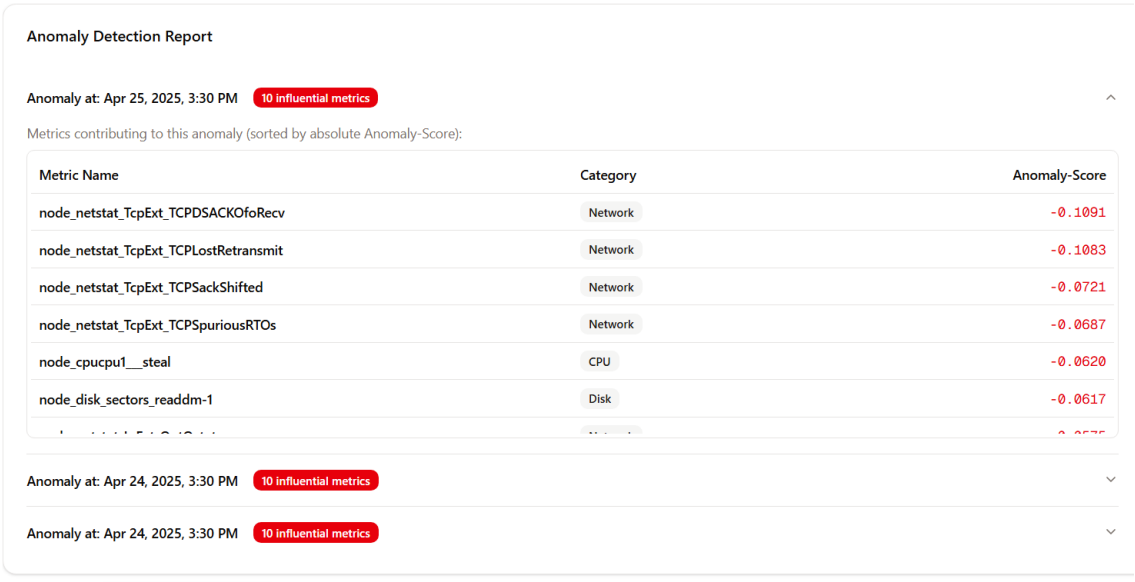
Korrelation mellan loggar och mätdata kan bekräftas genom gemensamma attribut som tidsstämplar och IP-adresser. Loggar kopplade till specifika noder med resursavvikelser, såsom ökad CPU-användning, kan identifieras och hämtas via det utvecklade gränssnittet.

5.3 Visualisering

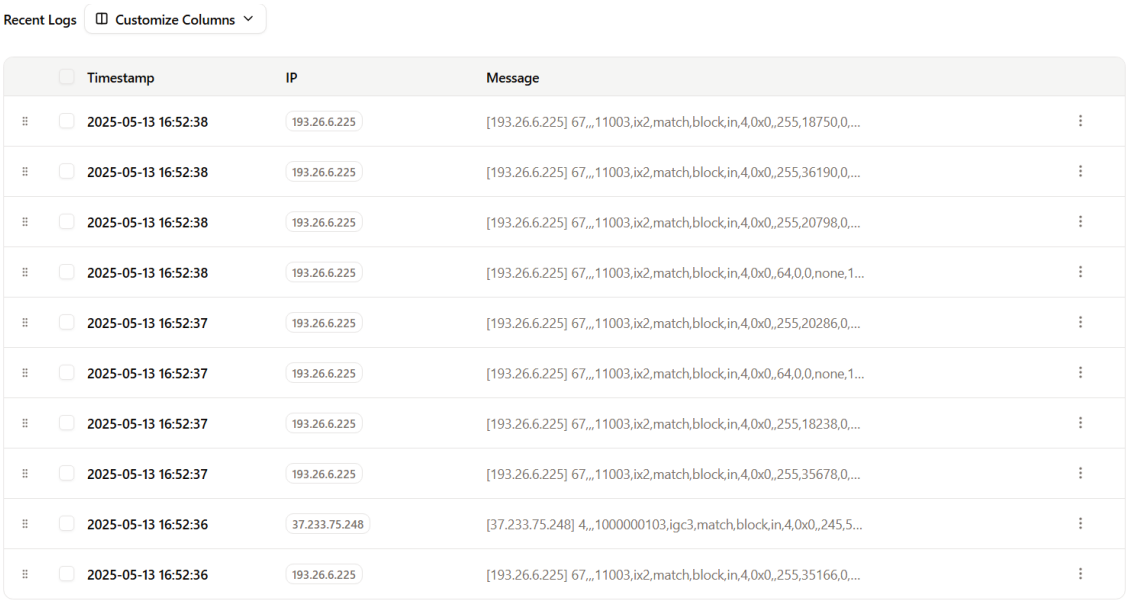
Visualiseringen består av två huvudsakliga komponenter, loggdata och en lista av de senaste anomalierna. När användaren klickar på en av anomalierna visas en lista över de mätvärden som sannolikt orsakade avvikelsen. Därefter kan användaren växla till Grafana för en detaljerad historisk översikt av relevant data.

5.3.1 Egenbyggd webbapplikation

I figurerna 5.5 och 5.6 visas strukturen av webbapplikationens huvudsakliga användningsområden. Då en effektiv grundorsaksanalys kräver information från både telemetri- och loggdata fungerar hemsidan som ett enkelt verktyg där dessa två funktioner kombineras för en systemöversikt.



Figur 5.6: Visualisering av anomalier samt dess mest avvikande mätvärden.



Figur 5.7: Visualisering av loggdata.

5.3.2 Verktygsbaserad vsisualisering i Grafana

När webbapplikationen identifierar en anomali och genererar en varning, ges användaren möjlighet att mer noggrant utforska den i Grafana. Verktyget möjliggör historisk inspektion av datavisualiseringar, vilket gör det möjligt att jämföra den aktuella datapunktens utveckling över tid. Genom att markera relevanta tidsintervall kan användaren fördjupa analysen och fokusera på de mest avvikande mätvärdena.

Kombinationen av en egenutvecklad dashboard med enkel översikt och det avancerade visualiseringsverktyget Grafana kan minska belastningen under utvecklingsarbetet. Funktionaliteten bekräftas i FE1, se appendix A.9, där resultaten bekräftar att länkningen mellan webbapplikationen och specifika Grafana-dashboards fungerar korrekt och uppfyller sitt syfte.

5.4 Slutgiltiga applikationen

Den slutgiltiga applikationen utgör en integrerad plattform för grundorsaksanalys, där olika komponenter samverkar för att effektivisera felsökning genom automatisk dataanalys, visualisering och sammanställning av relevanta telemetri- och loggdata. Flera delar av applikationen är automatiserade med hjälp av Bash-skript, vilket möjliggör att olika analysmoment körs sekventiellt för att reducera manuellt arbete. Bland annat startas insamling av data, förbehandling och modellkörning automatiskt vid schemalagda intervall eller vid nya indata. Denna automatisering minskar fel och sparar tid vid återkommande felsökningsarbete.

