The Essential Guide to

Observability for Cloud Native Environments

How to use observability to conquer complexity in cloud native environments

splunk>

How are you managing the increasing complexity of your distributed systems?

Embracing observability is a crucial step in a cloud-native journey.

Table of contents

Introduction

The cloud is here. Gone are the days of quarterly feature updates and waterfall deployments from IT and development teams. Increasingly, modern digital businesses rely on cloud native environments to deliver value to customers faster. Amazon, Netflix and Uber are the model for modern digital business. They have used the cloud to disrupt their industry and leave competitors in the dust.

However, while being cloud native helps a business scale, increase reliability and performance, and improve feature velocity, there are new challenges that engineering teams must face. Complexity, system performance and cost are among the top issues engineering leadership struggle with. When a customer-facing issue occurs, scoping and identifying the problem may require time and people to understand the hundreds of dependencies, APIs, serverless functions or third party components in a modern microservice architecture.

One service that lacks proper scalability or resource optimization, or that is experiencing latency or network connectivity problems, can impact another service and ultimately customers. Cost concerns from automatically scaling infrastructure or inefficient utilization of infrastructure are another frequent challenge.

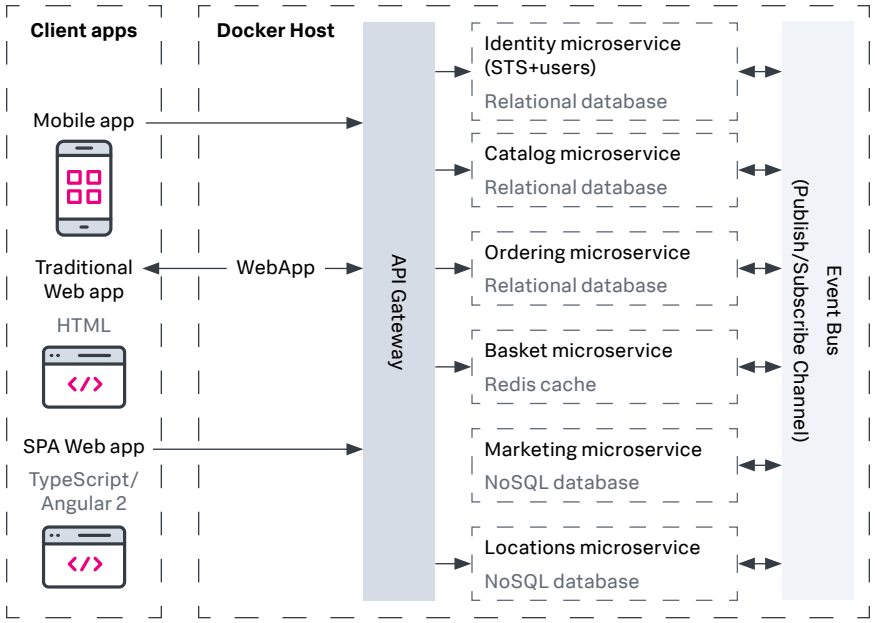


Figure 1: Cloud-native design

Why observability is a must-have to operate distributed systems

Engineering teams with cloud native environments rely on modern observability solutions to provide visibility across their environment, optimize system performance and help quickly troubleshoot problems. While many teams use observability when problems occur, teams increasingly use observability to proactively catch problems and ensure they deliver satisfying reliability and performance earlier in the development lifecycle.

Observability can even integrate with a CI/CD system to catch issues before they go into production at all. Additionally, as macroeconomic conditions increase pressure on budget decisions, observability solutions provide visibility and governance to help engineering leaders control the cost of cloud infrastructure and data usage as a system scales.

Understanding the challenges of cloud native

Cloud native environments offer benefits galore, like faster time to market and improved reliability — but they also introduce complexities that make it difficult to monitor the overall health and performance of a system. Engineering leaders can also feel squeezed between prioritizing system resilience and innovation as they work to both optimize customer experience and drive innovation.

Increased complexity in distributed systems

While cloud native environments help businesses achieve better time to market and increased reliability, they come with challenges. Microservices, containers and orchestration platforms add an explosion of complexity to a system, making it difficult to get a clear picture of health and performance of a system as a whole. While distributed systems enable applications to scale resources effectively and automatically, there is more to monitor and react to.

