Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Optimizing Full-Stack Application Performance through End-to-End Observability and Real-Time Diagnostics

Sudeep Annappa Shanubhog
Tential Solutions, USA

ARTICLE INFO

Received: 06 Feb 2026

Revised: 08 Feb 2026

ABSTRACT

In full-stack applications, performance problems are often not limited to the frontend or backend but can affect the entire stack as distributed architectures become increasingly common. When an issue arises in one component, it may quickly cascade throughout the application stack, leading to performance degradation. Customary application monitoring tools tend to provide visibility into different system layers in isolation. Because distributed systems can involve many interconnected components, end-to-end observability seeks to overcome these shortcomings by providing a single view of the application throughout the entire application stack. Telemetry is typically divided into three groups: logging, metrics, and distributed tracing, which provide different views of system behavior. Standardized telemetry collection allows interoperability between heterogeneous technology stacks, and effective instrumentation is the foundation of observability. Frontend instrumentation provides real user monitoring, whereas backend instrumentation provides server-side metrics, such as request latency distribution and the percentage of requests that experience failures. By relating user-reported latency to the backend processing time, targeted optimizations can be performed. Observability platforms use machine learning algorithms to uncover patterns that are overlooked by threshold-based alerts. Large language models are promising candidates for automating root cause analysis in complex cloud systems.

Keywords: End-to-end Observability, Distributed Tracing, Performance Optimization, Anomaly Detection, Telemetry Instrumentation

I. Introduction

Because modern full-stack applications are tightly coupled, multi-layered systems, the performance impact of problems propagates and quickly gets distributed to the entire stack. This poses an important challenge for developers in providing a good user experience. Additionally, research on the complexity of websites shows that more than half of all websites have a median page load time of over 2 seconds [1], and the user experience quickly deteriorates after this point. Approximately 49% of users abandon an app or switch to a competitor after performance issues [1]. The relationship between latency and business results has been well-documented in the industry.

Customary monitoring systems assess the individual components of a system in isolation and are thus unable to account for synchrony and dependency issues introduced by the system's distributed architecture, with modern web pages accessing multiple servers in multiple administrative domains. More than 60% of webpages fetch content from at least five non-origin servers when loading [1]. Because external services account for over 35% of the total bytes downloaded in median cases [1], this distributed nature carries many potential points of failure. All external dependencies contribute to the total latency and complicate the performance diagnosis. Supervisor tools designed for monolithic applications are insufficient for distributed applications [2].

The transition to microservice development models also exacerbates observability problems. Microservices require the decomposition of functionality into many independent components, each running in containers and managing functionality for a specific portion of the business logic [2]. Current solutions for orchestration do not provide built-in observability features for this architectural

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

style [2]. Monitoring tools are mainly designed around deployment management and resource utilization models, but do not support advanced monitoring of the characteristics of distributed applications [2]. Observing metrics that describe the interactions between the components of an application requires special-purpose tooling.

End-to-end observability seeks to address this by providing visibility across the application stack, combining front-end user experience metrics with back-end service performance metrics, and correlating these data with infrastructure health information, where applicable. Cloud-native observability seeks to provide full context indicators of the health and status of applications [2]. Unlike monitoring, observability focuses on the internal state of a system. Analogous to control theory, observability is the extent to which internal states can be inferred from external outputs [2]. It can also be applied to the collection and correlation of multiple signals in distributed systems.

By considering multiple measurable dimensions, the content complexity of web applications includes the number and size of objects that are retrieved to render a web page. Service complexity is the distribution of service resources over servers and administrative domains [1]. Research indicates that the number of requested objects is the dominant predictor of client-perceived loading time [1]. Approximately 20% of web pages include more than 100 objects on their homepage [1]. These measures of complexity yield an understanding of performance optimization, as discussed below. Observability frameworks that correlate such measurements at the front-end, back-end, and infrastructure boundaries provide thorough real-time diagnostics.

