happiest minds
The Mindful IT Company
Born **Digital** . Born **Agile**

Decoding the Digital Nervous System: Observability Unveiled

A digital nervous system is a framework that is aware of every intricate detail of a software system. It is like a sixth sense for the system - alerting about the problems before they happen, tracking every user interaction across dozens of services, and helping us understand the root causes of problems.

In this article, we will dive deep into how observability works, detailing its pillars and architecture. It is also crucial to understand the importance of observability.

Implementing observability in software systems directly provides the following business benefits:

- **Improved Operational Efficiency :** Quickly identify issues, bottlenecks, and inefficiencies, leading to faster resolution and smoother operations.
- **Reduced Downtime :** Proactively monitor system health to catch problems early, minimizing disruptions and maintaining business continuity.
- **Enhanced Customer Satisfaction :** Ensure high system reliability and performance, resulting in better user experiences and increased customer trust.
- **Data-Driven Decision-Making :** Leverage insights from logs, metrics, and traces to make informed business and technical decisions that drive growth.
- **Custom Dashboards for Insights :** Create tailored dashboards that focus on key performance indicators (KPIs), giving stakeholders real-time visibility into system health and business impact.
- **Faster Issue Resolution :** With detailed data on system performance, teams can quickly pinpoint and address the root cause of problems, reducing resolution time.
- **Resource Optimization :** nalyze system performance trends to optimize resource usage, reducing waste and improving cost efficiency.
- **Continuous Improvement :** Use insights to identify patterns, refine processes, and drive ongoing improvements across the system and organization.
- **Increased Competitive Advantage :** Reliable, high-performing systems can improve customer retention and differentiate the business in a competitive market.

Despite its critical importance, observability remains an underappreciated tool in software engineering. Several key challenges contribute to this undervaluation:

1. **Shared Responsibility Ambiguity:** Observability is a collaborative effort between development and DevOps teams, yet there is often a lack of clear delineation of roles and responsibilities. This ambiguity can lead to gaps in system monitoring and analysis.

2. **Perceived Value Limitation:** Many development teams struggle to recognize the full value of observability, particularly when production environments are not directly managed by the development team.

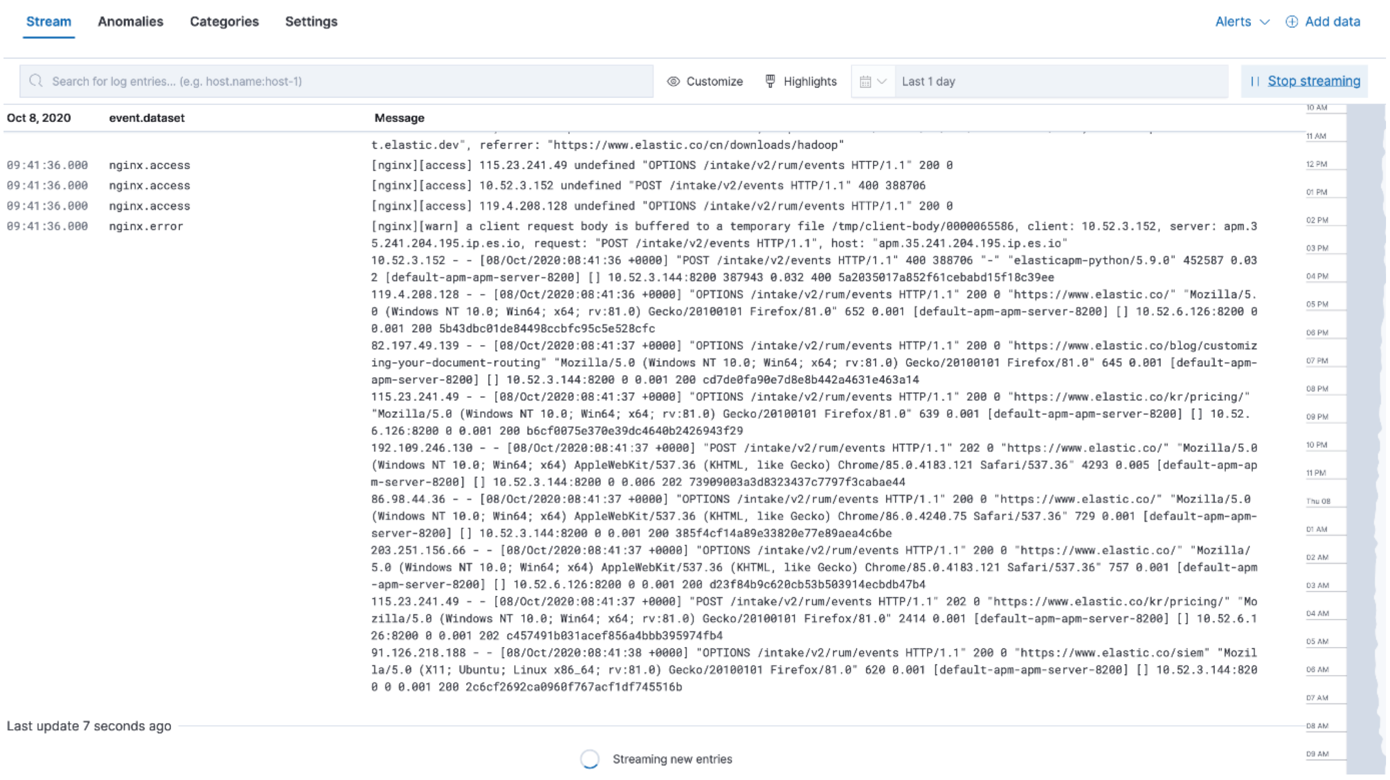
3. **Inherent Complexity :** Observability encompasses every component of a software system, making it a multifaceted and intricate discipline.