To put it simply, you can't predict what will go wrong, or how one new change will impact interconnected system components. Traditional monitoring built for a time of monolithic applications, on-prem infrastructure, and static web pages often falls short at providing an entire picture of infrastructure health,

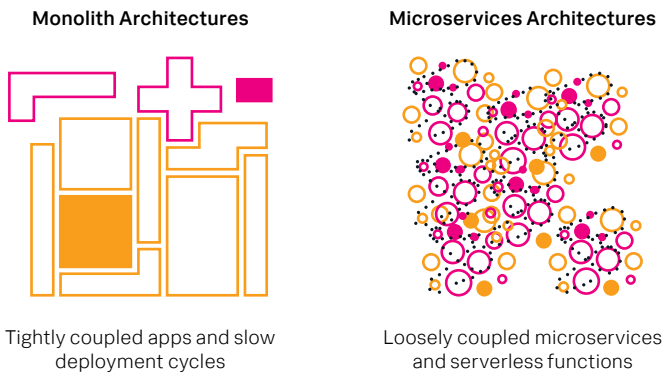


Figure 2: Monolithic vs Microservice Architectures

application performance, end user experience, and network behavior. This can lead to blindspots for teams who need to understand if new changes have impacted system performance.

Difficulty identifying, troubleshooting and resolving issues

As an engineer deploys new code, configurations or updates to their specific service, it's always possible it will negatively impact another portion of the system. The result can be new slowness, errors or anomalies that degrade customer experience or business outcomes. Alerting, isolating and resolving issues across cloud native environments can be daunting. With numerous services communicating with each other, pinpointing the root cause of problems becomes time-consuming and complex. No single engineer has context to quickly troubleshoot problems across the entire system.

Traditional monitoring solutions that sample data — or offer piecemeal visibility across distributed systems — can lead to longer delays in resolving an issue, and require more people to understand what's wrong. Additionally, many legacy monitoring systems don't provide visibility into the user's experience. Given how much of modern applications run in browser, this can be a critical mistake.

The balancing act: system resilience vs. innovation

Heads of engineering often must choose between system resilience and driving new innovation. While delivering new features and functionality to the market is necessary to grow a business, every change brings risk to system resilience. While most monitoring solutions provide the basics for alerting and availability, uptime alone is no longer sufficient for customer facing applications.

Prioritizing what to fix first reduces burnout and makes sure that customer experience and business metrics are kept as healthy as possible.

The promise of observability

Defining observability and its role in modern engineering practices

Observability refers to the ability to gain insight into the internal workings of a system by observing its outputs and behaviors. In modern engineering practices, observability has become a crucial concept as it goes beyond traditional monitoring by emphasizing the holistic understanding of complex cloud native environments. It allows engineers to gain real-time visibility into the performance, health, and behavior of distributed systems, enabling them to identify and resolve issues proactively.

Key components of observability: metrics, traces and logs

Observability is achieved through the combination of three essential components: metrics, traces and logs. Metrics focus on quantitative measurements, such as response times, error rates and resource utilization, enabling performance analysis and trend identification. Metrics are also used to identify *if* a problem exists in the system.

Tracing follows the path of a request through various services, offering insights into end-to-end system behavior and dependencies. This lets an operator determine *where* in a complex system the problem is.

Finally, logging captures detailed event logs and system-generated messages, providing a historical record of system behavior and enabling the operators to determine *what* the problem was and to ultimately resolve it.

Benefits of adopting Observability practices in cloud native environments

Adopting observability practices in cloud native environments brings numerous benefits. Providing teams with a clear understanding of system performance, especially for troubleshooting, is job one. Especially during incident response, observability can increase mean time to detect (MTTD) and mean time to resolution (MTTR). Additionally, observability can help optimize performance across services and the end-user experience. In general, observability is critical in helping IT and engineering teams make decisions based on system data, helping to foster continuous improvement.

By collecting and analyzing data from metrics, traces and logs, and combining that with data from synthetic and real-user monitoring platforms, engineers gain valuable insights into system behavior, user interactions and performance patterns. These insights drive evidence-based decisions, enabling teams to prioritize efforts, allocate resources effectively and optimize system performance. Furthermore, by continuously monitoring and analyzing observability data, teams can identify areas for improvement, implement iterative changes and drive ongoing enhancements to the system.

