PROJECT REPORT - Complete Observability System (Metrics, Logs & Traces)

NAME: KANNAIAH LOKESH

GitHub Repository: ***https://github.com/Lokesh-Soft-Dev/complete-observability-system***

1. INTRODUCTION :

This project focuses on designing and deploying a **complete observability system** using open-source tools. Observability enables DevOps teams to monitor, debug, trace, and understand application behavior in real time.

The stack includes:

- ✓ **Prometheus** for metrics
- ✓ **Loki + Promtail** for centralized logs
- ✓ **Jaeger** for distributed tracing
- ✓ **Grafana** for visualization

All components run locally using **Docker Compose**, requiring no cloud resources.

2. OBJECTIVE :

The project aims to:

- ✓ Implement a fully local observability setup
- ✓ Monitor application performance through metrics
- ✓ Collect and centralize logs
- ✓ Enable end-to-end request tracing
- ✓ Visualize metrics, logs, and traces in Grafana
- ✓ Build dashboards for operational insights

3. TOOLS & TECHNOLOGIES :

- ✓ **Prometheus** – Metrics scraping & storage
- ✓ **Grafana** – Visualization & dashboards
- ✓ **Loki** – Log storage
- ✓ **Promtail** – Log collector
- ✓ **Jaeger** – Distributed tracing
- ✓ **Docker Compose** – Multi-container orchestration
- ✓ **Python Flask** – Sample instrumented application

4. SYSTEM ARCHITECTURE :

The sample Flask application generates:

- HTTP responses
- Structured logs
- Prometheus metrics **(/metrics)**
- Distributed traces using OpenTelemetry

Prometheus scrapes metrics periodically.
Promtail collects Docker logs and ships them to Loki.
Jaeger receives and visualizes trace data.
Grafana acts as the unified observability interface.

Note: A *screenshots* folder will be included in the GitHub repository showing Grafana, Prometheus, Jaeger, and Loki outputs.

5. IMPLEMENTATION DETAILS

Step 1 – Application Setup

- Flask app implemented with normal and error endpoints
- Metrics instrumented using Prometheus client
- Tracing enabled using OpenTelemetry

Step 2 – Containerization

- Dockerfile created
- Dependencies installed
- App exposed on port 5000

Step 3 – Monitoring Configuration

- Prometheus configured to scrape metrics every 15 seconds
- Loki + Promtail configured to collect and store logs

Step 4 – Tracing Setup

- Jaeger configured to receive OpenTelemetry spans
- All services orchestrated with Docker Compose

Step 5 – Visualization

Grafana configured with three data sources:

- Prometheus (metrics)
- Loki (logs)
- Jaeger (traces)

Dashboards created for metrics, logs, and trace insights.

6. RESULTS & ANALYSIS:

The observability stack worked successfully:

- Prometheus scraped application metrics and visualized trends
- Loki captured logs via Promtail and displayed them in Grafana
- Jaeger visualized request traces with timing details
- Grafana unified all observability signals in one place

Key insights: ·

- Error endpoints produced identifiable traces
- Request counts and latency metrics provided performance visibility
- Logs correlated with traces helped identify root causes

7. **DELIVERABLES**:

The repository includes:

- ✓ **docker-compose.yml**
- ✓ Application source code
- ✓ Prometheus config
- ✓ Loki & Promtail configs
- ✓ Dashboard JSON files
- ✓ Screenshots folder
- ✓ PDF Project Report

8. **CONCLUSION**:

This project demonstrates a complete observability solution integrating **metrics, logs, and traces** into one ecosystem. It reflects real-world DevOps monitoring systems used in production.

Suitable for:

- Debugging performance issues
- Monitoring uptime and latency
- Understanding request flow

Future Enhancements:

- Integrate Alertmanager
- Deploy stack on Kubernetes
- Add SLO/SLI dashboards
- Increase application complexity

This project forms a solid foundation for DevOps and observability engineering.

* * *