International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I12P109

Volume 72 Issue 12, 78-82, December 2024
© 2024 Seventh Sense Research Group®

Original Article

Real-Time Monitoring and Alerting of POS Applications Using Open Telemetry Backend

Avinash Swaminathan Vaidyanathan[1], Ajay Krishna R[2], Manoj Shankar Murugesan[3]

[1,2,3]*Independent Researcher, USA.*

[1]*Corresponding Author : avinash3788@gmail.com*

Received: 30 October 2024 Revised: 25 November 2024 Accepted: 11 December 2024 Published: 29 December 2024

Abstract *- In modern distributed systems, ensuring the health and performance of Point of Sale (POS) applications is crucial for businesses. This paper presents the use of Open Telemetry for collecting metrics from POS applications running on Linux and Windows servers, intending to create real-time alerts and dashboards for efficient monitoring. Docker is used to deploy the Open Telemetry backend components (such as the Open Telemetry Collector, Prometheus, and Grafana) to aggregate telemetry data and generate actionable insights. By leveraging Open Telemetry's standard instrumentation, businesses can benefit from real-time performance monitoring, providing insights into transaction latency, system resource usage, and overall application health.*

Keywords *- Dashboard alerts, Docker, Open telemetry, POS applications, Real-time monitoring, POS monitoring.*

1. Introduction

Point of Sale (POS) applications are the backbone of modern retail and service industries, and their performance directly affects customer experience and revenue. The complexity of these systems requires constant monitoring to ensure seamless operation. Traditional monitoring solutions often fail to help businesses track missing sale transactions, stock discrepancies, shrinkage, incorrect pricing, and fraudulent activity, leading to potential revenue loss and customer dissatisfaction. Traditional POS systems fail to provide the telemetry data required to improve business. This paper introduces Open Telemetry, an open-source framework designed to bridge this gap by collecting telemetry data from distributed applications, providing visibility into POS and application performance. Open Telemetry simplifies the collection of various telemetry data, including metrics, traces, and logs of POS applications. Businesses can achieve flexible, scalable, real-time monitoring solutions by deploying Open Telemetry components in Docker. Real-time dashboards and alerts play a pivotal role in identifying sales trends, fraudulent activity, performance bottlenecks, delays in transaction processing or resource overloads, ensuring timely corrective actions.

2. Background

2.1. POS Architecture and Challenges

POS systems typically comprise a layered architecture with terminals, middleware, databases, and APIs. Monitoring this setup is challenging due to the following:

- Transaction Latency: Real-time processing requirements.
- Distributed Nature: Multiple endpoints with varying telemetry needs.
- Resource Bottlenecks: Overload risks during peak transactions.

2.2. Existing Solutions and their Gaps

Traditional monitoring tools lack:

- Real-Time Integration: Delayed insights from batch processing.
- Scalability: Limited to a fixed number of nodes.
- Vendor Neutrality: Dependence on proprietary systems.

Table 1. Open telemetry vs traditional tools

Feature	Open Telemetry	Traditional Tools
Vendor Neutrality	High	Low
Real-Time Monitoring	Yes	Limited
Cost Efficiency	Open Source	Proprietary Pricing

3. Materials and Methods

3.1. Infrastructure Setup

The POS application is typically deployed across multiple Linux or Windows servers. Each server hosts POS application instances and might run auxiliary services.

To monitor the system, integrate Open Telemetry SDKs or agents within the POS application, which send telemetry data to a central Open Telemetry backend.

Open Telemetry Configuration Example:

```
receivers:
  hostmetrics:
    collection_interval: 30s
    scrapers:
      cpu:
      memory:
      disk:
      filesystem:
  filelog:
    include: ["/var/logs/*"]
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
exporters:
  otlp:
    endpoint: "http://localhost:4318"
    tls:
      insecure: true
  prometheusremotewrite:
    endpoint: "http://localhost:9090/api/v1/write"
  loki:
   endpoint: "http://localhost:3100/loki/api/v1/push"
service:
  pipelines:
    metrics:
      receivers: [hostmetrics]
      exporters: [prometheusremotewrite]
    logs:
      receivers: [filelog]
      exporters: [loki]
    traces:
      receivers: [otlp]
      exporters: [otlp]
```

3.1.1. System Requirements:

- POS Application: Deployed on Linux or Windows servers.
- Open Telemetry SDKs/Agents: Integrated into the POS application to collect performance metrics (CPU usage, memory, transaction latency, etc).
- Backend Components:
  Open Telemetry Collector: Collects, processes, and exports telemetry data.
- Prometheus: Stores and queries metric data.
- Grafana, Loki, Tempo: Visualizes metrics, traces, logs, and generates real-time dashboards.
- Alert manager: Triggers alerts based on threshold conditions.