II. Related Work

Earlier work in distributed system monitoring led to the development of telemetry and log viewers that focused on individual layers of system complexity. These siloed views are inadequate for understanding the interdependencies in modern systems. MonitorRank applies a random walk to call graphs to provide a likelihood ranking of the root causes in service-oriented environments. Frontend and backend services can also be compared based on the similarity in the patterns of execution. ShiViz is the first tool to visualize the execution of distributed systems as a time-space diagram. It supports the visualization of distributed execution traces by linking the visualization elements to the corresponding entries in the execution log. This framework integrates the three pillars of telemetry into a holistic observability architecture. Logging captures ordered records of the system activity. Metrics are periodic measurements of performance, and distributed tracing is the reconstruction of entire transactions across service boundaries. Instrumentation consists of front-end, back-end, and infrastructure techniques. Deep neural networks suitable for time-series data can perform probabilistic outlier detection. The Long Short-Term Memory architecture can capture the hybrid dynamic behavior of nonlinear physical plants combined with digital controllers. Large language models automatically forecast the root cause after ingesting and processing diagnostic messages using chain-of-thought prompting. RCACopilot scales are deployed in production cloud-based environments in a broader enterprise context.

III. Cross-Layer Observability Framework

A. Architectural Components

Logs, metrics, and distributed traces are the foundational building blocks of observability that allow us to assess the visibility of a given system. These views provide complementary insights into the system and often offer synergistic insights. Industry surveys often find that 88% of organizations are at some stage of observability, with 31% at full-stack observability [3]. Logs are the most established form of observability, and are tracked sequentially at the application, infrastructure and network levels throughout a system's activities. Log entries typically consist of a timestamp, severity level, component that produced the log, and description of the event. Organizations with high observability maturity experience a 69% reduction in the meantime to resolve unplanned downtimes [3]. Well-

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

designed logging implementations in distributed computing environments include a correlation token to help analyze cross-boundary activities between organizations and technologies.

Metrics are numerical measurements collected periodically and used to analyze trends, establish baselines, and detect anomalies in a technological environment. The basic types of metrics can be classified taxonomically. Amassing counters monotonically increase, and instantaneous gauges are point-in-time measurements. Statistical distributions are the ranges of values and frequencies [3]. Organizations with mature observability practices have achieved an average of 66% faster mean time to resolution for customer-impacting incidents [3]. Modern observability architectures implement multidimensional metric models that associate core metrics with their contextual dimensions.

Distributed tracing addresses these visibility and tracking challenges owing to architectural changes, where a single transaction often spans multiple independent services. This observability dimension reconstructs the entire transactional journey. Implementation frameworks propagate correlation identifiers across service boundaries, and services may also write span records that document their behavior [3]. Companies at the high end of observability have a 63% lower frequency of outages [3]. Trace files can be converted into directed graphs, translating abstract distributed interactions into specific concrete execution paths.

B. Telemetry Standards and Protocols

Telemetry collection is standardized to enable interoperability with heterogeneous technology stacks. Open instrumentation protocols define vendor-agnostic application programming interfaces for generating traces, metrics, and logs using standardized semantic conventions. Trace context propagation headers enable the propagation of traces across the service boundaries [4]. Organizations adopting these standards will achieve faster time to market and more flexibility than proprietary instrumentation approaches.

For cloud-native systems, orchestration and communication can be difficult to manage because of the distributed nature of the system. System-wide visibility becomes challenging with an increasing number of microservices [4]. Prometheus, Grafana, and Jaeger are used for monitoring, statistics, visualization, and diagnostics of observability backends. While Prometheus collects and analyzes metrics data, Grafana is used to visualize, and Jaeger is used for analyzing bottlenecks in service-to-service communication with distributed tracing [4]. These tools provide information that development teams can use to debug and resolve these issues.

Sampling strategies attempt to balance the depth of observability against performance degradation. Instrumenting every aspect of high-volume systems generates excessive data. More advanced collection methodologies use adaptive sampling, where the sampling rate is adjusted depending on the characteristics of the sampled transaction to ensure maximum visibility of anomalous transactions with the lowest possible telemetry overhead for normal transactions [3]. Service meshes provide features for securing and monitoring traffic among microservices. Traffic management and observability platforms help prevent black holes and improve resilience [4].

Telemetry Component	Primary Function	Data Characteristics	Key Application
Logging	Sequential activity documentation	Timestamp, severity, component identifier	Behavioral deviation identification
Metrics	Performance indicator sampling	Counters, gauges, distributions	Trend identification and baseline establishment
Distributed Tracing	Transaction journey reconstruction	Span records with timing information	Execution pathway visualization
Context Propagation	Trace continuity across boundaries	Correlation headers	Cross-service request mapping

Table 1: Telemetry Pillars and Functional Characteristics in Enterprise Observability Systems [3, 4]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

IV. Instrumentation Techniques for Modern Applications

Instrumentation is an integral part of full-stack observability, consisting of the deployment of telemetry collection capabilities in systems to monitor relevant performance metrics. The balance between comprehensiveness and the effort to implement the approach is key; developer implementations are informative but require wide-ranging coordination and collaboration between teams [5]. Using agents or transparent instrumentation can help reduce implementation overhead while limiting the available configuration options.

