Journal of Technological Innovations

Est. 2020

Elevating System Reliability through Observability in Cloud Native Applications

Savitha Raghunathan

Email: saveetha13gmail.com

Abstract:

The surge in cloud native technology adoption has fundamentally changed application development, deployment, and management, highlighting the need for scalability, resilience, and agility. In this environment, traditional monitoring is insufficient for ensuring the health and performance of cloud native applications, characterized by their distributed nature and dynamic operation. This whitepaper focuses on the pivotal role of observability in understanding and managing these complex systems. Dissecting observability's key components—logs, metrics, and traces—provides a detailed overview of how development and operations teams can gain real-time insights into system health, facilitating quick debugging and system optimization. This paper aims to provide organizations with the knowledge to build a comprehensive observability framework tailored to the unique demands of cloud native ecosystems.

Keywords: Observability, Alerting, Monitoring, Cloud Native, Logs, Metrics, Traces, Distributed Tracing

1. Introduction

The transition to cloud computing and the rise of cloud native technologies have revolutionized the software development lifecycle and operational practices. Applications now leverage the cloud's scalable and elastic nature, utilizing microservices architectures, containerization, dynamic orchestration, and continuous delivery models. These advancements, while beneficial, introduce significant complexity in maintaining system health and performance. The distributed architecture and ephemeral resource allocation require shifting from traditional monitoring to a more comprehensive observability approach to ensure applications perform reliably in production settings.

2. The Importance of Observability in Cloud Native Applications

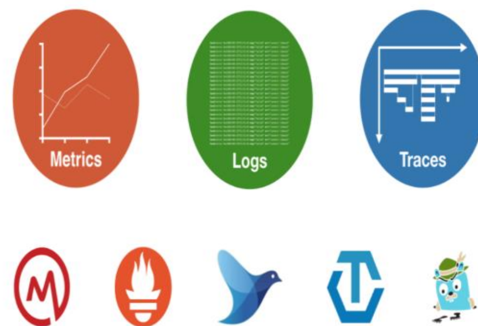

Fig 1: 3 Pillars of Observability [5]

Observability transcends traditional monitoring by tracking expected issues (known unknowns) and providing insights into unforeseen problems (unknown unknowns) [4]. As shown in Figure 1, It involves collecting and analyzing data from various system outputs—logs, metrics, and traces [1] —to infer the system's internal state. This capability is crucial in cloud native environments, where

applications' dynamic and distributed nature complicates the detection and diagnosis of issues.

Figure 2 below, shows the overlap between the three pillars of observability.

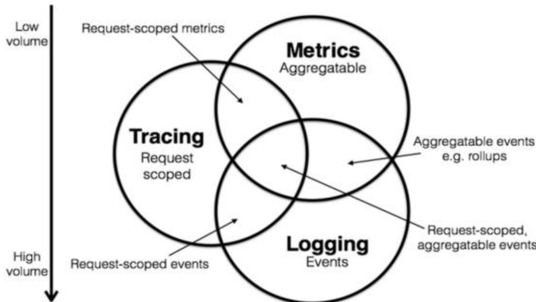

Fig 2: Overlap between Logs, Metrics, and Traces

Logs

Logs offer a chronological record of events within the application or infrastructure, which is crucial for troubleshooting and understanding system behavior.

Metrics

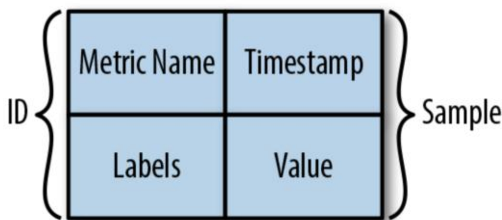

Fig 3: Metric sample from prometheus [1]

Metrics, numerical data collected over time, provide a snapshot of system health, including performance indicators like CPU usage, memory consumption, and request latency.

Traces

Traces record a request's lifecycle across different system components, which is essential for pinpointing performance bottlenecks and understanding microservices interactions.