I bakgrunden körs en ML-modell kontinuerligt som tar emot och bearbetar inkommande telemetridata där den i närtid detekterar avvikande mönster i exempelvis CPU-användning, nätverkslatens och så vidare. Varje identifierad avvikelse betraktas som en potentiell anomali och skickas vidare till applikationens gränssnitt via en dedikerad API. På användargränssnittet sammanfattas dessa anomalier som en överskådlig rapport, där händelser grupperas efter tidsstämpel. Detta gör det möjligt för användaren att snabbt se när och var avvikelser uppstod.

Varje detekterad anomali länkas till en specifik panel i det underliggande Grafana-dashboardet, vilket ger användaren snabb åtkomst till både historiska trender och aktuell status för den drabbade enheten. Denna direktlänkning eliminerar behovet av att manuellt leta efter relevanta mätpunkter, vilket effektiviserar felsökningen. Parallellt med telemetridata presenteras loggdata som korrelerats utifrån tidsstämpel och IP-adress. De loggar som sammanfaller med anomalin i tid och källa visas direkt i användargränssnittet på samma sida. Detta ger ytterligare sammanhang till händelsen och hjälper till att förstå orsaken bakom avvikelsen.

Slutanvändaren får därmed tillgång till både telemetridata och loggar visualiserade i ett och samma gränssnitt. Detta skapar ett samlat beslutsunderlag för att utföra grundorsaksanalys utan att behöva växla mellan olika verktyg eller källor.

Följaktligen erbjuder den färdiga applikationen en automatiserad och integrerad plattform för att utföra grundorsaksanalys som kombinerar ML, korrelation av loggar med telemetridata samt intuitiv visualisering för att stödja snabbare och mer träffsäkra analyser.

6
Diskussion och slutsats

Detta projekt syftar till att undersöka hur grundorsaksanalys, identifiering samt diagnostisering av systemproblem kan effektiviseras med hjälp av en användarvänlig interaktiv applikation för visualisering av data. Nedan följer en ingående diskussion utifrån det ovan givna resultatet, konkretisering av utvecklingsmöjligheter samt en slutsats vilken kopplar samman diskussionen med syftet.

6.1 Diskussion av resultat

De olika versionerna av anomalidetektering visar en tydlig utveckling från simplare modeller tränade på syntetisk genererad data, till mer komplexa system som integrerar grundorsaksanalys och hanterar närtidsdata på en fjärrserver. De tidiga versionerna, 1 och 2, använder Isolation Forest på syntetisk data och ger en hög accuracy. Det är viktigt att notera att syntetisk data ofta är mer förutsägbar än verklig systemdata, vilket kan bidra till de positiva resultaten. I version 3 och 4, används riktig systemdata och även SHAP används för att möjliggöra grundorsaksanalys, vilket ger en mer praktiskt tillämpbar lösning. Version 4 är särskilt intressant eftersom den implementerar och kombinerar två olika modeller, Isolation Forest och Random Forest. Utav de två modellerna är Random Forest betydligt effektivare på denna data. Isolation Forest har låg accuracy, precision och recall, medan Random Forest når 100% över samtliga mätvärden. Å ena sidan kan den höga prestandan vara ett tecken på att den data som används för validering är för lik modellens träningsdata, å andra sidan tydliggör figur A.4 framför allt att de två anomalidatasetten, anomalidata och validationdata, innehåller olika mätvärden. Fortsättningsvis består validationdata av fler telemetripunkter med avvikande mätvärden och dessutom finns det inga datainstanser där mer än en komponent agerar som en anomali, till skillnad från anomalidata. Däremot uppnår Random Forest i version 4 en orimligt hög prestanda, vilket därför bör beaktas kritiskt.

Version 5 visar en viktig utmaning i praktiken. När modellen används på närtidsdata sjunker prestandan för Random Forest markant, medan Isolation Forest bibehåller en låg prestanda. Detta indikerar att modellen kan vara känslig för förändringar i datadistributionen, vilket är vanligt i system med varierande belastningsmönster. ML-modellens känslighet visar på vikten av kontinuerlig datahämtning och träning av ML-modellen, för att bibehålla en aktuell modell.

Den villkorsbaserade metoden är effektiv när tydliga tröskelvärden kan definieras.

Metoden är särskilt användbar vid övervakning av kända telemetripunkter där normalvärden kan uppskattas. Styrkan ligger i dess förutsägbarhet, enkelhet och att den är lätt att implementera och förstå. Dessa egenskaper gör den till ett bra komplement i miljöer där enkelhet och transparens prioriteras. Dock begränsas metoden av det manuella arbetet som krävs för att sätta tröskelvärden och dess oförmåga att anpassas till nya typer av beteendemönster i system.

Utifrån resultatet av anomalidetekteringen går det att argumentera att en kombination av ML-baserade och regelbaserade metoder är det mest robusta angreppssättet. Den villkorsbaserade metoden kan snabbt upptäcka uppenbara avvikelser, medan den ML-baserade kan identifiera mer komplexa, icke-linjära mönster och bidra med förklarbarhet genom SHAP vilket gynnar grundorsaksanalysen. Vidare framgår det att integreringen av en grundorsaksanalys med SHAP-analys ökar systemets nytta markant. Detta gör det möjligt att inte bara identifiera att något är fel, utan även vad som är fel, ett avgörande steg i praktiska felsökningssituationer.

Logghanteringens resultat visar tydligt hur viktig komponenten är i systemets analysförmåga. Genom att samla och indexera loggar från samtliga noder i närtid kan systemet erbjuda en skalbar och sökbar databas av händelser kopplade till telemetrin i telemetripunkterna, vilket är avgörande för att identifiera och förstå sambandet mellan systemfel och driftavvikelser. Integrationen med mätdata möjliggör inte bara snabb detektion av anomalier, utan också en djupare insikt i deras ursprung. Med logghantering kan specifika felmeddelanden sammanlänkas med telemetridata från anomalidetekteringen och spåras tillbaka till enskilda tjänster eller noder, vilket annars kräver en tidskrävande manuell felsökning. Gränssnittet ger dessutom användarna ett responsivt och skalbart verktyg för att söka bland loggar, även under hög belastning. Sammantaget fungerar logghanteringen inte bara som ett stödverktyg utan som en bärande pelare i systemets användbarhet inom närtidsanalys och grundorsaksanalys.

Visualiseringen är en avgörande komponent för att möjliggöra en effektiv och framförallt intuitiv grundorsaksanalys. Den egenutvecklade webbapplikationen fyller sin funktion som en översiktlig ingångspunkt, där användaren snabbt får en sammanfattning av aktuella anomalier samt associerade mätvärden i närtid från olika noder. Detta skapar ett första lager av insikt som effektivt guidar användaren vidare i analysprocessen. Genom att i webbapplikationen gå vidare till Grafana, där historisk data kan studeras i mer detalj, uppnås en fördjupning av analysen och användaren kan se vad som faktiskt händer med systemet. Användaren får därmed möjlighet att identifiera när och var något går fel, men även förstå utvecklingen av anomalierna över tid och koppla mönster till systembeteenden. Kombinationen av ett lättöverskådligt UI för snabb identifiering och ett avancerat verktyg för datainspektion är både tidsbesparande och användarvänligt. Resultatet bekräftar att integrationen mellan de två komponenterna fungerar pålitligt, vilket stärker konceptets användbarhet i framtida implementationer. Visualiseringslösningen bekräftas därmed som det främsta praktiska stödet i systemets användarvänliga felsökningsstrategi.