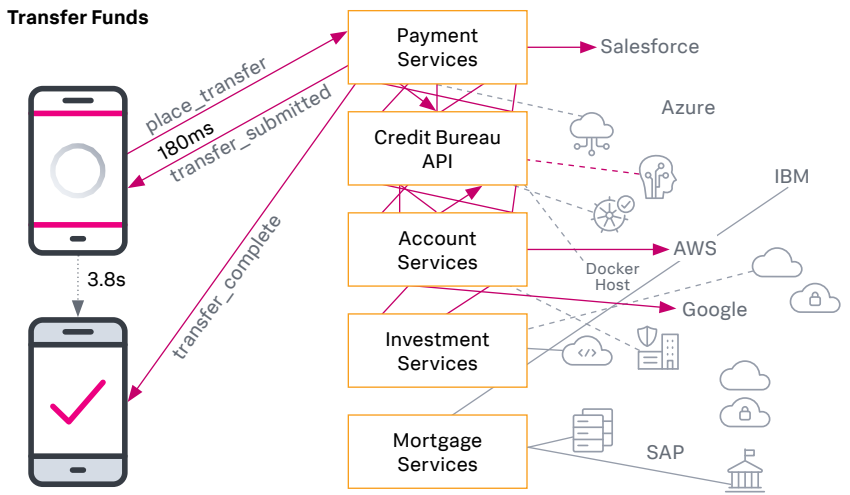


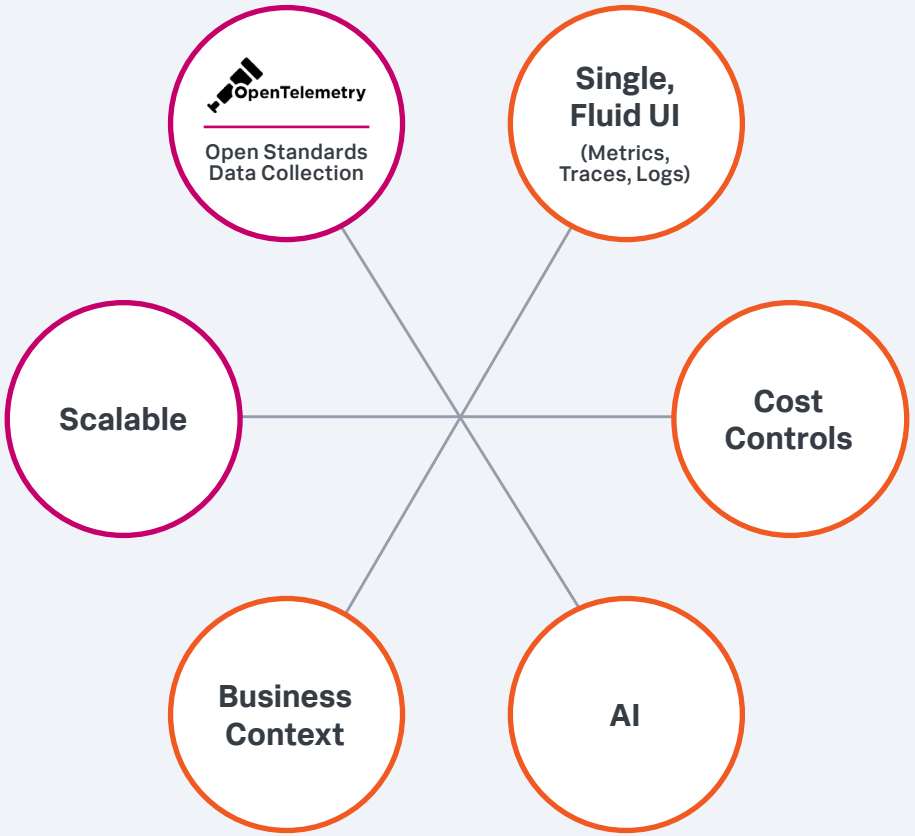
Figure 3: Example of transactions from a banking app

