ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Traditional Techniques and Emerging Technologies in Observability

Sriram Pollachi Subburaman* and Srividhya Chandrasekaran

USA

ABSTRACT

Identity and access management is the bedrock of cybersecurity. Identity access to digital resources is governed by its techniques, procedures, and rules, which also define the breadth of identity permission over those resources. Some new cyberattack or data breach pops up in the news every week. Many data breaches occur due to inadequate security measures, software flaws, human mistake, malevolent insiders, or the abuse of access and privileges. An improved access control system is possible with the use of AI methods. In order for organisations to better handle authentication and access control in order to reduce cyber risks and other IAM difficulties, studies into artificial intelligence in IAM are necessary. With an eye towards AI's potential uses in identity and access management - more especially in the areas of privilege monitoring, administration, and control - this research investigates the nature of the connection between AMIS and AI. To better understand how AI works in minimising recognised IAM issues, this study aimed to present evidence from the relevant literature. This study's results show how AI reinforces identity and access management, which helps with automating procedures, keeping up with technology advances, and reducing the prevalence of cyber threats. One way to accomplish this is by using a binary classification system for security access control, which takes the PDP problem and turns it into a yes/no question. In order to create a distributed, effective, and accurate policy decision point (PDP), a vector decision classifier is also built using the supervised machine learning technique. Kaggle-Amazon access control policy dataset evaluated performance by comparing the proposed mechanism to previous research standards in terms of performance, duration, and flexibility. Given that the PDP is not in direct contact with the PAP, the proposed approach accomplishes a high level of secrecy in relation to access control requirements. In conclusion, PDP-based ML can manage massive access requests, execute many major policies simultaneously, and have a 95% accuracy rate, all without policy conflicts, with a response time of about 0.15 s. The security of access control can be enhanced by making it more responsive, flexible, dynamic, and dispersed.

***Corresponding author**

Sriram Pollachi Subburaman, USA.

Received: December 05, 2022; **Accepted:** December 13, 2022; **Published:** December 19, 2022

Keywords: Distributed Infrastructures, Monolithic Architecture, Microservices Architecture, Quality of Service (QOS), Monitoring, Telemetry

Introduction

The evolution of distributed infrastructures, characterized by multiple components spread across networked computers that communicate and coordinate through message passing, has been significantly influenced by the imperative to attain exceptional Quality of Service (QoS). This pursuit of superior QoS is essential for fostering optimal Quality of Experience (QoE), driving notable advancements in distributed systems [1]. The advent of cloud computing ushered in the era of cloud-native development, where developers prioritize creating, deploying, and maintaining applications at hyper-scale over physical deployment locations [2,3]. This approach enables developers to focus entirely on continuous, agile software delivery, assuming that infrastructure is readily available. However, the rapid advancement of these paradigms is also causing a proliferation of new applications and services, stretching the capabilities of Ops teams to monitor and manage effectively. In recent years, the landscape of software architecture has undergone significant changes. One major change is the move away from large monolithic applications to smaller, more finely grained deployment units called microservices. These microservices communicate mostly through synchronous Representational State Transfer (REST) and asynchronous events, departing from the previous trend of larger and less flexible applications [4].

In static monolithic architecture, a database, UI and the backend are deployed as a single unit. Any change in the component, requires a build and deploy of entire application. In this case, the application developer is mainly responsible for collecting observability data.

Contrastingly, microservices architecture organizes components as a set of loosely coupled services deployable independently and managed by Ops teams. As developers decompose applications into microservices, transporting them in containers across distributed cloud providers and consistently redeploying them under DevOps supervision, the demand for meticulous observability becomes increasingly imperative. The evolving techniques for designing and managing distributed systems underscore the importance of observing services and infrastructure to ensure seamless operations.

Distinguishing Monitoring from Observability

Observability is defined as a measure of how well the internal state of a system can be inferred from its external outputs [5]. Telemetry data, including logs, traces, and metrics, are key external outputs for distributed systems like microservices. They encompass details such as machine resource usage and application-generated log-

level data. Observability offers both high-level system health overviews and detailed insights into implicit failure modes.