Den färdiga applikationen visar tydligt på värdet av att integrera flera tekniker och verktyg i en enda applikation för effektiv grundorsaksanalys. Genom att kombinera villkors- och ML-baserad anomalidetektering, automatiserad datainsamling och korrelation av loggar med närtidstelemetri, skapas ett grundläggande felsökningsstöd som minimerar behovet av manuell hantering. Systemets modularitet samt skalbarhet gör det möjligt att anpassa delar av funktionaliteten vid behov. Den sekventiella automatiseringen via Bash-skript gör att analysen, från datainsamling till visualisering kan genomföras på ett effektivt och enkelt sätt. Gränssnittets styrka ligger i att det presenterar komplex information på ett överskådligt och sammanhängande sätt. Användaren får inte bara en visuell varning om upptäckta anomalier, utan även länkar och tillhörande visualiseringar i Grafana, vilket skapar ett tydligt sammanhang mellan systembeteende och dess orsaker. Detta minskar belastningen vid analys och leder till mer träffsäkra slutsatser. Applikationen fungerar därmed som ett bevis på att en sammanhållen och automatiserad felsökningsplattform kan realiseras med standardiserade verktyg och automatisering. Även om systemet bör betraktas som en prototyp eller konceptvalidering, visar det en tydlig potential för praktisk tillämpning i mer komplexa miljöer. Det slutliga resultatet visar att en kombination av teknik, automatisering och användarfokus kan förbättra förmågan att identifiera och åtgärda systemfel på ett effektivt och användarvänligt sätt.

6.2 Utvecklingsmöjligheter

Då projektet är en bas för utvecklingen av en mer avancerad grundorsaksanalysmodell finns det möjliga utvecklingsområden. Dels områden som planeras att utforskas men som på grund av resursbrist inte är möjliga, dels områden som uppdagas under projektets gång.

6.2.1 Förbättrad tidsanalys

Delproblemet gällande tidsanalys är något som är nedprioriterat och som därav inte nämns i resultatet. Den enda hänsyn som tas till tidsanalys är att all telemetri- och loggdata som samlas in ges en tidsstämpel för att möjliggöra en kronologisk sortering. Anledningen till att delproblemet är nedprioriterat är att även om hög tillförlitlighet på tidstämplar i data är viktigt inom grundorsaksanalys, är det inte lika avgörande i funktionaliteten av produkten som utvecklas. Övriga delproblem är mer integrerade i varandra och kräver färdigställande av varandra innan de kan uppnå full funktionalitet. Att ge tidsstämplar till data är ett enkelt första steg i tidsanalysen vilket även är avgörande för övriga delar av projektet. Men eftersom allt annat i produktutvecklingen fungerar oavsett om tidsstämplarna stämmer eller ej är resterande problem inom tidsanalysen mer en fråga om slutgiltig kvalitet i produkten. Därav blir större delen av problemformuleringen för tidsanalys ett möjligt förbättringsområde.

6.2.2 Implementering av generativ AI

En framtida utvecklingsmöjlighet som ytterligare kan underlätta användarupplevelsen är att implementera en funktion där användaren kan ställa frågor gentemot en Generativ AI som kan beskriva exempelvis vilka datapunkter som är avvikande, samt orsaker som det kan bero på. En sådan funktion är inte inom ramen för detta projekt, utan lämnas därför som en vidare forskning- och utvecklingsmöjlighet.

Exempelvis skulle framtida forskning kunna bestå av att man tar en redan utvecklad modell, som ChatGPT4-o och implementerar inbyggd funktionalitet som en chatruta i webbapplikationen. En annan mer gedigen utvecklingsmöjlighet är att skapa en dedikerad modell just för telemetrianalys. Detta förslag kräver stora resurser för att samla in, annotera och träna en modell för det specifika ändamålet.

6.2.3 Robustare anomalidetekteringssystem

Den anomalidetektering som utvecklas i detta projekt är anpassad och tränad på data från ett specifikt system, vilket gör att den inte är tillräckligt generaliserad för att användas effektivt på en bredare uppsättning system.

En viktig forsknings- och utvecklingsmöjlighet är att skapa ett mer robust och generellt anomalidetekteringssystem. För att uppnå detta krävs att träningsdata samlas in över en längre tidsperiod, från flera olika servrar med varierande dataflöden och arbetsuppgifter. Dessutom måste den insamlade datan annoteras. När detta genomförs kan den nya algoritmen, efter att den tränats, enkelt integreras i den befintliga produkten, vilket förbättrar dess funktionalitet och tillämpbarhet på en större mängd system.

6.2.4 Utökad visualisering

Inom projektets ramar och tidsbegränsning anses visualiseringen vara ett sista steg där alla delkomponenter sammanställs på en centraliserad plats. I och med detta finns stora utvecklingsmöjligheter inom både grafisk visualisering och hantering av loggdata. Den grafiska komponenten av visualiseringen sker i nuläget exklusivt i Grafana, vilket är ett kraftfullt verktyg men saknar anpassningsbarheten som ett skräddarsytt system bidrar med. Utöver anpassningsbarhet förenklas användarvänligheten då all relevant information för en grundorsaksanalys presenteras för användaren på ett ställe. Den största vidareutvecklingspotentialen ligger hos sammanställningen av loggdata med det grafiska gränssnittet. Genom detta kan användaren visuellt hitta avvikelser och med ett knapptryck få tillgång till systeminformation kring samma tidpunkt som avvikelsen.

6.3 Slutsats

Syftet med projektet är att undersöka hur grundorsaksanalys, identifiering samt diagnostisering av systemproblem kan effektiviseras genom utveckling av en inter-

aktiv applikation för användarvänlig visualisering av telemetri- och loggdata. Från resultatet syns nödvändigheten av att kombinera Grafana med en egengjord webbapplikation för att öka användarvänligheten. Dessutom visar resultatet att visualiseringen ger en fokuserad bild som illustrerar vilka datapunkter som är viktigast för anomalidetektering.

Vidare konstateras att det finns forskningsmöjligheter inom ramen för denna produkt. Exempelvis diskuteras hanteringen av tidsförskjutningar och kronologisk sortering av data. Utöver detta rekommenderas implementationen av en generativ AI som i kombination med webbapplikationen och slutprodukten ytterligare skulle kunna hitta mönster i telemetri- och loggdata.

Avslutningsvis konstateras att identifiering och diagnostisering av systemproblem effektiviseras genom att korta felsökningstiden, automatiskt hitta anomalier, presentera anomalier och deras orsak. Detta med hjälp av en enhetlig användarvänlig helhetslösning som förenar datadriven analys med intuitiv och effektiv visualisering.

Källförteckning

[1] Digitalisation World. "Average DDoS attack cost businesses £325,000". [Online]. Tillgänglig: https://m.digitalisationworld.com/news/66951/average-ddos-attack-cost-businesses-325000. (2024).

