OpenTelemetry Observability Dashboard

P17-T1 ThoughtData Project

Kian Rezaeinejad, Dakobie Garcia,

Daniel Ekema, Jaytin Bolling

IT Capstone 4983: W01, W02

Donald Privitera

December 3rd, 2025

TABLE OF CONTENTS

Executive Summary 3

Background ..... 4 - 6

Business and Project Background 4

Project Scope, Objectives, and Deliverables 5

Technical Background 6

Technical Summary and Objectives/ achievements ..... 7 – 9

Outcomes and Achievements 8

Achievements by Objective ..... 8 - 9

Project Planning and Management Summary ..... 9 - 13

Overview of Project Management 9

Project Process by Stages and Milestones 10

Team Contribution Summary ..... 11 -12

Workload Summary ..... 12 - 13

Team Reflection on Project Experience ..... 13 - 15

Project Success Factors 13

Team Collaboration and Communication ..... 13 -14

Challenges Encountered 14

Areas for Improvement 15

What Would Be Done Differently 15

Recommendations for Future Teams 15

Bibliography ..... 16 - 17

Appendices 18

Executive Summary

This project developed an observability system that demonstrates how modern applications generate and analyze telemetry. Many organizations use distributed architectures, where microservices communicate across multiple systems. These environments require observability to understand performance and identify failures. Observability relies on three types of telemetry: metrics, which measure performance; logs, which record events; and traces, which show how a request moves across services. OpenTelemetry is widely adopted because it standardizes how these signals are collected and exported.

The project created a complete telemetry pipeline using the OpenTelemetry Collector, Elasticsearch, and Grafana. The collector received data through OTLP protocols and applied processing steps such as resource detection, attribute filtering, and batching before exporting telemetry to external systems. This configuration is documented in the project's collector file. A Linux server provided by the sponsor hosted the demo application and dashboard environment. Access instructions, server endpoints, and authentication details are documented in the team's server guide. Grafana dashboards were built using Elasticsearch queries and included panels for error trends, host activity, service-level patterns, and message analysis. These dashboards are defined in the provided JSON file. The completed system demonstrates the full observability workflow: generating telemetry, processing it through a centralized collector, storing it in Elasticsearch, and visualizing it in Grafana. The project provides a reproducible reference model that can support future development, additional data sources, and expanded alerting or performance analysis features. The final implementation reflects industry practices and offers a stable foundation for continued work in modern observability systems.

Background

Business and Project Background

Modern organizations rely on distributed systems, often built using microservices. A microservices architecture divides an application into smaller, independent services that communicate over a network. While this design improves scalability and flexibility, it introduces challenges in understanding system behavior, identifying failures, and diagnosing performance issues. As systems become more complex, observability becomes essential for maintaining reliability. Observability refers to the ability to understand the internal state of a system based on external outputs such as logs, metrics, and traces.

Industry adoption of observability tools has grown because they allow engineering teams to detect anomalies, monitor service health, and reduce incident resolution time. Companies such as Google, Microsoft, and Uber support OpenTelemetry because it standardizes data collection and works across diverse platforms without locking organizations into a single vendor. OpenTelemetry enables consistent insight across heterogeneous environments, which is particularly valuable in cloud-native systems.

This project was initiated to create a complete observability demonstration suitable for instructional and operational use. The goal was to design and implement an end-to-end telemetry pipeline that collects data from a running application, processes it through the OpenTelemetry Collector, stores it in Elasticsearch, and presents it through Grafana dashboards. The project addresses the need for an accessible reference implementation that future KSU teams and industry partners can replicate or extend.

Project Scope, Objectives, and Deliverables

The project scope centered on constructing a functional observability environment that integrates the three core telemetry signal types:

• **Metrics**: numerical measurements that describe performance (e.g., request count, latency, CPU usage).
• **Logs**: detailed records of events or system actions.
• **Traces**: sequences of operations that follow a request through multiple services.

The primary objectives included:

1. **Designing a telemetry pipeline** using the OpenTelemetry Collector configured with receivers, processors, and exporters.
2. **Deploying a demo application** on the sponsor-provided Linux server to generate logs, metrics, and traces.
3. **Configuring Elasticsearch and Grafana** to store and visualize telemetry data.
4. **Developing dashboards** that present error trends, service performance, and host-level activity.
5. **Documenting the system** so another team can reproduce or extend the implementation.