Let's systematically unravel the complexities & challenges of observability by focusing on how observability works under the hood.

Pillars of Observability

Observability is built on three fundamental pillars:

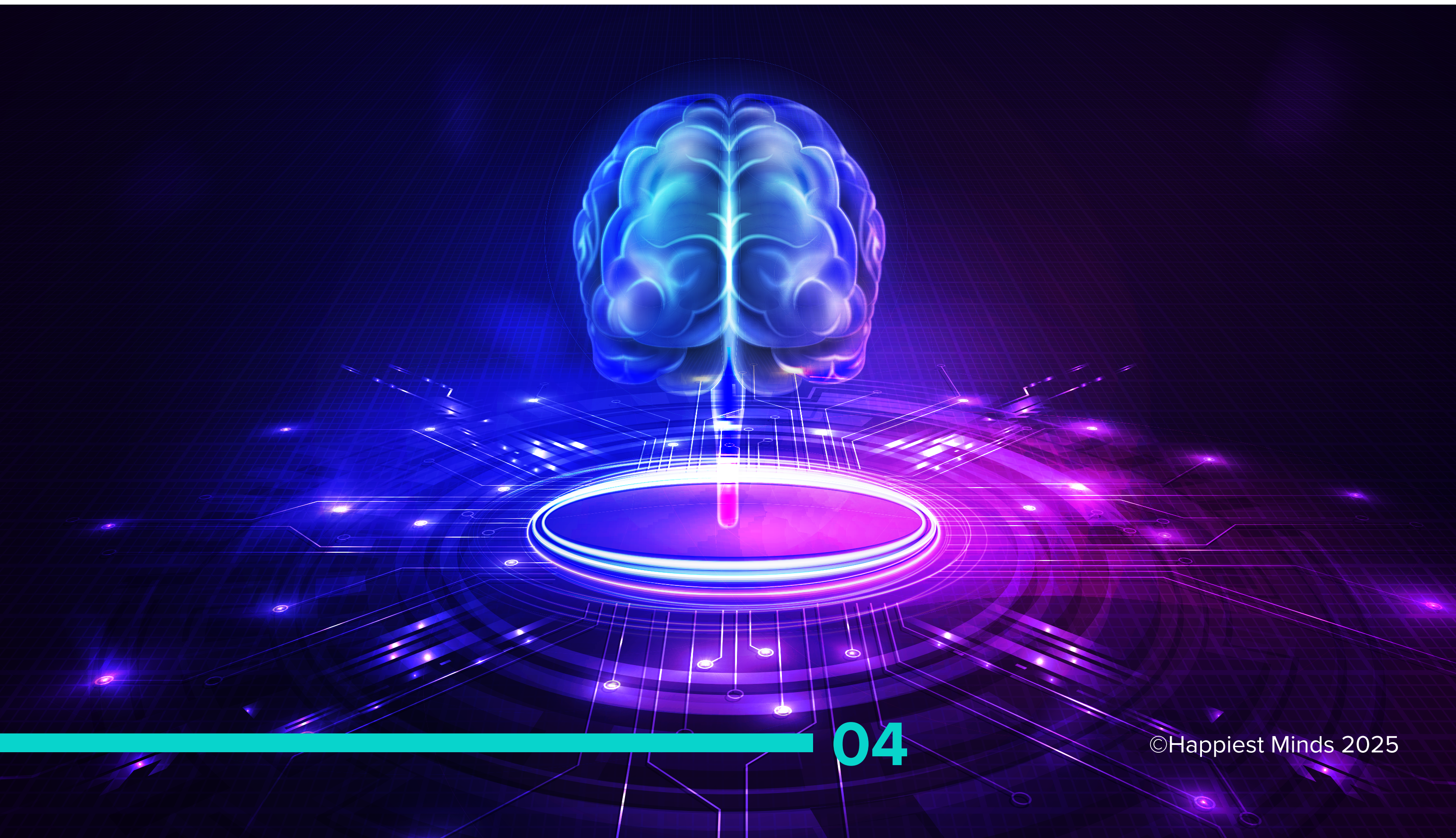
1. Logs: These are detailed records of events that happen in the system, like error messages, operational sequences or status updates. An observability system can display live stream of logs & historical logs based on retention policy.