[2] DNSstuff. "What Is a DDoS Attack and How Can You Detect It With Log Analysis?" [Online]. Tillgänglig: https://www.dnsstuff.com/detect-ddos-attack-with-log-analysis. (2024).

[3] C. Kidd och S. Watts. "What Is Root Cause Analysis? The Complete RCA Guide". [Online]. Tillgänglig: https://www.splunk.com/en_us/blog/learn/root-cause-analysis.html. (2024).

[4] Myndigheten för samhällsskydd och beredskap. "Tidsplan för NIS2-införandet i Sverige". [Online]. Tillgänglig: https://www.msb.se/sv/amnesomraden/informationssakerhet-cybersakerhet-och-sakra-kommunikationer/krav-och-regler-inom-informationssakerhet-och-cybersakerhet/nis-direktivet/tidsplan-for-nis2-inforandet-i-sverige/. (2024).

[5] Myndigheten för samhällsskydd och beredskap. "Det här är NIS2-direktivet". [Online]. Tillgänglig: https://www.msb.se/sv/amnesomraden/informationssakerhet-cybersakerhet-och-sakra-kommunikationer/krav-och-regler-inom-informationssakerhet-och-cybersakerhet/nis-direktivet/det-har-ar-nis2-direktivet/. (2024).

[6] L. Lunter. "Simplify Your Path to NIS2 Compliance with Log Management and SIEM". [Online]. Tillgänglig: https://logmanager.com/blog/it-compliance/nis2-compliance-log-management-siem/. (2025).

[7] M. Y. Eltoweissy, *Distributed Systems (Computers)*, AccessScience, Accessed: Apr. 20, 2025, jan. 2020. DOI: 10.1036/1097-8542.201450. URL: https://www.accessscience.com/content/article/a201450.

[8] B. Andersen och T. Fagerhaug, *Root Cause Analysis: Simplified Tools and Techniques*, 2. utg. Milwaukee, WI: ASQ Quality Press, 2006.

[9] J. Soldani och A. Brogi, "Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey", *ACM Computing Surveys*, årg. 55, nr 3, s. 1–39, febr. 2022. DOI: 10.1145/3501297. URL: https://doi.org/10.1145/3501297.

[10] T. Wang och G. Qi, "A Comprehensive Survey on Root Cause Analysis in (Micro) Services: Methodologies, Challenges, and Trends", *arXiv preprint arXiv:2408.00803*, 2024. URL: https://arxiv.org/abs/2408.00803.

[11] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun and Xiaoyun Li, "MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microser-

vice Environments", i *Proceedings of the Web Conference 2021 (WWW '21)*, New York, NY, USA: ACM, 2021, s. 3087–3098. DOI: 10.1145/3442381.3449905. URL: https://doi.org/10.1145/3442381.3449905.

[12] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, Murali Chintalapati, Saravanakumar Rajmohan and Dongmei Zhang, "Onion: Identifying Incident-Indicating Logs for Cloud Systems", i *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*, New York, NY, USA: ACM, 2021, s. 1253–1263. DOI: 10.1145/3468264.3473919. URL: https://doi.org/10.1145/3468264.3473919.

[13] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqiang Duan and Dan Pei, "Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems", i *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*, New York, NY, USA: Association for Computing Machinery, 2022, s. 996–1008. DOI: 10.1145/3540250.3549092. URL: https://doi.org/10.1145/3540250.3549092.

[14] Hanzhang Wang, Phuong Nguyen, Jun Li, Selcuk Kopru, Gene Zhang, Sanjeev Katariya and Sami Ben-Romdhane, "GRANO: Interactive Graph-Based Root Cause Analysis for Cloud-Native Distributed Data Platform", *Proceedings of the VLDB Endowment*, årg. 12, nr 12, s. 1942–1945, aug. 2019. DOI: 10.14778/3352063.3352105. URL: https://doi.org/10.14778/3352063.3352105.

[15] C. Majors, L. Fong-Jones och G. Miranda, *Observability Engineering: Achieving Production Excellence*. O'Reilly Media, 2022, ISBN: 9781492076391. URL: https://books.google.se/books?id=JmZuEAAAQBAJ.

[16] C. Sridharan, *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, 2018, ISBN: 9781492033424. URL: https://books.google.se/books?id=07EswAEACAAJ.

[17] Prometheus. "Overview | Prometheus". [Online]. Tillgänglig: https://prometheus.io/docs/introduction/overview/. (2025).

[18] Grafana. "About Grafana | Grafana". [Online]. Tillgänglig: https://grafana.com/docs/grafana/latest/introduction/. (2025).

[19] Cloudflare. "What is SSH? | Secure Shell (SSH) protocol". [Online]. Tillgänglig: https://www.cloudflare.com/en-gb/learning/access-management/what-is-ssh/. (2025).

[20] D. K och J. Skelton. "Anomaly Detection Using Isolation Forest in Python". [Online]. Tillgänglig: https://www.digitalocean.com/community/tutorials/anomaly-detection-isolation-forest. (2024).

[21] Scikit-learn. "IsolationForest". [Online]. Tillgänglig: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. (2025).

[22] BuiltIn. "Random Forest: A Complete Guide for Machine Learning". [Online]. Tillgänglig: https://builtin.com/data-science/random-forest-algorithm. (2025).

[23] K. Peffers, T. Tuunanen, M. A. Rothenberger och S. Chatterjee, "A design science research methodology for information systems research", *Journal of Management Information Systems*, årg. 24, nr 3, s. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302.

[24] F. T. Liu, K. M. Ting och Z.-H. Zhou, "Isolation Forest", i *2008 Eighth IEEE International Conference on Data Mining*, 2008, s. 413–422. DOI: 10.1109/ICDM.2008.17.

[25] E. F. Agyemang, "Anomaly detection using unsupervised machine learning algorithms: A simulation study", *Scientific African*, årg. 26, e02386, 2024, ISSN: 2468-2276. DOI: https://doi.org/10.1016/j.sciaf.2024.e02386. URL: https://www.sciencedirect.com/science/article/pii/S2468227624003284.

[26] Q. Yang, W. Cao, G. Chen, J. Wu, Z. Tao och J. Lv, "A Study on Anomaly Detection in Massive Multivariate New Energy Data Based on Isolation Forest", i *2024 14th International Conference on Power and Energy Systems (ICPES)*, 2024, s. 341–346. DOI: 10.1109/ICPES63746.2024.10856614.

[27] R. Primartha och B. A. Tama, "Anomaly detection using random forest: A performance revisited", i *2017 International Conference on Data and Software Engineering (ICoDSE)*, 2017, s. 1–6. DOI: 10.1109/ICODSE.2017.8285847.

[28] A. Gupta och R. Simon, "Enhancing Security in Cloud Computing With Anomaly Detection Using Random Forest", i *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2024, s. 1–6. DOI: 10.1109/ICRITO61523.2024.10522227.