Monitoring is the process of collecting data and generating reports on different metrics that define system health monitoring and observability are often discussed together as they both contribute to maintaining system reliability [6]. While they share a common goal, there exists a nuanced difference between them, and they are, in fact, interconnected. To put it simply, observability does not replace monitoring, nor does it render monitoring unnecessary; rather, they complement each other and work in tandem. Monitoring notifies operators of operational failures, while observability helps identify the location, cause, and trigger of the failure.

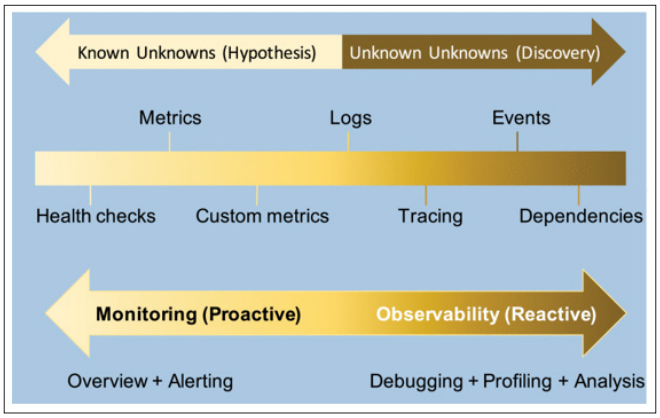

Figure 1: Observability vs Monitoring [7]

Industry Contributions to Observability and Monitoring

Monasca an open-source monitoring service, seamlessly integrates with OpenStack. It is designed to be multi-tenant, scalable, and fault-tolerant, providing a Representational State Transfer (RESTful) API for efficient metrics processing and querying. Additionally, it offers streaming alarm and notification engines [8]. Another widely used tool is Open Telemetry, employed to collect telemetry data from cloud-native applications and their underlying infrastructure. This enables the monitoring of their overall health and performance. Grafana Labs contributes an open-source monitoring and observability platform that facilitates querying, viewing, and alerting based on metrics [9,10]. The ELK stack, comprising Elasticsearch, Logstash, and Kibana, provides centralized logging, aiding in the discovery of application issues from a unified location [11].

For microservices monitoring, Apache Skywalking is an open-source observability tool engineered to assist operators in identifying issues, receiving critical alerts, and monitoring system health. Consul, a service mesh solution, offers a comprehensive control plane, encompassing service discovery, configuration, and segmentation capabilities [12,13]. Cilium, an open-source solution based on extended Berkeley Packet Filter (eBPF) technology, ensures secure network connectivity between services deployed on Linux container management platforms such as Docker and Kubernetes [14].

Prometheus, a Cloud Native Computing Foundation (CNCF) project, is a system that provides multi-dimensional time-series monitoring for resources and services. It collects metrics, evaluates rules, displays results, and issues alerts [15].

Among cloud-based vendors, Microsoft Azure Monitor collects and analyzes telemetry data from Azure and on-premises environments [16]. It monitors web application availability, performance, usage, and optimizes infrastructure performance. Amazon CloudWatch is utilized for monitoring AWS resources and applications, offering visibility into resource utilization, application performance, and operational health across the entire stack, from applications to supporting infrastructure [17]. Google Cloud Operations Suite encompasses various components such as Cloud Monitoring, Cloud Trace, and Cloud Logging, covering multiple observability dimensions [18].

Observability Datatypes

The three important observability data types are logs, metrics, and traces. Logs comprise structured and unstructured text lines generated by a system during the execution of specific code segments. Log monitoring is emphasized in [19]. Metrics are numeric representations of data utilized by Ops teams to assess the long-term behavior of a system, service, or network component. Metrics monitoring is emphasized in [20]. Traces illustrate the complete path of a request or action as it traverses different components within a distributed system. Trace monitoring is emphasized in [21].