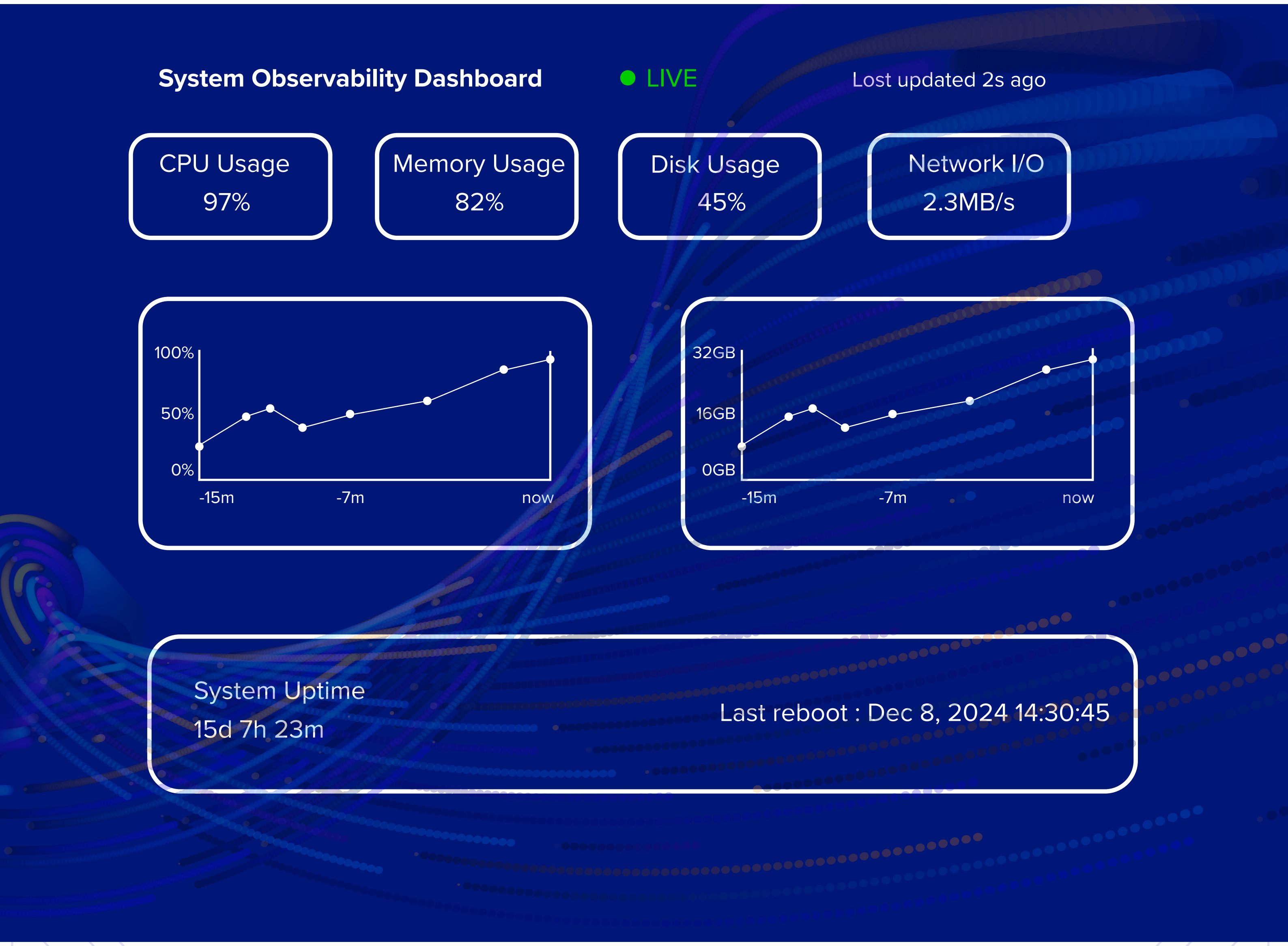
```
Oct 8, 2020    event.dataset    Message
                                t.elastic.dev", referrer: "https://www.elastic.co/cn/downloads/hadoop"
09:41:36.000   nginx.access     [nginx][access] 115.23.241.49 undefined "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0
09:41:36.000   nginx.access     [nginx][access] 10.52.3.152 undefined "POST /intake/v2/events HTTP/1.1" 400 388706
09:41:36.000   nginx.access     [nginx][access] 119.4.208.128 undefined "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0
09:41:36.000   nginx.error      [nginx][warn] a client request body is buffered to a temporary file /tmp/client-body/0000065586, client: 10.52.3.152, server: apm.3
                                5.241.204.195.ip.es.io, request: "POST /intake/v2/events HTTP/1.1", host: "apm.35.241.204.195.ip.es.io"
                                10.52.3.152 - - [08/Oct/2020:08:41:36 +0000] "POST /intake/v2/events HTTP/1.1" 400 388706 "-" "elasticapm-python/5.9.0" 452587 0.03
                                2 [default-apm-apm-server-8200] [] 10.52.3.144:8200 387943 0.032 400 5a2035017a852f61cebabd15f18c39ee
                                119.4.208.128 - - [08/Oct/2020:08:41:36 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/" "Mozilla/5.
                                0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0" 652 0.001 [default-apm-apm-server-8200] [] 10.52.6.126:8200 0
                                0.001 200 5b43dbc01de84498ccbfc95c5e528cfc
                                82.197.49.139 - - [08/Oct/2020:08:41:37 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/blog/customiz
                                ing-your-document-routing" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0" 645 0.001 [default-apm-
                                apm-server-8200] [] 10.52.3.144:8200 0 0.001 200 cd7de0fa90e7d8e8b442a4631e463a14
                                115.23.241.49 - - [08/Oct/2020:08:41:37 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/kr/pricing/"
                                "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0" 639 0.001 [default-apm-apm-server-8200] [] 10.52.