3. Tools for Observability

The ecosystem offers a range of tools for enhancing observability in cloud native applications, spanning open source and commercial solutions. Key tools include Prometheus for metric collection and alerting, Grafana for data visualization, and distributed tracing systems like Jaeger and Zipkin [9]. These tools cater to the unique challenges of observing cloud native environments, facilitating comprehensive data collection and analysis. Selecting the right set of tools is crucial for effective observability. Organizations should consider:

- Tools like Prometheus [9] are popular in cloud native ecosystems for metrics and alerting due to their dynamic service discovery capabilities.

- For log aggregation and analysis, solutions like ELK (Elasticsearch, Logstash [9] , and Kibana), EFK (Elasticsearch, Fluentd, and Kibana), or Loki are widely used [10].

- For distributed tracing, Jaeger or Zipkin [9] can offer deep insights into the behavior and performance of microservices (Refer figure 4).

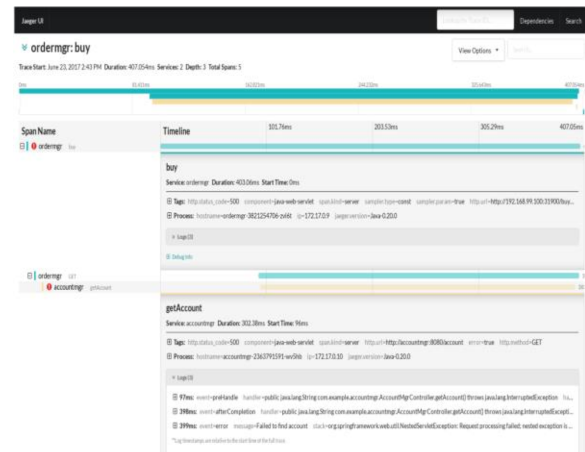

Fig 4: Jaeger trace showing three scans [2]

4. Implementing Observability into Cloud Native Platforms

To effectively integrate observability into cloud native platforms, organizations should:

Identify Key Data Points

The first step towards achieving comprehensive observability in cloud native platforms is identifying key data

points [3] that accurately reflect the system's health and performance. This process involves:

Defining Critical Metrics: Identifying metrics critical to the application's performance, such as latency, error rates, throughput, and system resource utilization (CPU, memory, disk I/O, network bandwidth).

Determining Log Data to Collect: Specifying the types of log data that offer insights into system operations, including error logs, transaction logs, and audit logs.

Understanding Tracing Requirements: Recognizing the need to trace data to help understand the flow of requests through the microservices architecture, including service interaction patterns and latency issues.

This process requires collaboration between development, operations, and business teams to ensure that the collected data aligns with technical and business objectives and provides a holistic view of system performance and health.

Select Suitable Tools

Compatibility with Technology Stack: The chosen tools should seamlessly integrate with the existing technology stack, including programming languages, frameworks, and infrastructure platforms.

Scalability and Performance: Tools must be capable of scaling with the application, handling large volumes of data without significant performance degradation.

Ease of Use and Integration: Tools that offer user-friendly interfaces and easy integration with other observability and development tools should be preferred, facilitating a smooth workflow for developers and operators.

Community and Vendor Support: Tools with active community support and robust vendor backing ensure access to resources, documentation, and assistance for troubleshooting and enhancements.

Instrument Applications

Instrumenting applications involves modifying or configuring application code and infrastructure to collect the necessary observability data:

Application-Level Instrumentation: Add libraries or agents to application code that enable the emission of metrics, logs, and traces. It involves using Prometheus client libraries, OpenTelemetry SDKs for tracing, or logging frameworks compatible with the chosen observability platform.

Infrastructure and Network Instrumentation: Leverage service mesh technologies like Istio or Linkerd to automatically capture telemetry data from microservices interactions without requiring changes to the application code.

Configure Data Analysis and Alerting

After data collection is set up, the next step involves configuring the analysis tools and setting up alerting mechanisms:

Analysis and Visualization: Use tools like Grafana [5] to create dashboards that provide real-time visualization of metrics, logs, and traces. Customize these dashboards to display key performance indicators and trends clearly and concisely.

Alerting Mechanisms: Configure alerts based on predefined thresholds for metrics or specific log events. It involves using tools like Alertmanager [6][7] with Prometheus or integrating the alerting capabilities of commercial observability platforms. The alerting system should be dynamic, allowing for easy adjustment of thresholds as the system evolves.