Frontend instrumentation metrics are gathered via real-user monitoring (RUM) on the client side. These include page load time, time to first contentful paint, and cumulative layout shift, all of which can affect the user experience. Over 50% of websites have median page load times of over 2 seconds, which can act as a baseline for frontend monitoring [5]. Most modern full-stack frameworks are equipped with built-in instrumentation frameworks to facilitate data collection from applications. Including security in instrumentation design avoids the creation of vulnerabilities during telemetry collection.

Backend instrumentation helps to quantify the server-side performance. Request latency distributions inform the application endpoint processing efficiency. The failure points of a system can be identified by its error rates, whereas capacity limitations can be identified by resource usage patterns [5]. Applications with full backend instrumentation are 25% faster to deploy and 40% cheaper to run than minimally instrumented applications. Modern applications are layered and require instrumentation across all layers. Application codes require probes to monitor business logic. Middleware components should have built-in automated instrumentation for cross-cutting concerns that cross-cut application layers so that developers are not required to change the code.

Infrastructure telemetry data include container metrics, network throughput, and system resource saturation. A few studies on containerized environments have shown complications in selecting infrastructure metrics [6]. The set of metrics can be very large, with more than 70 measurements of system load, CPU, memory and disk activity, network utilization, and process activity. The overhead of collecting these metrics can vary widely when considering the experimental load. Dedicated experiments were conducted on individual servers with four cores clocked at 3.6 GHz using 16 GB of RAM [6], measuring that as the number of metrics collected increased, the measurement overhead on the monitored system increased.

The right metrics depend on the workload requirements and the nodes on which they run. Node-level metrics are exposed by system interfaces and show infrastructure behavior. Statistics at the processor level (I/O, average processor load) provide a general idea of the system status [6]. Task statistics obtained through application instrumentation can provide a more accurate view of the performance. Experiments with periodic workloads, with a 10 ms period and 1.9 ms runtime budget, indicate real-time performance monitoring [6].

Different sampling strategies provide different depths of observability and overheads on the system, but sampling nodes every three seconds provides reasonable granularity and low overhead [6]. The monitoring parameters were set based on the isolation guarantees provided by the underlying systems. The experimental study had a data collection window of 30 min, with 80 s of cooling between different measurement phases of a campaign [6]. Anomaly detection may require sampling over a short period to observe state evolution. Well-isolated resources can be sampled less often.

Some database queries are difficult to track using application-layer monitoring. Database query instrumentation reveals performance issues and exposes slow queries. A query-level instrumentation organization can achieve 50% better performance by removing bottlenecks one at a time. As performance in the database is critical for application performance, instrumentation is a worthwhile investment. Combining infrastructure telemetry with application metrics (foundation-level visibility) enables diagnostics from the infrastructure through the application to the end-user.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