Implementing observability in cloud native environments

Choosing the right observability tools and frameworks

Whether an application began in the cloud and uses microservices, serverless frameworks and Kubernetes, or if its engineers are in the process of breaking down monolithic apps into smaller services, it's important to choose a solution that meets technological needs and provides additional business benefits. Some themes and questions to guide evaluation of Observability solutions include:

- **OpenTelemetry support:** Does the solution offer native support for open standards?
- **Data scale support:** Can the solution ingest the volume of telemetry associated with my entire environment and tech stack?
- **Ease of use:** Can engineers get started easily, understand how system components perform, and receive recommendations on potential root cause based on real time analysis of complete data sets?
- **Team efficiency:** Can teams troubleshoot with one single solution, in a timely manner? Are the necessary components of the system in a single UI, that integrates with their existing telemetry data and leverages AI?
- **Cost controls:** Can engineering teams measure when scaling infrastructure significantly impacts monitoring spend? Can the cost of the observability platform itself be predicted and controlled?
- **Business context:** Can engineering teams easily add tags and custom measurements to their telemetry data, and align service performance to business results? Can business results be included as alert sources?



Instrumenting applications for observability

Thorough, and hopefully pain free, instrumentation is another consideration. Modern teams are using OpenTelemetry to standardize their measurements. Any team will need to incorporate logging libraries, metrics collectors, and distributed tracing agents within their code base. This process requires a lot of work, so should only be done once. This is the main drive behind the adoption of OpenTelemetry — by using OpenTelemetry, data can be sent to any observability platform (or even to multiple platforms,) providing maximum flexibility. OpenTelemetry even supports automatic instrumentations for popular languages and frameworks, speeding this process up. Do it once, and be future-proof for the next big thing in observability.

A single, fluid UI to troubleshoot every issue with confidence

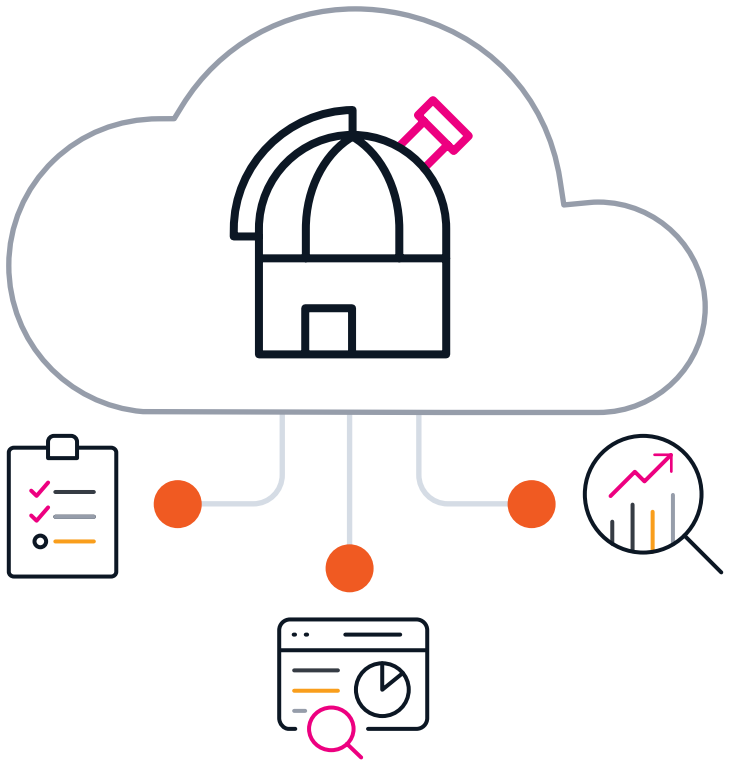
Job one for an observability solution is to provide a seamless troubleshooting experience in an easy-to-understand UI, at scale. Ideally, effective troubleshooting of cloud native environments often begins with an alert from abnormally performing metrics, an engineer uses trace data to isolate specific components common in the problem, and uses log data to pinpoint the exact problem with granularity. Therefore, Observability UI's must come with:

1. Dynamic service maps to visualize where a problem exists.
2. AI to suggest where the biggest problems or root cause exists.
3. A fluid troubleshooting experience where each click provides additional context.
4. The ability to easily add custom business metrics and tags.

Engineering teams using multiple monitoring tools must reconcile data by jumping across separate tools to reconcile data from their system.