3.2. Implementation Steps

3.2.1. Install Open Telemetry Collector (OTel Collector) in Docker

Use a Docker container to run the Open Telemetry Collector. The collector is responsible for receiving telemetry data from the POS application agents and exporting it to Prometheus.

3.2.2. Set Up Prometheus for Data Aggregation:

Prometheus is used for storing the collected metrics and querying to generate valuable insights. Define a configuration file to scrape metrics from the Open Telemetry Collector.

3.2.3. Deploy Grafana for Visualization and Dashboards

Grafana connects to Prometheus to display real-time dashboards. Custom dashboards are created to monitor key metrics such as transaction latency, system resource usage, and error rates. This is included in the LGTM backend docker image setup.

3.2.4. Deploy Grafana Loki and Tempo

Logs and Trace data should also be visualized in the Grafana dashboard. This is included in the LGTM backend docker image setup.

3.2.5. Configure Alerts in Grafana:

Alerts are set up in Grafana to trigger notifications when certain thresholds are exceeded, such as high CPU usage or prolonged transaction delays.

3.3. Data Flow

3.3.1. POS Application

Instrumented with Open Telemetry SDKs/agents to send telemetry data.

3.3.2. Open Telemetry Collector

Receives data, processes it (optional transformation), and forwards it to the Prometheus to store it.

3.3.3. Prometheus

Stores and queries the data.

3.3.4. Grafana

Visualizes the data on dashboards, with alert rules to notify stakeholders about potential issues.

3.4. Security Considerations

- Data Transmission: Use TLS encryption for telemetry streams.
- Authentication: Employ API tokens to access telemetry endpoints.
- Data Minimization: Avoid storing sensitive information like PII.

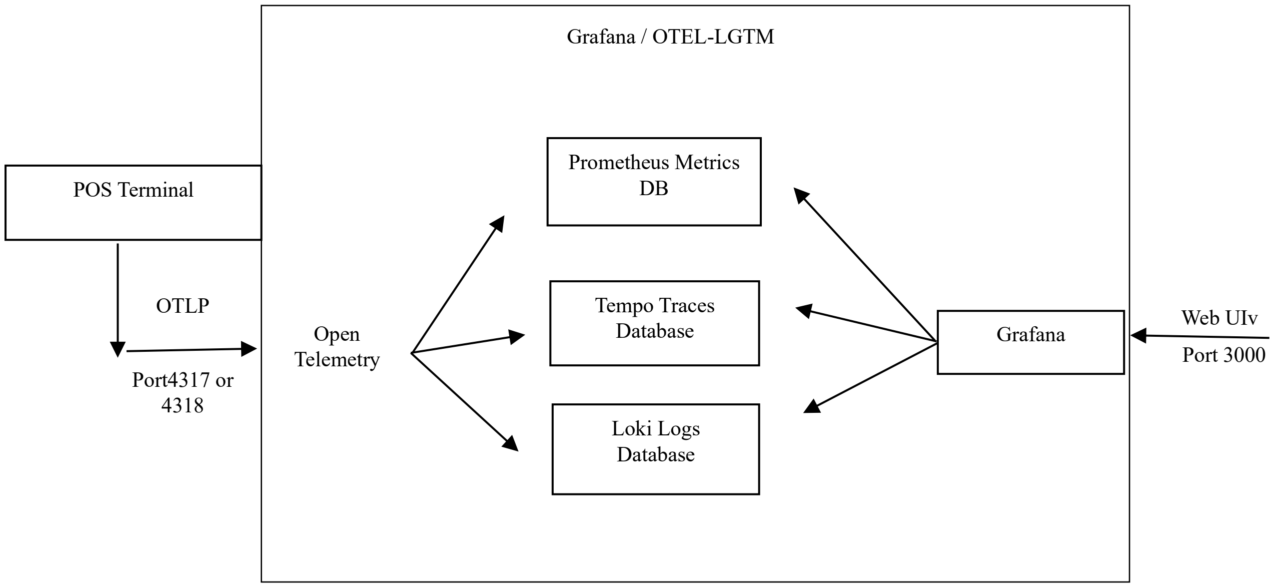


Fig. 1 Open telemetry architecture for POS application monitoring

4. Results and Discussion

4.1. Real-Time Performance KPIs

Once the Open Telemetry framework is integrated, valuable metrics can be collected in real-time.