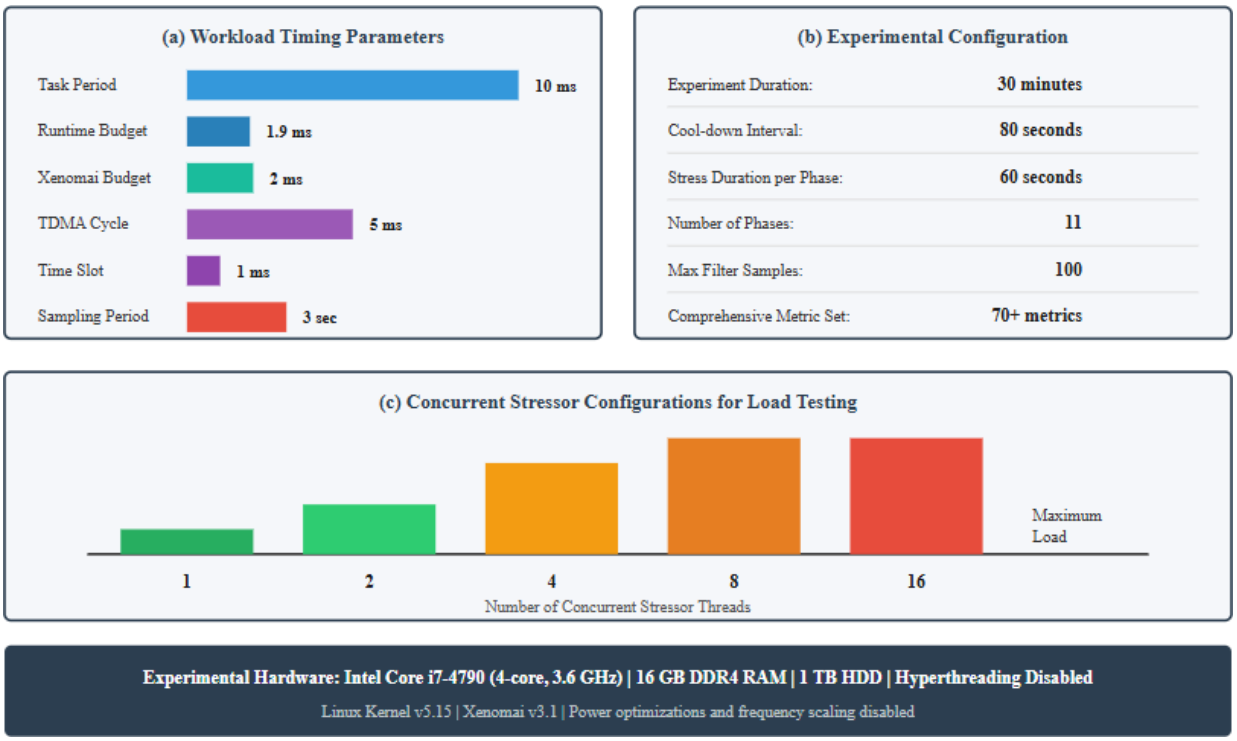

Figure 1: Instrumentation Parameters for Full-Stack Application Monitoring [5, 6]

V. Correlating Frontend and Backend Performance

The development of effective performance optimization techniques maps user-perceived latency to back-end latency. Similar to other aspects of modern service-based architectures, this is divided into a collection of services. User behavior often spans multiple layers, and understanding how frontend rendering delays relate to service call delays is important for diagnosing performance issues. Injecting trace context into API calls eases tracing across architectural layers [7].

These websites consist of many services, each of which runs on several machines. Each service handles a portion of the requests that are routed to the appropriate service. To create the call graph of a request, the frontend service receiving the request sends it out to the downstream services to collect and process the data [7]. These callee services can be further expanded. For example, LinkedIn operates over 400 services on thousands of machines scattered across a few data centers [7], making it very difficult to achieve performance correlation or to diagnose problems.

To solve the correlation problem, MonitorRank uses an unsupervised algorithm for metric classification, inferring performance dependency between services either from correlations observed in historical or recent time series (measurements of the same metric over time), from the structure of the service call graph, or from similarities between the frontend and backend services [7]. Anomalous performance in one service often cascades to downstream services [7]. The algorithm uses random walk algorithms over the call graph to obtain a ranking of the possible root causes. Experiments using it on production outages showed improvements of 26%–51% in mean average precision compared to baseline methods [7]. Thus, this approach considerably reduces the time, domain knowledge, and human effort required for performance diagnosis.

Visualization techniques complement algorithmic correlations by presenting the behavior of distributed systems in a more understandable manner. For instance, time-space diagrams visualize the happens-before relation between events on different hosts [8]. ShiViz is a tool that utilizes execution logs to create interactive visualizations of distributed system traces by linking the

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

visualization to the underlying text-based log information [8]. Developers can use these visualizations to understand event ordering and communication between tasks.

Our visualization-based correlation analysis can help with three tasks. First, ordering relevant events in time (e.g., whether backend latencies precede frontend performance degradation). Second, querying interaction patterns to discover communication topologies that lead to bottlenecks [8]. Third, execution pairs can be used to identify the structural differences between strong and weak system states. In controlled experiments with 39 subjects, better performance in answering system-comprehension questions was shown with visualizations rather than raw log data [8].

The evaluation of ShiViz's effectiveness showed its impact on users' understanding of the logs. In the treatment group, 79% of the participants who saw ShiViz got event ordering questions right, compared to 47% in the control group who looked only at raw logs [8]. ShiViz treatment performed considerably better in recognizing request-response patterns, correctly doing it in 96% of the trials compared to 33% in the control group [8]. The tasks with the largest disparity between groups were those that required participants to identify hosts with similar host behaviors. For these tasks, the ShiViz group achieved 61% accuracy. The control group answered only 7% of the same questions correctly [8]. Thus, visualization allows developers to better understand the execution behavior of distributed systems.