Scaling observability to meet growth, while maintaining cost controls

Any monitoring solution will say they scale, but as applications scale to meet global demand, rough edges with monitoring or observability platforms can shine through. A well-architected observability solution will: a) scale alongside the demand from its customers (your business) without dropping valuable data, b) maintain performance at scale, continuing to update dashboards and services maps in seconds, despite demand, and c) do this in a way that has predictable costs, not enormous bills and surprise overages.



Using observability to overcome common challenges

Proactively detecting problems, and real-time issue detection

Real-time monitoring and alerting are essential components of modern observability solutions. By continuously monitoring system metrics, traces and logs in real time, organizations can proactively detect and address potential issues before they impact user experience or system performance.

Modern components such as serverless functions can have lifetimes measured in milliseconds. Waiting five minutes for alerts simply doesn't work anymore. Automated alerts and notifications enable engineering teams to respond swiftly and take remedial actions, minimizing downtime and ensuring optimal system health.

Centralized logging and log analysis for comprehensive system insights

Centralized logging plays a crucial role in finding the root cause of problems and gaining comprehensive insights into complex distributed systems. By aggregating logs from various components and services, organizations can analyze patterns, detect anomalies and troubleshoot issues more effectively.

Sophisticated log analysis tools enable powerful searching, filtering and correlation capabilities, allowing VPs of engineering and their teams to gain deeper visibility into the system's behavior and uncover valuable insights.

> "We had built a big transactional engine, but it was hard to troubleshoot these systems when something went wrong."

— Erik Swan, former CTO and co-founder of Splunk

Advanced analytics and anomaly detection for rapid problem identification

Modern observability solutions use advanced analytics and anomaly detection techniques to identify and highlight abnormal system behavior. By employing machine learning algorithms and statistical models, these solutions can detect deviations from normal patterns, identify potential issues, and alert engineering teams. This enables faster problem identification and accelerates the mean time to resolution (MTTR) and ensures adherence to service-level objectives (SLOs) by focusing efforts on critical areas that require attention.

Distributed tracing and service dependency mapping for root cause analysis

Distributed tracing provides end-to-end visibility into requests flowing through complex, distributed systems. By tracing the path of requests across services and capturing timing and contextual information, VPs of engineering can understand the dependencies and performance bottlenecks within their environment.

Service dependency mapping visualizes these relationships — allowing for better root cause analysis, performance optimization and troubleshooting of system-wide issues. As the application changes and new services are deployed and old ones are removed, a dynamic service map makes sure that issues are pinpointed quickly every time.

> "We replaced our monolith with microservices so that every outage could be more like a murder mystery."

— Corey Watson, former head of observability at Stripe

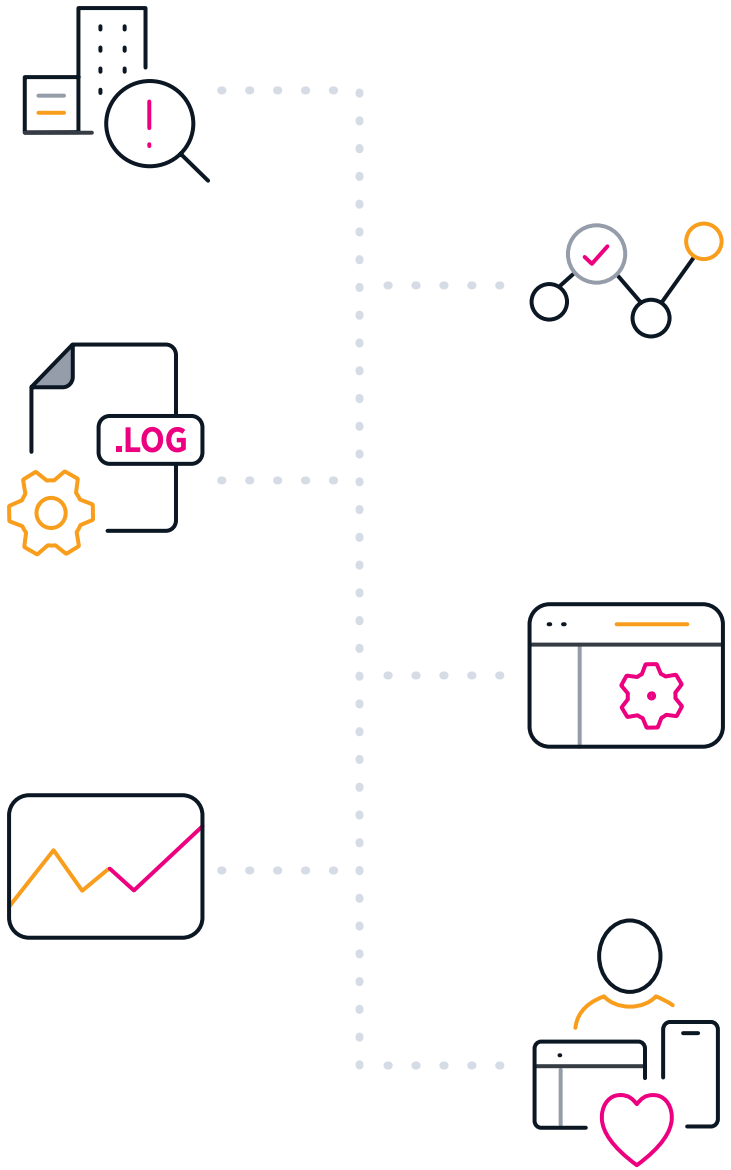
Collaborative incident response and post-incident analysis