Cultivate a Culture of Observability

Implementing tools and processes is only part of the solution; cultivating a culture of observability within the organization is equally important:

Continuous Learning: Providing training and resources for teams to stay updated on best practices in observability, understanding the tools, and interpreting the data effectively.

Collaboration: Encouraging open communication and collaboration between development, operations, and business teams to ensure observability insights are shared and acted upon. This includes regular review

sessions to discuss observability findings and implications for system improvement.

Incorporating Feedback Loops: Using insights from observability data to inform development practices, system architecture decisions, and operational strategies. This continuous feedback loop helps refine observability practices and improve system reliability and performance.

By following these guidelines, organizations can effectively integrate observability into their cloud native platforms, enhancing system reliability, performance, and overall operational efficiency.

5. Best Practices for Maximizing Observability

Effective observability in cloud native environments requires:

- Integrating observability throughout the software development lifecycle [11].
- Balancing detailed data collection with the management of high-cardinality data.
- Automating alerts and anomaly detection to identify issues swiftly [8].
- Combining logs, metrics, and traces for a holistic view of system health [8].
- Ensuring scalability of observability tools and practices to accommodate growing applications and infrastructure.

6. Conclusion

In the evolving landscape of cloud native applications, observability is critical for maintaining system reliability and performance. By embracing observability, organizations can navigate the complexities of modern application architectures, ensuring their systems are resilient, performant, and capable of meeting the dynamic demands of the digital age. As technology progresses, the strategies for implementing observability will continue to advance, necessitating ongoing adaptation and learning to harness its full potential.

References

[1] C. Sridharan, "Distributed Systems Observability," O'Reilly Media, Inc., Jul. 2018. Available: https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html

[2] D. Mueller-Klingspor, "Using OpenTracing with Jaeger to Collect Application Metrics in Kubernetes," Red Hat Developer, Jul. 10, 2017. https://developers.redhat.com/blog/2017/07/10/using-opentracing-with-jaeger-to-collect-application-metrics-in-kubernetes#deploying_on_kubernetes

[3] A. Mukherji, "Four Steps to Implement an Observability Strategy for Microservices," DevOps.com, Oct. 28, 2019. https://devops.com/four-steps-to-implement-an-observability-strategy-for-microservices/

[4] T. Treat, "Microservice Observability, Part 1: Disambiguating Observability and Monitoring," Brave New Geek, Oct. 03, 2019. https://bravenewgeek.com/microservice-observability-part-1-disambiguating-observability-and-monitoring/

[5] M. Tan, "What's next for Observability," Grafana Labs, Oct. 21, 2019. https://grafana.com/blog/2019/10/21/whats-next-for-observability/

[6] M. Burillo, "Kubernetes Monitoring with Prometheus: AlertManager, Grafana, PushGateway (part 2).," Sysdig, Aug. 27, 2018. https://sysdig.com/blog/kubernetes-monitoring-with-prometheus-alertmanager-grafana-pushgateway-part-2/

[7] I. Huckova, "Step-by-step guide to setting up Prometheus Alertmanager with Slack, PagerDuty, and Gmail," Grafana Labs, Feb. 25, 2020. https://grafana.com/blog/2020/02/25/step-by-step-guide-to-setting-up-prometheus-alertmanager-with-slack-pagerduty-and-gmail/

[8] New Relic, In, "A Three-Phased Approach to Observability," Code Motion World, Jul. 2020. Available: http://mediarepository.codemotionworld.com/docs/3-phased-approach-to-observability.pdf

[9] G. Ouillon, "What Is Modern Observability?," New Relic, Inc., Jun. 18, 2020. https://newrelic.com/blog/best-practices/what-is-modern-observability

[10] B. Huo and D. Ma, "Cloud Native Observability: Log Management," KubeSphere, Jun. 25, 2019.

https://kubesphere.io/conferences/logging/
[11] J. Wills, “Instrumentation, Observability & Monitoring of Machine Learning Models,” InfoQ, May 28, 2019. https://www.infoq.com/presentations/instrumentation-observability-monitoring-ml/