By providing a waterfall view of the end-to-end request flow, developers can identify the bottlenecks on the critical path. The searching capability allows developers to generate relevant event sub-graphs and locate them in the execution trace [8]. These subgraphs may include filters on host identifiers and event metadata, thus allowing for graph-based correlation and visual discovery of application logic. This, in turn, enables targeted optimization, improving performance on an app-wide basis.

Evaluation Method	Metric Category	Performance Outcome
MonitorRank Algorithm	Mean average precision improvement over baseline	26-51% gain
MonitorRank Algorithm	Root cause ranking accuracy in service-oriented architectures	Statistically significant improvement
MonitorRank Algorithm	Reduction in diagnostic time and domain expertise requirements	Demonstrated reduction
ShiViz Visualization (Treatment Group)	Event ordering comprehension accuracy	79%
ShiViz Visualization (Treatment Group)	Request-response pattern detection accuracy	96%
ShiViz Visualization (Treatment Group)	Host behavior identification accuracy	61%
Log Reading Only (Control Group)	Event ordering comprehension accuracy	47%
Log Reading Only (Control Group)	Request-response pattern detection accuracy	33%
Log Reading Only (Control Group)	Host behavior identification accuracy	7%

Table 2: MonitorRank Algorithm Effectiveness and Visualization Tool Accuracy Metrics [7, 8]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

VI. AI-Driven Anomaly Detection and Predictive Diagnostics

Machine learning algorithms are useful for observability platforms if they can detect patterns of anomalous behavior that would not be clear through threshold-based alerting. Rule-based approaches require knowledge of the system's configuration and operation logs. Models of cyber-physical systems are difficult to create because algorithmic control is closely linked to complex physical processes [9]. Unsupervised machine learning can be used to model a system, using the data logs collected by historians.

DNNs for time series have been shown to be effective at anomaly detection in distributed systems. The Secure Water Treatment testbed has been evaluated using DNN and one-class SVM models for 36 different types of attacks [9]. The DNN has 0.98295 precision, 0.67847 recall, and 0.80281 F measure; the one-class SVM has 0.92500 precision, 0.69901 recall, and 0.79628 F measure [9]. DNN achieves a lower false positive rate than the SVM, but the SVM detects more anomalies. Long Short-Term Memory architecture captures the dynamic behavior of hybrid systems, which combine non-linear dynamical processes with digital control logic [9]. Probabilistic outlier detection is based on the assumption that outliers will have a lower probability than other points.

Root cause analysis (RCA) for cloud services is a challenge. The size and complexity of the systems being monitored, as well as the sheer amount of data generated, make customary approaches to RCA - gathering and analyzing logs, metrics, and traces manually - time-consuming and error-prone, not to mention extremely difficult due to the varying degrees of available data. This continuum of available information, from little or none to an abundance of information, is termed the information spectrum. At the two extremes of this spectrum, root cause analysis can be particularly challenging to execute.

LLMs are promising candidates for improving the automation of RCAs, and RCACopilot addresses this problem by combining the gathering of diagnostic information and the LLM-based prediction of the RCs [10]. It achieves a Micro-F1 score of up to 0.766, and a Macro-F1 score of up to 0.533, for cloud incidents root cause categories predictions. A version of this component to retrieve diagnostics data has been used by more than 30 Microsoft teams for more than 4 years [10]. The average time spent on inference per incident is 4.205 seconds, and the root causes of incidents are often similar or identical. Research shows 93.80% of recurrent incidents are retriggered within 20 days of their occurrence [10].

Chain-of-thoughts prompting allows LLMs to list intermediate reasoning steps to make a prediction. Diagnostic information can be further distilled into a summary containing sufficient detail to make an accurate prediction. This matching between incident handlers and LLM components increases the adaptability and scalability of incident response. Since the LLM maps diagnostics to numeric vector spaces, the Euclidean distance between points becomes equivalent to the semantic distance between incidents. To return recommended demonstrations, nearest neighbor search on a combination of Euclidean and temporal distance is employed [10].