Key deliverables included the collector configuration file containing resource detection and transformation rules, the Grafana dashboard JSON, and the server connection and usage guide

Technical Background

OpenTelemetry is an open-source framework for generating, collecting, and exporting telemetry. It provides standardized APIs and SDKs that allow applications to emit logs, metrics, and traces in a consistent format. The OpenTelemetry Collector serves as a central component in the telemetry workflow. It receives data through multiple protocols, processes it using pipelines, and exports it to storage or analysis platforms. In this project, the collector was configured with OTLP receivers for both gRPC and HTTP ingestion, processors for batching and attribute cleanup, and exporters for forwarding data to Grafana Cloud and Elasticsearch as specified in the project's configuration file

Grafana is a visualization platform that converts raw telemetry into interactive dashboards. Grafana was installed on the sponsor's Linux server and accessed through a VPN connection using HTTP endpoints and administrative credentials described in the team's documentation. The dashboards created for this project include time-series graphs, bar charts, and tables that show error counts, service-level distributions, and host-level activity

Elasticsearch acts as the storage backend for logs and certain traces. Its indexing and query capabilities allow Grafana panels to retrieve and aggregate telemetry efficiently. In combination, these technologies form a complete observability stack capable of supporting performance monitoring, failure analysis, and system optimization.

.

Technical Summary

The project implemented an observability system that collects, processes, stores, and visualizes telemetry using the OpenTelemetry Collector, Elasticsearch, and Grafana. The design followed a pipeline-based architecture in which logs, metrics, and traces flow from a demo application to the visualization layer through standardized data paths.

The OpenTelemetry Collector formed the core of the system. It used OTLP receivers for gRPC and HTTP ingestion, processors for resource detection, attribute cleanup, and batching, and exporters for forwarding telemetry to Grafana Cloud and Elasticsearch. These components are defined in the project's collector configuration file, which outlines pipelines for traces, metrics, and logs.

Grafana served as the visualization platform and retrieved telemetry from Elasticsearch. The final dashboard presented error trends, service-level distribution, and host-level activity through time-series graphs, bar charts, and tables. The dashboard structure and panel queries are documented in the provided JSON configuration.

The telemetry originated from a demo application deployed on a sponsor-provided Linux server. Access to the server was established through the university VPN and SSH credentials, enabling the deployment of the application, the collector, and the visualization environment. The server documentation specifies connection procedures, URL endpoints, and authentication details used throughout the project. The overall architecture was designed to be modular, reproducible, and compatible with industry-standard observability practices. Each component contributed to a complete workflow that supports real-time monitoring and system analysis.

Outcomes and Achievements

The project successfully produced a fully functional observability system capable of collecting, processing, storing, and visualizing telemetry from a running application. The final solution met all core objectives:

- **End-to-end telemetry pipeline** using OpenTelemetry Collector
- **Real-time dashboards** presenting logs, metrics, and traces
- **Successful integration** of Grafana and Elasticsearch
- **Live error monitoring**, including host-level and service-level insights
- **System documentation** that allows future teams to reproduce the environment

Achievements by Objective

1. **Telemetry Pipeline Creation**

   The collector pipelines were successfully implemented for all three signal types. Resource enrichment, attribute filtering, batching, and Grafana Cloud exports functioned correctly, demonstrating a complete and accurate OpenTelemetry configuration.

2. **Application Deployment and Data Generation**

   The demo application produced meaningful telemetry that enabled realistic visualizations in Grafana. This created a practical simulation of a distributed system under active observation.

3. **Dashboard Development**

   Each dashboard panel displayed data sourced from Elasticsearch and reflected real-time system behavior. Panels such as “Errors by Host” and “Top Error Messages” allowed the system to detect patterns that would support debugging and system performance analysis.

4. **Documentation and Reproducibility**

   The server guide, collector configuration, and dashboard files provide sufficient detail for replication. Future teams can deploy additional applications or integrate other exporters without redesigning the system.

Project Planning and Management Summary\

Overview of Project Management

The project followed a structured planning and coordination process that emphasized scheduling, distributed work assignments, and continuous progress monitoring. Collaboration occurred through weekly team meetings, sponsor meetings, and class check-ins. Progress was tracked using shared documentation, GitHub repositories, and a team workspace on the university-provided Linux server. Each project phase was supported by written updates and logs that documented accomplishments, challenges, and next steps.

The team employed several project management practices, including milestone-based planning, iterative prototype development, and task escalation when clarification was needed. Changes to the project plan were made in response to sponsor feedback, technical barriers, and grading requirements. Communication with the sponsor and advisor ensured alignment with expectations and provided guidance when the team encountered obstacles. These practices supported project continuity even during periods of technical difficulty or scheduling changes.