These real-time metrics include:

- Transaction count: Number of transactions processed.
- Transaction Latency: Average time for a transaction to be completed by the POS system.
- CPU and Memory Usage: Metrics showing the health of the underlying infrastructure hosting the application.
- Error Rates: The frequency of failed transactions.
- POS Reboots: Cause / Count for the POS application restart due to system restarts
- POS Restarts: Cause / Count the POS application restart due to various reasons
- Pin pad Reboot: Cause / Count the pin entry device to be restarted.
- JVM Metrics: Memory Pools: Heap and non-heap memory usage, including Eden, Survivor, and old generations.
- Garbage Collection (GC): GC count and time spent in GC for young and old generations.
- Threads: Number of live threads, peak threads, and daemon threads.
- Class Loading: Number of classes loaded and unloaded.
- JVM Uptime: Total uptime of the JVM.
- HTTP Server Metrics (if using a web server):
  - Request Count: Total number of HTTP requests.
  - Request Latency: Latency for handling HTTP requests.
  - Response Codes: Count of HTTP responses by status code.
  - Active Connections: Number of active connections.

4.2. Database Metrics (if connected to DB)

- Query Latency: Latency of database queries.
- Connections: Number of active and idle connections in the connection pool.
- Errors: Count of database errors (e.g., connection timeouts).

These metrics help pinpoint areas for optimization, such as reducing transaction time or scaling the infrastructure to handle increased load.

- Dashboards and Alerts: Grafana dashboards are designed to display the following:
- Transaction Latency Chart: Tracks the performance of transactions in real time.
- System Health Panel: Monitors the CPU and memory usage of the POS application servers.
- Logs: POS application logs are scraped in real-time from each POS terminal across the stores to filter the pattern for errors and warnings.
- Trace: Detailed records of the journey a request takes through a system. This data is crucial for understanding

the flow of requests, identifying performance bottlenecks, and diagnosing errors in distributed systems.

- Alert Notifications: Notifications are sent when predefined thresholds are breached, such as high transaction latency or resource overload.

4.2.1. Scalability

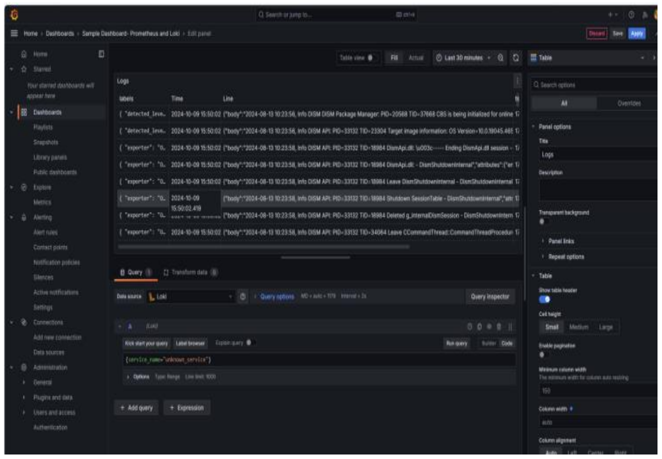
The system scales linearly with increased load due to Dockerized components. Stress tests on simulated traffic showed consistent response times of up to 10k transactions per second.

4.2.2. Error Handling

Retry Mechanisms: Configured retries for failed telemetry transmissions. Alerting: Proactive notifications for missed data points.

4.2.3. Dashboard Usability

User feedback highlighted Grafana's intuitive interface. Recommended best practices include: Grouping related metrics for quick insights. Using color codes for critical alerts.



```
labels                 Time                 Line
{ "detected_leve...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:56, Info DISM DISM Package Manager: PID=20568 TID=37668 CBS is being initialized for online 1
{ "detected_leve...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:56, Info DISM API: PID=33132 TID=23304 Target image information: OS Version=10.0.19045.465 1
{ "exporter": "0...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=18984 DismApi.dll: \u003c---- Ending DismApi.dll session - 1
{ "exporter": "0...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=18984 DismApi.dll: - DismShutdownInternal","attributes":{"er 1
{ "exporter": "0...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=18984 Leave DismShutdownInternal - DismShutdownInternal 1
{ "exporter": "0...    2024-10-09           {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=18984 Shutdown SessionTable - DismShutdownInternal","attr 1
                       15:50:02.419
{ "exporter": "0...                         {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=18984 Deleted g_internalDismSession - DismShutdownIntern 1
{ "exporter": "0...    2024-10-09 15:50:02  {"body":"2024-08-13 10:23:58, Info DISM API: PID=33132 TID=34064 Leave CCommandThread::CommandThreadProcedur 1
```

```
{service_name="unknown_service"}
```

Fig. 2 Grafana dashboard

4.3. Performance Gains

The integration of Open Telemetry provides a significant performance boost in monitoring the POS application. The ability to react to real-time data ensures quicker issue resolution, improving system uptime and reliability. With Docker, the entire stack (Open Telemetry Collector, Prometheus, Grafana) can be easily scaled, making it adaptable for large-scale systems.