A

Appendix

A.1 Testning av anomalidetektering med ML-modell, version 1

Datum: 9/3 Tid: 14:30

Testfalls ID: ADML1

Test Objekt: continuous_anomaly_detector_v1.py

Testarens Namn: Edvin Bengtsson

Beskrivning:

Testet genomfördes med syfte att dels evaluera prestandan av Isolation forest ML-modellen samt för att testa av appliceringen av ML-algoritmen, version 1. Detta genomfördes genom att python-skriptet *continous_anomaly_detector_v1.py* exekverades, varav modellen tränades och sedan förutsåg anomalier i ett testdataset. Både träningsdata och testdata var genererad av ett python-skript. Tidsstämplarna för de identifierade anomalierna skrevs ut. Därefter kördes programmet ytterligare en gång, där modellen förutsåg anomalier i ett nytt testdataset med 100 instanser av telemetridata, var och en med en angiven tidpunkt. Detta dataset motsvarade 10% av både tränings- och det första testdataseten i storlek, men med en lägre kontamineringsgrad på 1%, jämfört med 10% i de tidigare dataseten.

Fortsättningsvis sparas och hämtas genererad data i JSON-filer, för att simulera verklig data som sparas och uppdateras i JSON-filer. Programmet testas således inte bara för ML-modellens prestanda, utan också för huruvida denna process med datahanteringen fungerar.

Resultat:

Programmet exekverades som förväntat samt utan varningar eller fel. Vid första iterationen uppnådde ML-modellen en accuracy på 99.79% och på andra iterationen en accuracy värde på 98%. Fortsättningsvis skrevs och hämtades data korrekt från JSON-filerna.

Analys:

ML-modellen presterade med en hög accuracy, vilket troligtvis till stor del beror på att tränings- och testningsdata är relativt lika varandra, vilket således bör ha ökat prestandan av ML-modellen, trots att Isolation Forest är en unsupervised learning algoritm. Fortsättningsvis visade testet att python-skriptet som skrev data till, och hämtade data från JSON-filer fungerade. Ytterligare är en tydlig förbättring av ML-modellen att den tränas på verklig data.

A.2 Testning av anomalidetektering med ML-modell, version 2

Datum: 12/3 Tid: 10:00

Testfalls ID: ADML2

Test Objekt: continous_anomaly_detector_v2.py

Testarens Namn: Edvin Bengtsson

Beskrivning:

Syftet med testet var att evaluera prestandan av andra versionen av implementeringen av Isolation Forest modellen, där en modell tränades för vardera telemetripunkt. Således tränades tre modeller, en för *go_memstats_heap_released_bytes*, en för *go_memstats_sys_bytes* och en för *process_max_fds*. Ytterligare var denna genererade data, jämfört med den data som genererades i version 1, mer lik den verkliga telemetridata. Således testades även hur väl denna telemetridata kunde konverteras till användbar data i form av dataframes. Testet genomförde genom att python-skriptet *continous_anomaly_detector_v2.py* exekverades. Träningsdata valdes till storlek 100 000 enskilda data instanser med 0.01% kontamineringsgrad, medan testdata valdes till storlek 20 000 instanser med kontamineringsgrad 1%.

Resultat:

Data genererades korrekt samt konverterades korrekt från JSON-format till en dataframe. Se tabell A.1 för de tre ML-modellernas prestanda evaluering.

Tabell A.1: Prestanda evaluering av ML-modeller, version 2

ML-modell, utefter dess telemetridata	Accuracy	Recall	Specificity
go_memstats_heap_released_bytes	99.88%	100%	88.68%
go_memstats_sys_bytes	100%	100 %	100%
process_max_fds	100%	100%	100%

Analys:

ML-modellen presterade med höga värden på alla tre mätvärden, men troligtvis beror detta av att tränings och testningsdata har många likheter. Fortsättningsvis tog tränings och förutsägelse processen av modellerna lång tid, då detta skulle göras för flera modeller. Således bör endast en modell användas för all data. Utöver prestanda evalueringen, lyckades den mer verklighets-representativa telemetridata konverteras från JSON-format till en dataframe.

A.3 Testning av datahämtning genom Node Exporter samt formatering med Prom2json

Datum: 13/3 Tid: 11:00

Testfalls ID: DH1

Test Objekt: gather_data.sh

Testarens Namn: William Arkhult

Beskrivning:

Testet genomfördes för att bestämma om datahämtningen via SSH fungerar korrekt, samt om dessa mätvärden kan sparas ned på ett lätthanterligt och formaterat sätt för att användas i anomali detektering i närtid.

Resultat:

Testet resulterade i en effektiv datahämtning samt processering som är redo att analyseras med ML-algoritmen. SSH-tunneln mellan fjärrservern och den lokala maskinen som kör skriptet möjliggör länkning av portar och datahämtning. Data som hämtades processerades sedan med hjälp av prom2json som formaterar rådata från prometheus, till lätthanterliga JSON-filer. Vidare, användes ytterligare ett python-skript för att formatera och strukturera dessa JSON-filer för att öka läsbarheten samt göra det manuella arbetet och felsökningen mer effektivt.

Analys:

Resultatet kan tolkas som positivt då funktionalitet och prestanda mötte förväntningarna. Genom att öppna en ssh-tunnel och länka lokala portar till fjärrserverns port, nu är det möjligt att kontinuerligt både spara ner mätdata samt kontinuerligt läsa mätvärden för vidare databearbetning och analysering.

A.4 Testning av villkorsbaserad anomalidetektering

Datum: 21/4 Tid: 11:00

Testfalls ID: ADRB1

Test Objekt: Rule_based_anomaly_detection.py

Testarens Namn: William Arkhult

Beskrivning:

Testet genomfördes för att bestämma om anomalier kan detekteras genom att sätta förutbestämda tröskelvärden för alla mätvärden. Det skapades ett skript för att stressa olika komponenter på fjärrdatorn däribland processor, minne, virtuellt minne och IO. Under testet stressade de olika komponenterna under 60 sekunder var med inkrementellt ökande stresslaster. Samtidigt som fjärrdatorn stressades, hämtades det nya mätvärden var 15 sekund med hjälp av node exporter och prometheus. Dessa mätvärden jämfördes sedan villkorligt med tröskelvärden som ansågs vara "normala" för att detektera avvikelser.

Resultat:

Testet resulterade i att datorn påfrestades av stress funktionerna vilket avspeglades i anomali detekteringen. I figur A.1 illustreras bland annat hur minnet allokeras på ett ovanligt sätt på grund av –vm och –vm-bytes stressningen nätverksstacken upplever också förändringar i TCP/ICMP-aktivitet vilket kan bero på en stressad IO. Vidare, påverkas delat minne (Shmem) genom att det används annorlunda på grund av stressningen.

```
 but RandomForestClassifier was fitted with feature names
  warnings.warn(
SHAP values for anomaly saved to ../../frontend/app/dashboard/mldata.json
Anomaly detected at timestamp ['2025-05-14 11:13:31']
The features with the greatest anomaly score, for the detected anomaly, are:
node_memory_DirectMap4k: -0.0909
node_netstat_TcpExt_TCPSackShifted: -0.0909
node_netstat_Icmp_InEchos: -0.0909
node_netstat_TcpExt_TCPFACKReorder: -0.0909
node_netstat_TcpExt_TCPDSACKIgnoredOld: -0.0909
node_netstat_IcmpMsg_InType3: -0.0909
node_netstat_IcmpMsg_OutType0: -0.0909
node_netstat_TcpExt_TCPFromZeroWindowAdv: -0.0909
node_netstat_IcmpMsg_InType8: -0.0909
node_memory_Shmem: -0.0201
Anomaly prediction completed successfully.

Waiting 10 seconds before next cycle...
```

Figur A.1: Anomali detektering med villkorsbaserad modell

Analys:

Resultatet av testet kan tolkas som positivt då stressen påverkade vissa utav mätvärdena och applikationen reagerade med varningar på ett delvist korrekt sätt. Stressning av virtuellt minne, minnesskrivning samt nätverksaktivitet påverkades av stressen korrekt, däremot gav anomalidetekteringsapplikationen inga varningar för cpu påfrestningar. Detta är något som var förväntat innan testet utfördes, men som inte inträffade.