Project Process by Stages and Milestones

Early Research and Orientation (September 8–21)

The initial stages focused on understanding OpenTelemetry concepts, sponsor expectations, and project requirements. The sponsor assigned research topics that helped the team clarify how telemetry is collected, processed, and visualized in distributed systems. Team meetings during this period established communication routines and aligned individual responsibilities. The project plan and MS Project timeline were developed and revised based on early feedback.

Clarification and Realignment (September 22–October 5)

The team encountered confusion regarding deliverables and project direction, leading to escalated discussions with the advisor and sponsor. Meetings during this period focused on correcting misunderstandings and revising the project plan. Updated deliverables were confirmed, and the team resumed structured progress toward Milestone 2. Collaboration improved through clarified roles and expectations.

Milestone 2 Preparation and Prototype Foundations (October 6–19)

The team prepared for Milestone 2 by reviewing feedback, exploring Linux services, and testing basic telemetry flows. Work included accessing the server environment, configuring VPN connections, and examining how frontend dashboards would integrate with the backend collector. The team also presented Milestone 2 and began designing the initial dashboard framework.

Prototype Development and Dashboard Design (October 19–November 9)

The team deployed the demo application, configured Docker environments, and validated telemetry ingestion through the OpenTelemetry Collector. Dashboards were expanded to include error metrics, request traces, and service relationships. Figma was used to guide dashboard layout. A GitHub repository was created to organize documentation and source files. The team rehearsed the demonstration and implemented alerting concepts such as burn-rate thresholds.

Transition to Elasticsearch and Final Dashboard Completion (November 10–23)

The dashboard was updated to use Elasticsearch instead of Jaeger. The team resolved issues related to container configuration, user access, and shared folders on the Linux server. Final adjustments were made to dashboard panels, and the observability system was brought to its near-final form. The prototype was completed and submitted for review.

5c. Team Contribution Summary

Kian Rezaeinejad

Contributed to telemetry pipeline setup, Docker configuration, and testing of the demo application. Designed dashboard elements, coordinated communication with the sponsor and advisor, updated the research draft, and managed milestone scheduling.

Dakobie Garcia

Provided documentation, researched observability tools, organized meeting materials, supported dashboard design, maintained GitHub updates, and communicated with the sponsor regarding dashboard access and deliverables.

Daniel Ekema

Developed Grafana dashboards, configured Elasticsearch integration, created prototype demonstrations, authored YML configurations for connectivity, deployed dashboard components to the server, and facilitated testing.

Jaytin Bolling

Maintained weekly logs, organized documentation, prepared presentation materials, completed written report and got checked by writing center, contributed to system overview clarity, supported server navigation and workflow documentation, and ensured team deliverables remained consistent and well-structured.

Workload Summary

Workload hours were reported weekly by each member, providing a transparent record of effort distribution. Across milestones, individual contributions ranged between 5 and 15 hours per week depending on technical tasks, meetings, and deliverable preparation. The combined hours show consistent engagement from all team members.

Overall workload patterns:

• **Milestone 1:** Research-focused contributions, averaging 5–8 hours per person.

• **Milestone 2:** Increased hours due to prototype development, environment setup, and dashboard design (10–15 hours for several members).

• **Milestone 3:** Steady contributions averaging 9–12 hours, focused on refinement, testing, and documentation.

Team Reflection on Project Experience

Project Success Factors

The project benefited from consistent communication with the sponsor and instructor, structured weekly meetings, and detailed documentation that supported continuity. Adaptability played a major role, especially when the team faced configuration issues or unclear requirements. These factors enabled steady progress toward a functional observability system.

Team Collaboration and Communication

1. **General Collaboration**

Team members worked independently on assigned tasks but coordinated closely on shared responsibilities such as dashboard design, server configuration, and prototype testing. Documentation and shared logs supported alignment.

2. Meeting Arrangements

Frequent team, sponsor, and class meetings ensured that expectations, technical requirements, and project goals remained clear. These meetings helped correct early misunderstandings and maintain momentum.

3. Collaboration Tools

Collaboration relied on GitHub for documentation, the Linux server for deployment, VPN and SSH tools for access, and Figma for dashboard planning. Messaging platforms supported day-to-day coordination.

4. Other Experiences

Team cohesion improved as roles became clearer. Sponsor feedback helped redirect efforts after early setbacks, strengthening the project's structure.

Challenges Encountered

Technical challenges included configuring Elasticsearch, managing Docker containers, accessing the Linux server, and updating dashboards after replacing Jaeger with Elasticsearch. Non-technical challenges included early confusion about deliverables, scheduling conflicts, and managing workload balance during peak periods. These issues were resolved through testing, documentation review, and sponsor guidance.