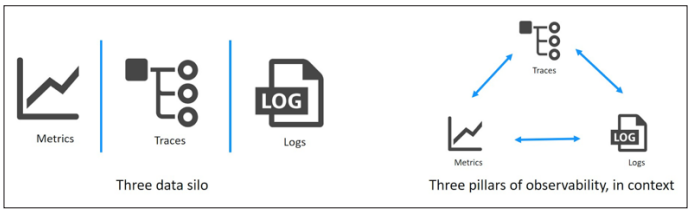

Figure 2: The Three pillars of observability [22]

Observability System Functionalities and Characteristics

Connection

Identify the potential anomaly from other anomalies. For instance, a modern observability system should collect and analyze data every second, presenting correlated information alongside anomalies for users to quickly identify potential root causes of issues.

Structure

With the dependency graphs in the tools, the system should be able to inject the right instrumentation automatically.

Response

The observability system should handle the issues dynamically. Granularity is a key characteristic of an observability system.

Interrelated Context

When operators review metrics regarding the health of a microservice, they should be capable of observing its impact on other services or components within the distributed system, as well as how those workloads are influenced by the Kubernetes cluster hosting them, and vice versa.

Simplify and Expedite Exploration

Imagine accessing all telemetry data in a single, near real-time view from any location. This innovative design should provide intuitive visualizations, requiring no configuration, enabling Ops teams to efficiently navigate large, intricate, distributed systems, and promptly identify and prioritize performance issues.

Unified Repository

This involves storing, alerting, and analyzing telemetry data using cohesive APIs, regardless of its distributed placement across

multiple nodes in an edge cluster. Typically, operators seek a platform capable of ingesting metrics, events, logs, and traces from various sources, including proprietary and open-source agents, APIs, and built-in instrumentation. However, accessing data through such unified APIs must be scalable to accommodate high ingest loads during peak periods.

Separate data sources from destinations: By decoupling sources and sinks, it should be simple to introduce or modify tools and redirect data without affecting production systems. For example, integrating a messaging bus into the observability system should fully separate the source and sink, eliminating direct interaction between them.

Observability Challenges

Based on the observability requirements and some of the best practices, we summarize some of the challenges in building observability tools concerning cloud architecture [23].

Microservice

Microservice architecture presents unique observability challenges due to its distributed nature. Each microservice operates on an isolated computing platform, such as a container, leading to interdependencies. Failures in one microservice can cascade throughout the system, impacting overall functionality. Therefore, beyond internal observability, understanding complex relationships among microservices is essential for a comprehensive view of the deployed system [24].

Serverless

Serverless platforms, typically furnish basic dashboards to monitor ongoing processes. However, these dashboards have limited capabilities, such as workload visualization and resource utilization reports, as they lack root access to the underlying operating system, making installation and configuration of additional observability tools impossible [25].

Containers

Collecting and analyzing telemetry from distributed containers is challenging, especially considering the dynamic nature of container deployments. Therefore, practitioners often prefer observability solutions integrated with container orchestration platforms like Kubernetes [26].

Heterogenous Data

Analyzing the diverse observability data poses challenges, requiring cleaning, formatting, and cross-referencing with time as the common factor.

Infrastructure

Merging logs, metrics, and traces gathered from infrastructure components like networks, servers, VMs, containers, and serverless functions with other telemetry such as application and OS data is essential for obtaining a comprehensive view. The complexities of infrastructure monitoring have been addressed in our prior research [27,28].

Conclusion

This paper has explored various strategies for achieving comprehensive observability in both applications and infrastructure within modern software systems. By leveraging a combination of traditional monitoring techniques and emerging technologies such as distributed tracing and service mesh architectures, organizations can enhance their operational visibility and troubleshooting capabilities. Through practical case studies and implementation guidelines, we have demonstrated the importance of observability in ensuring the reliability and performance of software systems in diverse environments. Moving forward, continued research and adoption of observability best practices will be essential for meeting the evolving needs of complex and dynamic software ecosystems [29].