A.5 Hämtning av loggar från Elasticsearch via SSH-tunnel

Datum: 21/4 Tid: 10:00

Testfalls ID: ELASTIC01

Test Objekt: elastic-backend

Testarens Namn: Muhammad Tariq Khalidee

Beskrivning:

Testet genomfördes i syfte att undersöka om det är möjligt att automatiskt öppna en SSH-tunnel från den lokala utvecklingsmiljön till en fjärrmaskin och därefter hämta loggdata från Elasticsearch-instans som körs på fjärrmaskinen. Testet innebar att ett Python-baserat backend-system utvecklades för att vid uppstart etablera en SSH-tunnel till den angivna fjärrservern. När tunneln är öppen skickas förfrågningar till Elasticsearch via den lokala porten som är vidarebefordrad till fjärrmaskinens port 9200. På så sätt kan applikationen hämta loggar utan att behöva exponera Elasticsearch direkt över internet.

Resultat:

Testet lyckades. Det var möjligt att hämta loggar från Elasticsearch via SSH-tunneln på ett tillförlitligt sätt. Applikationen kunde upprätta anslutningen och ta emot svar på loggförfrågningar. Förfrågningar baserad på tidsintervall returnerade förväntade resultat. Detta visar att lösningen fungerar tekniskt och att den valda metoden för fjärråtkomst till loggar är genomförbar i praktiken.

Analys:

Det konstaterades att loggarna som hämtas behöver förfinas för att underlätta analys och spårbarhet. I det aktuella testet innehöll loggarna inte en urskiljbar IP-adress eller andra avgörande identifikationer. Detta blir särskilt problematiskt i miljöer med flera olika noder eller enheter som skickar loggdata till samma Elasticsearch-instans. För att möjliggöra effektiv felsökning och källidentifiering behöver loggmeddelandena standardiseras och inkludera metadata såsom ursprunglig IP-adress, nodnamn eller annan entydig identifiering. Vidare krävs förbättrade filterfunktioner i backend-logiken för att enklare kunna isolera relevant data i en komplex miljö.

A.6 Test av Elastic-backend, filtreringsfunktioner och FastAPI

Datum: 5/5 Tid: 15:00

Testfalls ID: ELASTIC02

Test Objekt: elastic-backend

Testarens Namn: Muhammad Tariq Khalidee

Beskrivning:

Detta test syftade till att utvärdera backend-logiken i sin helhet, med fokus på flera specifika filtreringsfunktioner som utvecklats för att hämta loggar från Elasticsearch. Systemet använder FastAPI som webbramverk och exponerar ett REST-baserat API via den lokala porten localhost:8000. Backend är kopplat till Elasticsearch genom en SSH-tunnel och fungerar som ett mellanlager som tar emot HTTP-förfrågningar och skickar dem vidare till Elasticsearch och returnerar resultaten.
Flera nyckelfunktioner granskades och verifierades. Bland annat:

- Hämtning av ett visst antal senaste loggar (`get_logs`).
- Filtrering baserat på tidsintervall (`get_logs_by_time`).
- Filtrering baserat på IP-adress (`get_logs_by_ip`).
- Kombinerad filtrering på tid och IP-adress (`get_logs_by_time_and_ip`).
- Hämtning av loggar med felmeddelanden eller misslyckanden (`get_error_fail_logs`).

Varje funktion testades med hjälp av FastAPI-gränssnittet på `http://localhost:8000/docs/`.

Resultat:

Alla testade funktioner fungerade som förväntat. Backend-servern kunde ta emot och tolka HTTP-anrop korrekt, vidarebefordra dem till Elasticsearch och returnera relevanta loggposter. De olika filtreringsfunktionerna visade sig vara effektiva, och loggar kunde filtreras exakt efter angivna parametrar. Systemet svarade snabbt och stabilt, vilket indikerar att både FastAPI och Elasticsearch-integrationen är korrekt implementerade.

Analys:

I detta test användes en förbättrad uppsättning loggar där rsyslog på fjärrmaskinen hade konfigurerats för att inkludera identifierande information i loggmeddelandena, framför allt IP-adress för avsändaren. Detta var en avgörande förbättring jämfört med tidigare tester, där loggarnas ursprung inte alltid var tydligt. Rsyslog-konfigurationen har därmed visat sig vara ett viktigt steg för att underlätta felsökning och RCA. Dessutom kan nu elastic-backend fungera som ett server som kan ta emot och genomföra förfrågningar från frontend-gränssnittet.

A.7 Testning av anomalidetektering med ML-modell, version 4

Datum: 12/5 Tid: 14:30

Testfalls ID: ADML4

Test Objekt: ml_anomaly_finder.py

Testarens Namn: Edvin Bengtsson

Beskrivning:

Syftet med testet var att evaluera prestandan av fjärde versionen av en anomali detekterings ML-modell. I denna versionen implementeras ett system som tränar och validerar en generisk ML-modell. Både Isolation Forest och Random Forest Classifier tränas med optimerade hyperparameter värden, enligt tabell A.2. Hyperparametrarna som optimerats är:

- **max_features:** Anger maximala antalet features, alternativt hur stor andel av alla features, som används per träd.
- **max_samples:** Anger maximala antalet data instanser, alternativt hur stor andel av all data, som får användas per träd.
- **n_estimators:** Anger antalet träd som ska ingå per skog.