Areas for Improvement

Earlier clarification of project expectations would have reduced rework. More structured documentation, standardized workflows, and earlier technical implementation would have improved efficiency. Pipeline validation and automated checks could have reduced configuration errors.

What Would Be Done Differently

An earlier start on the technical pipeline would allow more time for testing and refinement. Defining team roles at the beginning could reduce task overlap. Earlier sponsor engagement on application selection and data flow design would help avoid mid-project transitions.

Recommendations for Future Teams

Future teams should expand the system by adding more applications, enhancing alerting, and introducing CI/CD workflows. Additional documentation, especially troubleshooting guidance, will support long-term sustainability. Consistent communication, early testing, and structured planning will help maintain progress and reduce implementation issues.

Bibliography

- Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site reliability engineering: How Google runs production systems*. O'Reilly Media. https://research.google/pubs/site-reliability-engineering-how-google-runs-production-systems/
- Burns, B., & Oppenheimer, D. (2021). Designing distributed systems for observability. *Communications of the ACM, 64*(3), 42–49. https://books.google.com/books/about/Designing_Distributed_Systems.html?id=SbM1EQAAQBAJ
- Cloud Native Computing Foundation. (2023a). *OpenTelemetry documentation*. https://opentelemetry.io
- Cloud Native Computing Foundation. (2023b). *OpenTelemetry Collector: Receivers, processors, exporters*. https://opentelemetry.io/docs/collector/
- Cloud Native Computing Foundation. (2024a). *OpenTelemetry semantic conventions*. https://opentelemetry.io/docs/specs/semconv/
- Cloud Native Computing Foundation. (2024b). *Sampling strategies (head vs. tail) in OpenTelemetry*. https://opentelemetry.io/docs/
- Elastic N.V. (2025, October 20). Elastic announces managed OTLP endpoint for easier, scalable OpenTelemetry [Press release]. *Business Wire*. https://www.businesswire.com/news/home/20251020492942/en/Elastic-Announces-Managed-OTLP-Endpoint-for-Easier-Scalable-OpenTelemetry

- Enterprise Management Associates. (2025, April 2). EMA webinar to explore OpenTelemetry's growing role in observability and IT performance [Press release]. *PR Newswire*. https://www.prnewswire.com/news-releases/ema-webinar-to-explore-opentelemetrys-growing-role-in-observability-and-it-performance-302417859.html
- Grafana Labs. (2024a). *Grafana: Metrics, logs, and traces (Tempo, Loki)*. https://grafana.com/docs/
- Grafana Labs. (2024b). *Loki: Log aggregation and label practices*. https://grafana.com/docs/loki/
- Jaeger Project. (2024). *Jaeger: End-to-end distributed tracing*. https://www.jaegertracing.io/docs/
- Majors, C., Miranda, G., & Barr, G. (2022). *Observability engineering*. O'Reilly Media. https://www.oreilly.com/library/view/observability-engineering/9781492076438/
- Prometheus Authors. (2024). *Prometheus documentation*. https://prometheus.io/docs/
- W3C. (2020). *Trace Context*. https://www.w3.org/TR/trace-context/
- Zipkin. (2024). *Zipkin distributed tracing*. https://zipkin.io/

Appendices

Appendix A. User Manual
Instructions for accessing the VPN, SSH login, the demo application, and Grafana dashboards.

Appendix B. Help Manual
Troubleshooting steps for collector errors, dashboard issues, and server access problems.

Appendix C. List of Materials
List of tools, technologies, and resources used, including OpenTelemetry, Docker, Elasticsearch, Grafana, and server files.

Appendix D. AI Prompts and Outputs
Prompts and AI-generated materials used during research and documentation.

Appendix E. Storage and Data Design
Overview of how Elasticsearch stored logs and telemetry data, including index usage.

Appendix F. Data Samples
Representative logs, metrics, and traces generated by the demo application.

Appendix G. Test Scripts
Scripts and commands used to test telemetry pipelines, server access, and dashboard functionality.

Appendix H. Step-by-Step Guides
Setup procedures for VPN, SSH access, running the demo app, and updating collector configurations.

Appendix I. Project Files List
Descriptions of key project files such as the collector configuration, dashboard JSON, and server documentation.

Appendix J. Progress Reports
Weekly logs and milestone summaries from the full project timeline.

Appendix K. Source Code
Configuration files, scripts, and other implementation artifacts.

Appendix L. Additional Materials
Screenshots, diagrams, and supporting visuals from the final system.