                                6.126:8200 0 0.001 200 b6cf0075e370e39dc4640b2426943f29
                                192.109.246.130 - - [08/Oct/2020:08:41:37 +0000] "POST /intake/v2/rum/events HTTP/1.1" 202 0 "https://www.elastic.co/" "Mozilla/5.0
                                (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36" 4293 0.005 [default-apm-ap
                                m-server-8200] [] 10.52.3.144:8200 0 0.006 202 73909003a3d8323437c7797f3cabae44
                                86.98.44.36 - - [08/Oct/2020:08:41:37 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/" "Mozilla/5.0
                                (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36" 729 0.001 [default-apm-apm-
                                server-8200] [] 10.52.3.144:8200 0 0.001 200 385f4cf14a89e33820e77e89aea4c6be
                                203.251.156.66 - - [08/Oct/2020:08:41:37 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/" "Mozilla/
                                5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36" 757 0.001 [default-apm
                                -apm-server-8200] [] 10.52.6.126:8200 0 0.001 200 d23f84b9c620cb53b503914ecbdb47b4
                                115.23.241.49 - - [08/Oct/2020:08:41:37 +0000] "POST /intake/v2/rum/events HTTP/1.1" 202 0 "https://www.elastic.co/kr/pricing/" "Mo
                                zilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0" 2414 0.001 [default-apm-apm-server-8200] [] 10.52.6.1
                                26:8200 0 0.001 202 c457491b031acef856a4bbb395974fb4
                                91.126.218.188 - - [08/Oct/2020:08:41:38 +0000] "OPTIONS /intake/v2/rum/events HTTP/1.1" 200 0 "https://www.elastic.co/siem" "Mozil
                                la/5.0 (X11; Ubuntu; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0" 620 0.001 [default-apm-apm-server-8200] [] 10.52.3.144:820
                                0 0 0.001 200 2c6cf2692ca0960f767acf1df745516b

Last update 7 seconds ago

Streaming new entries
```