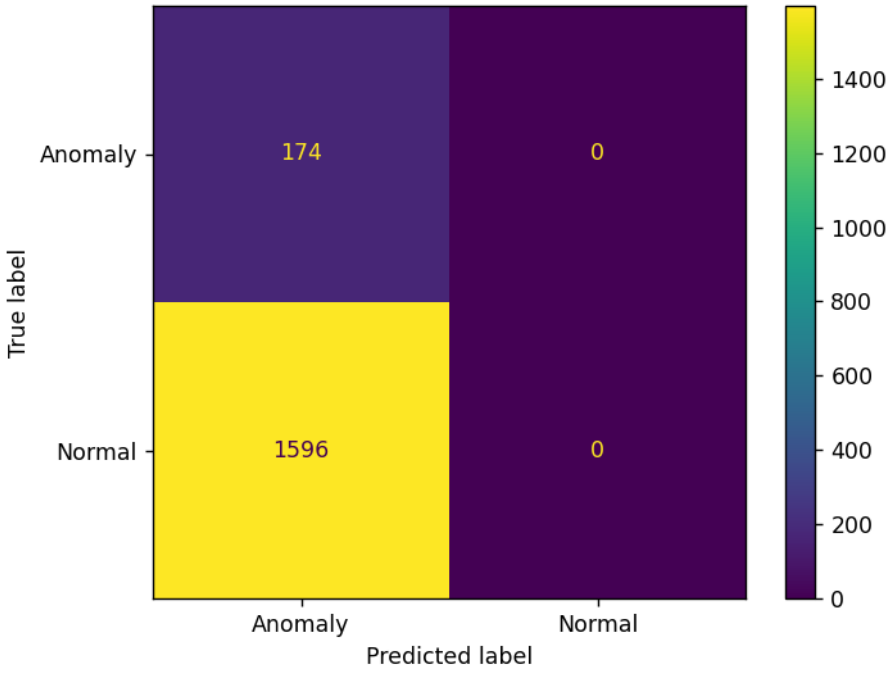
Båda modellerna tränades på *anomalydata, normaldata1, normaldata2* och *normaldata3* samt testas på *validationdata* och *normaldata4*. Testet genomförs genom att testfunktionen i python-skriptet ml_anomaly_finder.py exekveras.

Tabell A.2: Optimerade hyperparameter värden för ML-modellerna, version 2

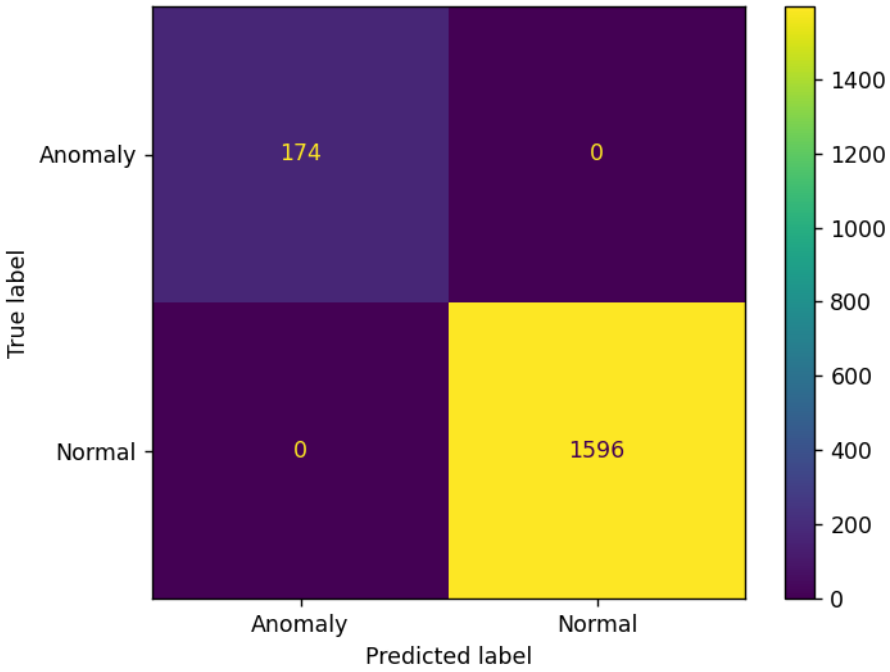
ML-modell	max_features	max_samples	n_estimators
Isolation forest	0.1	256	200
Random forest classifier	0.1	256	100

Resultat:

Modellernas prestanda evaluerades genom att visualisera respektive modells confusion matrix, se figur A.2 för Isolation forest och figur A.3 för Random forest classifier, samt genom att beräkna accuracy, precision, recall och specificity, se tabell A.3. Dessutom visar figur A.4 hur träningsdata förhåller sig till valideringsdata. Valideringsdata utgörs av ett normalt dataset, *normaldata4*, och ett anomali dataset, *validationdata*. Resterande data ingår i träningsdata, där alla dataseten är insamlade under olika tidpunkter och således uppdelade i separata dataset.



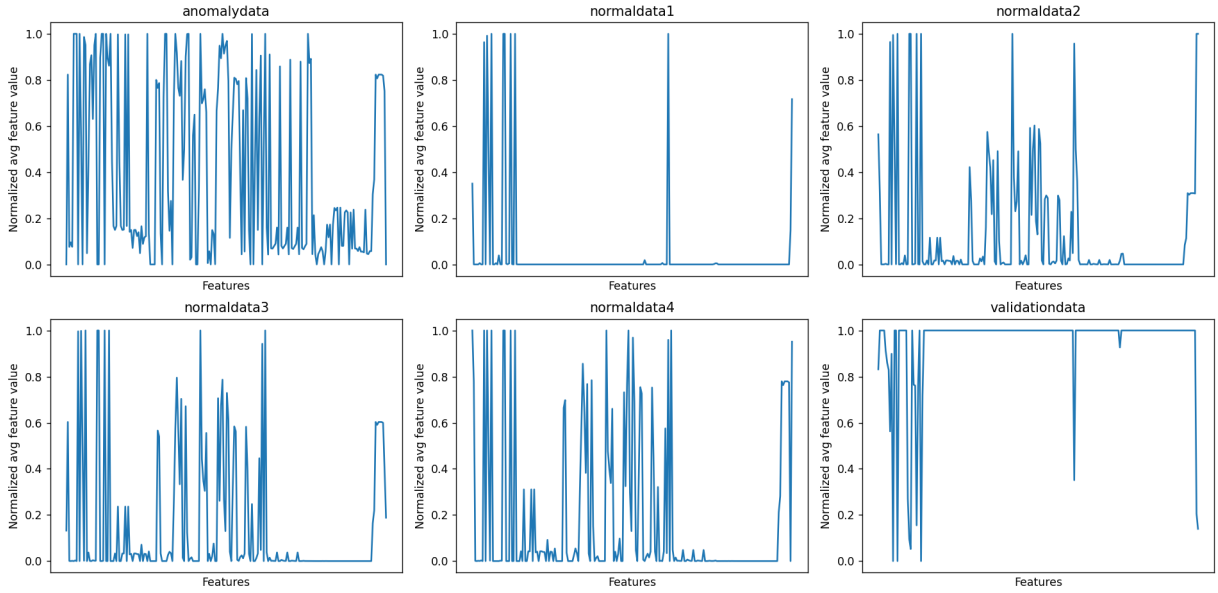
Figur A.2: Isolation Forest confusion matrix



Figur A.3: Random Forest confusion matrix

Tabell A.3: Prestandaevaluering av Isolation Forest samt Random Forest ML-modellerna, version 4.

ML-modell	Accuracy	Precision	Recall	Specificity
Isolation Forest	10%	1%	10%	100%
Random Forest Classifier	100%	100%	100%	100 %



Figur A.4: Grafer över normaliserade medelvärdet för alla features för vardera dataset.