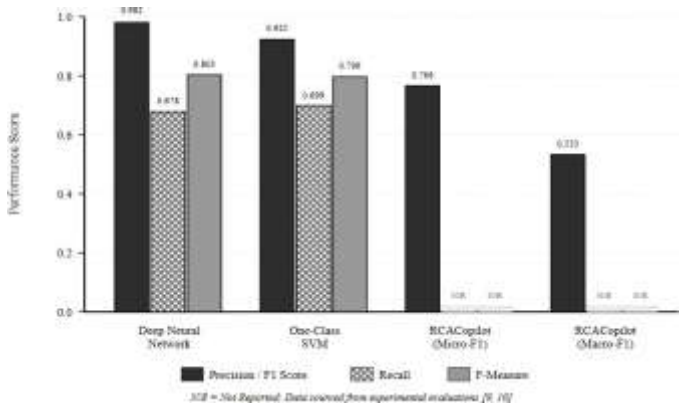

Figure 2: Comparative Analysis of Machine Learning Methods for Anomaly Detection [9, 10]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Conclusion

To optimize the performance of the entire application stack in a complex distributed environment, end-to-end observability distributes and standardizes the collection of telemetry data, cross-correlates that data throughout the application stack, and uses machine learning to cleverly and reliably detect and respond to performance problems. Service-oriented architectures decompose applications into many independent services, and a single user request can pass through multiple layers. An understanding of the relationship between the time spent rendering the page on the client and the specific calls to the backend service is key to performance debugging. Logging, metrics, and distributed tracing form the foundation. Each telemetry perspective offers a view unattainable from other perspectives: instrumentation of the frontend, backend, and infrastructure together provides the data needed for a full picture of performance. An alternative is to use time-space diagrams to model the happens-before relations between events on each host. This can improve understanding of distributed systems. Deep neural nets have also been used with time series data to identify anomalies in complex systems. Large language models can predict the root cause by prompting the collection of diagnostics and prompting the chain of thoughts. Future work includes automated remediation systems and advanced causal inference methods to identify multi-factor performance degradation in distributed cloud environments.

References

[1] Michael Butkiewicz et al., "Understanding Website Complexity: Measurements, Metrics, and Implications," ACM, 2011. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2068816.2068846

[2] Ioannis Tzanettis et al., "Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications," MDPI, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/5/2061

[3] Madhu Garimilla, "Designing Tomorrow's Observability: A Software Architect's Guide to Building Effective Monitoring Solutions," International Journal for Research in Applied Science & Engineering Technology, 2024. [Online]. Available: https://www.researchgate.net/profile/Madhu-Garimilla/publication/383862457_Designing_Tomorrow's_Observability_A_Software_Architect's_Guide_to_Building_Effective_Monitoring_Solutions/links/66de04fc64f7bf7b19a22177/Designing-Tomorrows-Observability-A-Software-Architects-Guide-to-Building-Effective-Monitoring-Solutions.pdf

[4] Vincent Uchenna Ugwueze, "Cloud Native Application Development: Best Practices andChallenges," International Journal of Research Publication and Reviews, 2024. [Online]. Available: https://www.researchgate.net/profile/Vincent-Ugwueze-2/publication/387296473_Cloud_Native_Application_Development_Best_Practices_and_Challenges/links/67757c05117f340ec3ea81f0/Cloud-Native-Application-Development-Best-Practices-and-Challenges.pdf

[5] Eunice Kamau et al., "Advances in Full-Stack Development Frameworks: A Comprehensive Review of Security and Compliance Models," International Journal of Multidisciplinary Research and Growth Evaluation, 2023. [Online]. Available: https://www.researchgate.net/profile/Gideon-Babatunde-2/publication/388102298_Advances_in_Full-Stack_Development_Frameworks_A_Comprehensive_Review_of_Security_and_Compliance_Models/links/683ec0f96a754f72b5902e30/Advances-in-Full-Stack-Development-Frameworks-A-Comprehensive-Review-of-Security-and-Compliance-Models.pdf

[6] Marco Barletta et al., "Criticality-aware Monitoring and Orchestration for Containerized Industry 4.0 Environments," ACM Transactions on Embedded Computing Systems, 2024. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3604567

[7] Myunghwan Kim et al., "Root Cause Detection in a Service-Oriented Architecture," ACM, 2013. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2494232.2465753

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[8] Ivan Beschastnikh et al., "Visualizing Distributed System Executions," ACM Transactions on Software Engineering and Methodology, 2020. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3375633
[9] Jun Inoue et al., "Anomaly Detection for a Water Treatment System Using Unsupervised Machine Learning," arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1709.05342
[10] Yinfang Chen et al., "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents," ACM, 2024. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3627703.3629553