2. Metrics: Quantitative measurements that track system performance, resource utilization, and key operational indicators, enabling precise performance analysis. Examples: Server CPU & memory utilization, system throughput, resource-consuming methods or classes etc.



System Observability Dashboard ● LIVE Lost updated 2s ago

CPU Usage 97%

Memory Usage 82%

Disk Usage 45%

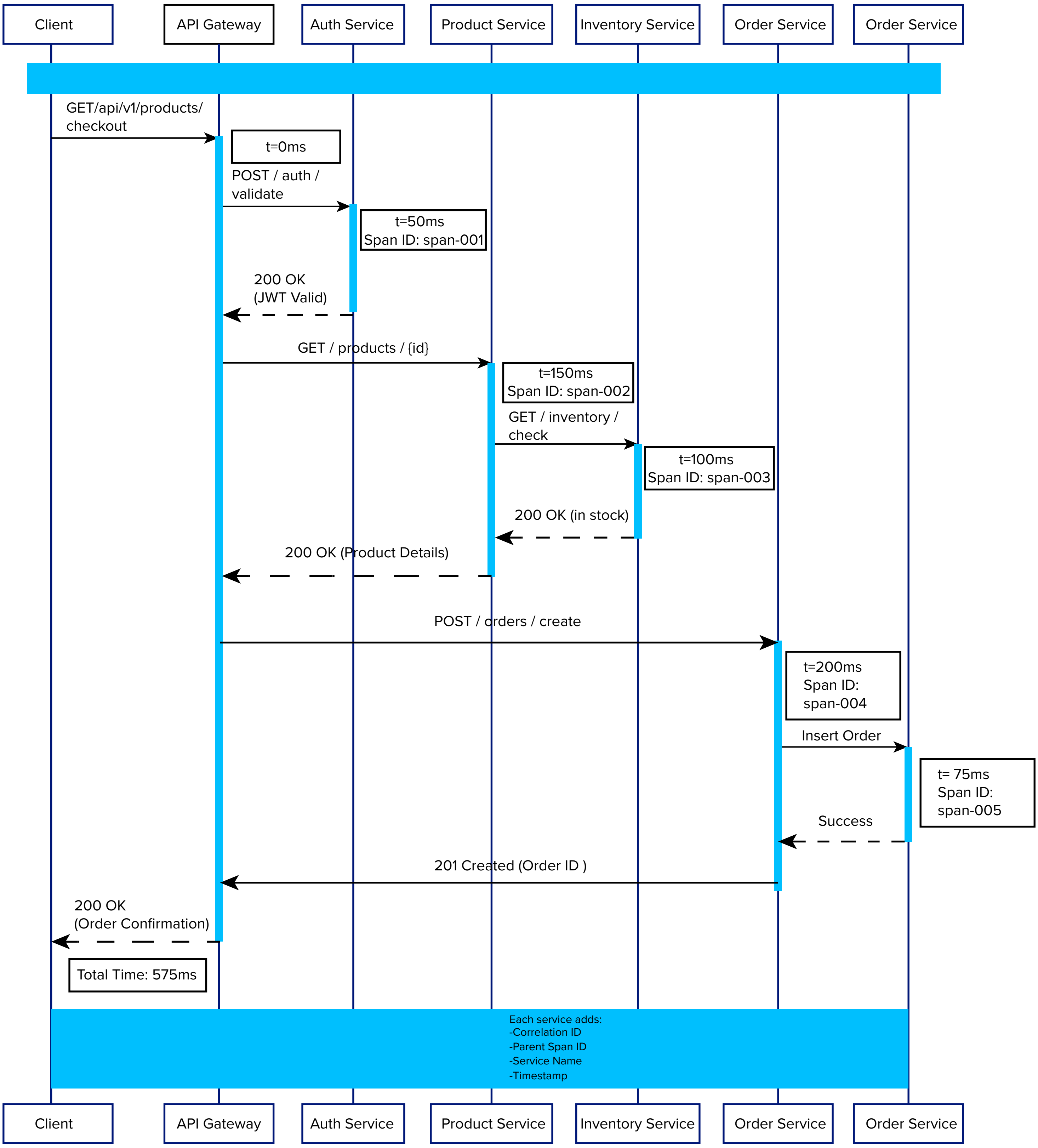
Network I/O 2.3MB/s

System Uptime 15d 7h 23m

Last reboot : Dec 8, 2024 14:30:45

3 **Traces**: In a distributed microservices architecture, a user journey often spans multiple services. Traceability provides the sequence of services involved, along with key details such as latency for each action, errors encountered, and the full transaction flow helping to pinpoint performance bottlenecks and system failures.

Traceability tracks requests across distributed services using correlation IDs. Each service interaction creates a span, forming a trace that shows the complete request journey and timing.



Observability Architecture

While logs, metrics, and traces might seem distinct in the observability landscape, they share a remarkable architectural DNA that transforms them from isolated data points into a comprehensive system narrative.