Analys:

Random Forest modellen presterade bättre än Isolation Forest modellen. I tabellen framgår det att Isolation Forest modellen presterade svagt för alla mätvärden förutom specificity, vilket förklaras av dess confusion matrix som visar att modellen ej kunde identifiera normala värden. Däremot klassificerade Random Forest modellen all data korrekt, vilket framgår både i tabellen och av dess confusion matrix. Å ena sidan är denna prestanda osannolik och kan fås då en supervised learning-modell testas på valideringsdata vilken är samma som träningsdata. Å andra sidan visar graferna över normaliserade värdena att den valideringsdata som använts ej är densamma som den träningsdata som använts. Således bör Random Forest modellen användas istället för Isolation Forest modell.

A.8 Testning av kontinuerlig anomalidetektering med ML-modell, version 5

Datum: 13/5 Tid: 12:00

Testfalls ID: ADML5

Test Objekt: run_anomaly_detection.py

Testarens Namn: Edvin Bengtsson och William Arkhult

Beskrivning:

Testet genomfördes med syfte att evaluera prestandan av både Isolation Forest och Random Forest modellerna vid kontinuerlig anomalidetektering på närtidsdata, både normal och anomalidata. Testet genomför genom att exekvera skriptet *run_anomaly_detection.py*. Båda modellerna tränades på samma data som i test ADML4, dvs *anomalydata, normaldata1, normaldata2* och *normaldata3*. I detta test utvärderas endast modellernas accuracy.

Resultat:

Båda ML-modellerna presterade ej som förväntat. Modellerna klassificerade all data som anomalier, både normal data och anomalidata. Således gavs en accuracy som endast beror på andelen anomalier kontra normal data. I tabell A.4 framgår mätvärdena för detta testfall, för båda modellerna.

Tabell A.4: Prestandaevaluering av Isolation forest samt Random Forest ML-modellerna, version 5.

ML-modell	Accuracy	Precision	Recall	Specificity
Isolation Forest	10 %	1 %	10 %	100 %
Random Forest Classifier	10 %	1 %	10 %	100 %

Analys:

Utifrån resultatet framgår det att båda ML-modellerna ej kan urskilja normal data från anomalier vid kontinuerliga tester av närtidsdata. Orsaken till detta kan bero på att modellernas träningsdata ej korrekt representerar den aktuella telemetridata från servern, vilket troligen kan förklaras av att servern inte beter sig på samma sätt som då träningsdata samlades in. En lösning kan vara att samla in ny träningsdata, men en annan mindre tidskrävande lösning är att implementera en anomalidetektering som bygger på både ML och villkor.

A.9 Testning av UI på centraliserad hemsida

Datum: 14/5 Tid: 20:10

Testfalls ID: FE1

Test Objekt: Frontend

Testarens Namn: Hampus Åkerlund & David Andersson

Beskrivning:

Testandet utfördes i syfte att färdigställa frontend-designen, både funktionellt och visuellt. Testandet kollade på vilka knappar som har funktionalitet och som leder dit de bör.

Resultat:

Efter testandet kan slutsatsen dras att alla knappars funktionalitet är som förväntat och leder användaren som förväntat. Det upptäcktes även knappar som inte längre används och åtgärdas genom att ta bort.

Analys:

Från resultatet kunde funktionaliteten verifieras av de knappar som behålls. Det fanns vissa knappar som inte ledde någonstans dessa togs bort för att de inte hade någon effekt på användarupplevelsen eller produkten.

Ett förbättringsförslag efter testandet är att ändra tidsspannet på telemetridata som visas genom Grafana till senaste 30 minuter istället för senaste dygnet.

A.10 Testning av kombinerad kontinuerlig anomalidetektering med ML-modell

Datum: 15/5 Tid: 18:30

Testfalls ID: ADMLRB5

Test Objekt: run_anomaly_detection.py

Testarens Namn: Edvin Bengtsson och William Arkhult

Beskrivning:

Testet genomfördes med syfte att evaluera prestandan av anomalidetekteringen bestående av en kombination av villkor och ML, vid kontinuerlig anomalidetektering på närtidsdata, både normal och anomali data. Testet genomfördes genom att skriptet *run_anomaly_detection.py* exekverades. Båda modellerna tränades på samma data som i test ADML4, dvs *anomalydata, normaldata1, normaldata2* och *normaldata3*. I detta test utvärderades endast modellernas recall och specificity. Fortsättningsvis evaluerades både Isolation Forest samt Random Forest modellerna var för sig.

Resultat:

Kombinationen av villkor och ML klassificerar alltid normal data som normal, både med Isolation Forest och med Random Forest. Dock detekteras endast ett fåtal av anomalierna av systemet. I tabell A.5 presenteras specificity samt recall för de två versionerna av kombinerad anomalidetektering.

Tabell A.5: Prestandaevaluering av villkor- och ML-baserad anomalidetektering, version 5.

ML-modell	Recall	Specificity
Isolation Forest	100 %	15 %
Random Forest Classifier	100 %	20 %

Analys:

Utifrån resultatet ovan presterar den kombinerade implementationen av villkor och ML bättre för anomalidetektering än vad endast ML-versionen gör. Utav den kombinerade implementationen presterar Random Forest bättre än Isolation Forest.

Jämfört med en ML-modell är villkoren enklare att anpassa efter nya förhållanden i den data som övervakas. En ML-modell behöver tränas på nytt med ny data. För villkoren räcker det däremot att ändra gränsvärdet för de utvalda features. Således kan nya förhållanden i den data som systemet övervakar avsevärt påverka ML-modellen, medan villkoren enkelt kan manuellt anpassas efter de nya tröskelvärdena.

A.11 Testning av slutprodukt

Datum: 16/5 Tid: 14:00

Testfalls ID: SLP

Test Objekt: Slutprodukt

Testare: Hela gruppen

Beskrivning:

Testet genomfördes i syfte att verifiera applikationens funktionaliteter vid simulerade driftstörningar och hög belastning på systemkomponenter. Särskilt fokus lades på applikationens förmåga att:

- Upptäcka genererade anomalier via ML-modulen.
- Visa anomalier korrekt i det grafiska gränssnittet.
- Korrelera dessa anomalier med relevanta felloggar.
- Möjliggöra vidare analys genom länkning till motsvarande graf i Grafana dashboard.

För att initiera testet användes ett automatiserat Bash-skript för att simulera verkliga scenarier. Skriptet utförde manuell belastning av olika systemkomponenter såsom CPU och RAM samt genererade felloggar.

Resultat:

ML-modulen identifierade vissa av de skapade belastningarna som avvikande och markerade dessa som anomalier med korrekt tidsstämpel. Anomalierna visualiserades i frontend-komponenten tillsammans med motsvarande felloggar. Vid klick på anomalipunkt i gränssnittet omdirigerades användaren korrekt till Grafana-panelen för vidare analys. Panelen visade både historisk och aktuell utveckling av den telemetriska mätserien.

Analys:

Testet validerar att hela systemets arbetsflöde, från datainsamling till visuell presentation, fungerar som tänkt under belastning. Kombinationen av ML-baserad detektion, loggkorrelation och interaktiv visualisering bekräftades. Samtliga funktionella krav relaterade till detta testfall uppfylldes.