References

1. Alimi IA, Patel RK, Zaouga A, Muga NJ, Xin Q, et al. (2021) Trends Cloud Computing Paradigms: Fundamental Issues Recent Advances and Research Directions Toward 6G Fog Networks. Rijeka, Croatia:IntechOpen https://www.intechopen.com/chapters/77006.
2. Hurwitz JS (2020) Cloud Computing for Dummies. Indianapolis, IN, USA:Wiley http://www.it-docs.net/ddata/372.pdf.
3. Gannon D, Barga R, Sundaresan N (2017) Cloud-native applications. IEEE Cloud Comput 4: 16-21.
4. Lauretis L (2019) From monolithic architecture to microservices architecture. Proc IEEE Int Symp Softw Rel Eng Workshops (ISSREW) 93-96.
5. What Is Observability. O'Reilly Online Learning https://www.oreilly.com/library/view/observability-engineering/9781492076438/ch01.html.
6. Observability vs Monitoring - Difference Between Data-Based Processes – AWS. Amazon Web Services, Inc.
7. Usman M, Ferlin S, Brunstrom A, Taheri J (2022) A Survey on Observability of Distributed Edge & Container-Based Microservices. IEEE Access 10: 86904-86919.
8. (2022) Monasca. OpenStack https://wiki.openstack.org/wiki/Monasca.
9. (2022) High-quality, ubiquitous, and portable telemetry to enable effective observability. OpenTelemetry https://opentelemetry.io/.
10. (2022) Grafana Loki. Grafana Labs https://grafana.com/oss/loki/.
11. (2022) Logstash: Collect Parse Transform Logs. Elastic https://www.elastic.co/logstash.
12. (2022) Apache SkyWalking. SkyWalking https://skywalking.apache.org/.
13. (2022) Identity-based networking with Consul. HashiCorp. https://www.consul.io/.
14. (2022) Introduction to Cilium & Hubble. Cilium https://docs.cilium.io/en/stable/overview/intro/.
15. (2022) From metrics to insight Power your metrics and alerting with the leading. open-source monitoring solution. Prometheus https://prometheus.io/.
16. Sahay R, Sahay R (2020) Azure monitoring. Microsoft Azure Architect Technologies Study Companion: Hands-on Preparation and Practice for Exam AZ-300 and AZ-303, Berkeley, CA, USA:Apress 139-167.
17. What is Amazon CloudWatch? AWS https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.
18. (2022) Google Cloud's operations suite (formerly Stackdriver). Google Cloud https://cloud.google.com/products/operations.
19. Cinque M, Della Corte R, Pecchia A (2019) Advancing monitoring in microservices systems. Proc IEEE Int Symp Softw Rel Eng Workshops (ISSREW) 122-123.
20. Brandón A, Pérez MS, Montes J, Sanchez A (2018) FMonE: A flexible monitoring solution at the edge. Wireless Commun Mobile Comput 2018: 1-15.
21. Fonseca R, Porter G, Katz RH, Shenker S, Stoica I (2004)

X-Trace: A pervasive network tracing framework. Proc 4th USENIX Symp Netw Syst Design Implement 14.

22. (2021) Unified observability and security. Dynatrace https://www.dynatrace.com/.
23. Pourmajidi W, Zhang L, Steinbacher J, Erwin T, MiranskyyA (2023) A Reference Architecture for Observability and Compliance of Cloud Native Applications. Arxiv https://arxiv.org/pdf/2302.11617.pdf.
24. Levin J, Benson TA (2020) Viperprobe: Rethinking microservice observability with ebpf. 2020 IEEE 9th International Conference on Cloud Networking (CloudNet) 1-8.
25. Cordingly R, Heydari N, Yu H, Hoang V, Sadeghi Z, et al. (2021) Enhancing observability of serverless computing with the serverless application analytics framework. Companion of 16 the ACM/SPEC International Conference on Performance Engineering 161-164.
26. Moradi F, Flinta C, Johnsson A, Meirosu C (2017) Conmon: An automated container based network performance monitoring system. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) 54-Z62.
27. Pourmajidi W, Steinbacher J, Erwin T, Miranskyy A (2017) On challenges of cloud monitoring. Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering 259-265.
28. Pourmajidi W, Zhang L, Miranskyy A, Steinbacher J, Godwin D, at al. (2021) The challenging landscape of cloud monitoring. Knowledge Management in the Development of Data-Intensive Systems 157-189.
29. (2018) Lessons from Building Observability Tools at Netflix. Medium https://netflixtechblog.com/lessons-from-building-observability-tools-at-netflix-7cfafed6ab17.