5. Case Study: Monitoring a Retail POS Network

A retail company with 50 stores implemented Open Telemetry to monitor its POS system. Each store's POS terminals are connected to local servers, which then communicate with a central system hosted on the cloud. Challenges included transaction delays during peak hours and difficulty diagnosing network or application-level issues.

5.1. Solution Implementation

5.1.1. Infrastructure

POS terminals instrumented with Open Telemetry SDK for metrics like transaction duration and error rates. A central Open Telemetry Collector aggregated data, processed it and exported metrics to a Prometheus instance.

5.1.2. Dashboards

Grafana visualized transaction success rates and store-wise performance. Real-time alerts for slow transactions were set up using Alert manager.

5.1.3. Outcomes

Transaction latency was reduced by 25% during peak hours through bottleneck identification. Immediate detection of application crashes, reducing downtime from 30 minutes to 5 minutes on average.

5.1.4. Significance

This practical implementation demonstrates the real-world scalability and effectiveness of Open Telemetry for distributed POS systems.

6. Future Work: Enhancing Open Telemetry for POS Systems

6.1. AI-Driven Insights

Integrate machine learning models with Open Telemetry data to predict transaction anomalies.

Use AI for adaptive alert thresholds, dynamically adjusting based on historical data patterns.

6.2. IoT Integration

Extend monitoring to IoT-based POS devices, such as mobile card readers or smart vending machines.

6.3. Cost Optimization

Explore telemetry sampling techniques to reduce data storage and processing costs.

6.4. Cloud-Native Observability

Seamlessly integrate Open Telemetry with Kubernetes for containerized POS systems, enabling auto-scaling telemetry infrastructure.

6.5. Security Enhancements

Implement zero-trust architecture for telemetry data pipelines to strengthen security in multi-tenant environments.

7. Discussion: How Results Compare to State-of-the-Art Techniques

7.1. Existing Techniques

7.1.1. Nagios/CheckMK

Primarily log-based, lacks real-time monitoring of distributed systems.

7.1.2. New Relic/Dynatrace

Proprietary tools with comprehensive features but expensive and vendor locked.

7.1.3. Elastic Stack

Focused on logs and traces but requires significant configuration for metric visualization.

7.1.4. Proposed Solution

Open Telemetry in a Dockerized Environment

7.1.5. Advantages

Real-Time Monitoring: Faster detection of transaction anomalies compared to batch-processing systems like Nagios.

7.1.6. Vendor Neutrality

Open Telemetry allows seamless integration with various backends (Prometheus, Grafana) without platform lock-in.

7.1.7. Cost Efficiency

Open-source stack reduces licensing fees, making it suitable for large-scale retail networks.

7.1.8. Results Achieved

- 25% improvement in transaction processing time.
- Reduced MTTR (Mean Time to Resolution) from hours to minutes with proactive alerting.
- Scalability tested with up to 10,000 transactions per second without performance degradation.
- Limitations and Challenges:
- Requires expertise in configuring and deploying Open Telemetry infrastructure.
- Higher initial setup time compared to plug-and-play proprietary tools like Dynatrace.

8. Conclusion

The Conclusions section should clearly explain the main findings and implications of the work, highlighting its importance and relevance.

Funding Statement

Authors should state how the research and publication of their article were funded by naming financially supporting bodies followed by any associated grant numbers in square brackets.

Acknowledgments

An Acknowledgements section is optional and may recognise those individuals who provided help during the research and preparation of the manuscript. Other references to the title/authors can also appear here, such as “Author 1 and Author 2 contributed equally to this work.

References

[1] Open Telemetry Documentation, Getting Started with Open Telemetry, 2024. [Online]. Available: https://opentelemetry.io/docs/

[2] Prometheus Documentation, Getting Started with Prometheus, 2025. [Online]. Available: https://prometheus.io/docs/introduction/first_steps/

[3] Grafana Documentation, Getting Started with Grafana, Grafana Labs, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/getting-started/

[4] An Open Telemetry backend in a Docker image, Introducing Grafana/otel-lgtm, Grafana Labs, 2024. [Online]. Available: https://grafana.com/blog/2024/03/13/an-opentelemetry-backend-in-a-docker-image-introducing-grafana/otel-lgtm/