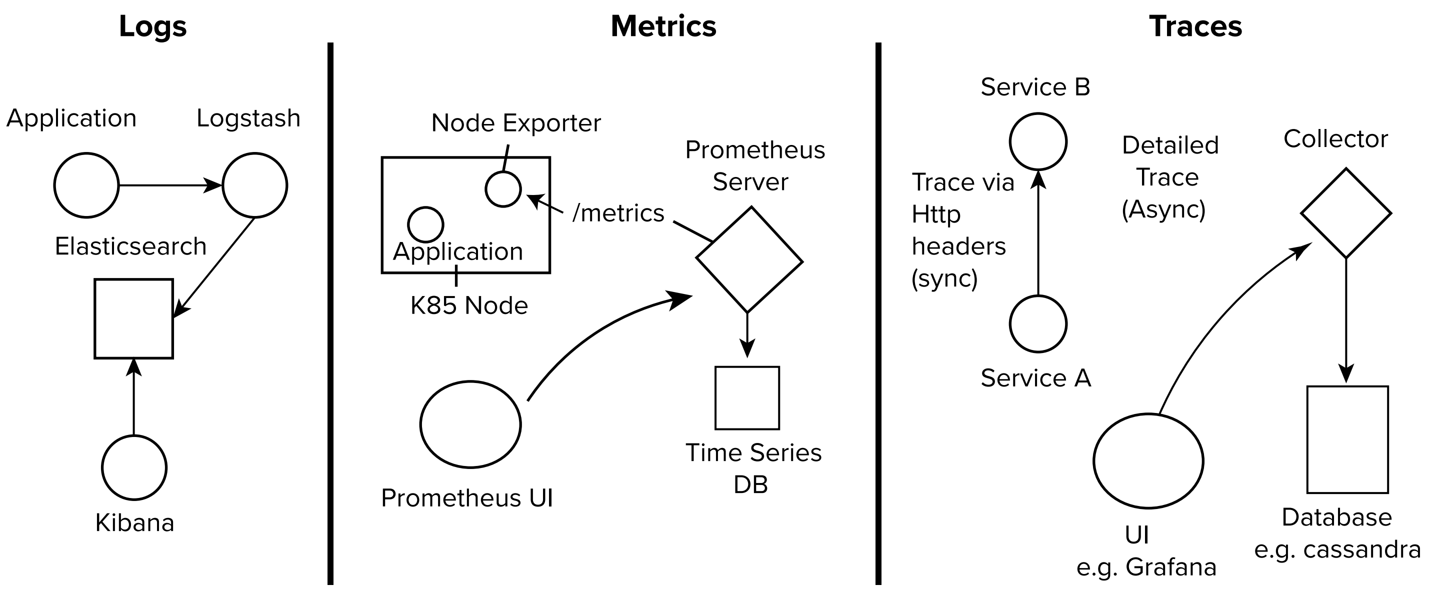
It is essential to understand 3 fundamental concepts that form the foundation of observability: **Instrumentation**, **Scraping**, and **Visualization.**

Instrumentation: This refers to adding code or tools to the application to collect data on its behavior, such as performance metrics, logs, and traces. It's like setting up sensors in the system that monitor and report its activity.

Scraping: : Scraping is the process of collecting data from the various services or applications that have been instrumented. Tools like Prometheus "scrape" data at regular intervals, pulling metrics from the services so that they can be analyzed.

Visualization: This is the process of displaying the collected data in a user-friendly way. Visualization tools, like Grafana, allow engineers to create dashboards that turn raw data into meaningful charts, graphs, and alerts, providing clear insights into system performance and health.

Having outlined the pillars and key concepts of observability, let's examine its architecture.



Logs

- Applications are responsible to instrument logs by adding logging code into source code to record runtime information. This is done with the help of libraries that application code can extend e.g. Opentelemetry (multiple languages), Winston (Node.js), Logback (Java), Structlog (Python), ZAP (Go) etc.
- The logs are then scraped by collection & processing tools e.g. Fluentbit, Logstash, Promtail.
- The data collection tools then send these processed logs to storage e.g. Elasticsearch, OpenSearch, Loki.
- The stored log data can then be queried and visualized in the form of meaningful dashboards by a tool e.g. Grafana, Datadog, Elasticsearch, OpenSearch.

Metrics

- Prometheus is the most widely used tool for metric & alert management. We use Prometheus to understand how metrics fit into observability.
- Metrics are utilization & performance data of the server/node on which the applications are running. These metrics are generally **instrumented** through an agent or sidecar service running on the same server/node. Occasionally the application can also generate its metrics e.g. threads, response time, heap memory usage by adding libraries from Prometheus.

  The libraries and exporters expose an API endpoint (/metrics) to provide the metric data.
- The metric data is **scraped** by a Prometheus server by calling the API endpoint.
- Alerting rules can be configured in the Prometheus server based on threshold values. The Prometheus server uses alert managers to send **alerts** via emails and messages.
- The scraped metric data is sent to a time series database for **storage** by the Prometheus server. The Prometheus server exposes APIs for metrics to be queried by a visualization tool.
- Visualization tools like Prometheus UI, Grafana can call the APIs or directly query the database using query language PromQL to **visualize** the metric data meaningfully.

Traces

- Traces are applicable in a distributed architecture with microservices. Each microservice is responsible for **instrumenting** trace data (span ID, correlation ID, etc.) by adding dependencies in the application e.g. Opentelemetry, Zipkin. Other tools like Jeager require an agent to be deployed as a sidecar alongside the application to **instrument** tracing data.
- Traces are simultaneously **scraped** to two destinations immediate microservice in the flow & centralized trace collector. Minimal trace information i.e. Span is sent to immediate microservice via HTTP header synchronously along with the request details. Detailed trace data is sent to trace collector e.g. Jeager collector or Zipkin collector asynchronously. This is done to minimize performance impact on the source microservices. The trace data is saved in a database like Cassandra / Elasticsearch etc.
- UI tools like Grafana, Jeager UI **visualize** the traces by querying the trace data from collector.

Responsibility Boundaries

Observability is a shared responsibility between Developers & DevOps engineers. Shared responsibility with unclear boundaries can lead to conflicts and under-performing systems. Let's define the boundary of responsibility:

Developers:
Code-level instrumentation
trace/log quality

Devops:
Infrastructure setup
data collection
visualization platforms

Development teams typically own the instrumentation, embedding observability hooks within the application code, defining meaningful traces, and ensuring log richness. They're responsible for creating high-quality, context-aware telemetry that captures application-specific behaviors.

DevOps engineers, conversely, focus on the observability infrastructure & visualization. Their domain includes setting up collection tools like Prometheus, configuring log stores such as Elasticsearch, designing dashboards, and managing centralized tracing systems. They ensure the observability pipeline is robust, scalable, and provides comprehensive system insights.

The value of observability is realised at the intersection of the two domains.

Bringing It All Together

Observability requires the right tools and architecture to function effectively.

Essential Building Blocks

Core Components

- Collection Tools: Your system's sensors (Prometheus, Fluentd, Jaeger, OpenTelemetry Collector)
- Storage Solutions: Your system's memory (Elasticsearch, Loki, Tempo, Opensearch)
- Visualization: Your system's dashboard (Grafana, Kibana, Prometheus UI)

Popular Stacks

For Startups & Small Teams

- PLG Stack (Prometheus, Loki, Grafana)

- Perfect for Kubernetes environments
- Cost-effective monitoring solution
- Quick to set up and maintain
- Getting started with Grafana & Prometheus

For Growing Organizations

○ OTLP Stack (OpenTelemetry, Tempo, Loki, Prometheus)

- Vendor-neutral solution
- Future-proof architecture
- Strong signal correlation
- Great for scaling teams
- Getting started with Opentelemetry

For Medium to Large Organizations

○ ELK Stack (Elasticsearch, Logstash, Kibana)

- Powerful search capabilities
- Extensive community support
- Rich ecosystem of plugins
- Learn more about ELK Stack

For Enterprise & Complex Systems

○ Enterprise Solutions (Datadog, Dynatrace)

- Full observability out of the box
- Minimal setup required
- AI-powered insights
- Datadog documentation

Quick Start Guide

01 Start with basic metrics collection using Prometheus

02 Add structured logging with ELK, PLG, or OTLP stack

03 Implement distributed tracing as your system grows

04 Build meaningful dashboards to visualize your data

Next Steps

01 Explore our recommended tools' documentation

02 Join observability communities on Slack or Discord

03 Start small, measure what matters, and scale as needed

Remember: Good observability is a journey, not a destination. Start with what you need most and grow your observability practice alongside your system.

About the Author

Tarique Ansari is an Architect at Happiest Minds with over 14 years of experience in architecting, designing, and delivering scalable, high-performance software solutions. He has proven expertise in leading cross-functional teams, particularly in the E-Commerce domain, while implementing innovative technologies and methodologies. Tarique is adept at leveraging cloud platforms, microservices, and agile practices to drive operational efficiency and business success.