Modern observability solutions facilitate collaborative incident response and post-incident analysis, enabling effective communication and knowledge sharing among engineering teams. With shared visibility into system metrics, traces and logs, teams can collaborate in real-time during incident response, diagnose issues collectively, and implement effective mitigation strategies. Post-incident analysis helps identify the root causes, capture lessons learned and improve overall system resilience and reliability.

Real-user and synthetic monitoring

Increasingly, organizations are taking a customer-centric approach to monitoring with digital experience monitoring, or the combination of RUM (real user monitoring) and synthetic monitoring. RUM data measures and visualizes your actual user experience, and can help engineering teams find and fix customer-facing issues.

Synthetic data, on the other hand, can help proactively test user experience across an entire journey, and proactively find problems before customers do. Both real-user and synthetic monitoring can be incorporated into metric, trace, and log data, to further resolve and prioritize end user facing issues.



Case studies:

Examining success from who's doing it right

Here are some real-world examples of how modern observability solutions have helped Splunk customers solve complex challenges in cloud native environments.

Rappi

The Latin American delivery service processes nearly nine million orders a month as it grows its business across the region. Their engineering team manages more than 1,000 microservices, 6,000 hosts and 15,000 containers. "We're all attuned to the potential business impact of downtime, so we're grateful that Splunk observability helps us be proactive about reliability and resilience with end-to-end visibility into our environment, says Jose Felipe Lopez, engineering manager for Rappi.

Dana

Dana has 135 million Indonesians using their digital payment platform. They've experienced 150% growth in annual transactions, making them one of the most popular e-wallet providers in Indonesia. Dana used legacy monitoring solutions, including homegrown tools, which resulted in blind spots across their environments. Troubleshooting performance issues also required an arduous manual process of sifting through application and infrastructure logs.

"Our old data monitoring tool, which included some legacy homegrown solutions, only gave us eyes into specific problems, without a holistic picture of the entire environment," says Norman Sasono, chief technology officer. "Splunk gives us both the depth and breadth of visibility we need, helping us reduce gaps from dropped transactions."

Remember three things:

1. **VPs of engineering face new challenges when managing cloud native environments.**

   In today's cloud native environments, engineering leadership faces numerous challenges brought about by the increasing complexity of distributed systems. From ensuring system stability to driving innovation, they must navigate a landscape that demands seamless performance, scalability and reliability. The intricate nature of these environments poses significant hurdles in terms of visibility, troubleshooting and efficient resource utilization.

2. **Observability practices are vital to ensure system security and reliability.**

   Embracing observability practices is crucial for teams operating cloud native environments. Observability offers a powerful toolkit to gain comprehensive insights into system behavior, enabling proactive detection of issues, rapid troubleshooting and informed decision making.

   Observability empowers engineering teams to understand the intricacies of their complex systems, facilitating efficient incident response and fostering a culture of continuous improvement.

3. **Observability is a journey — and you should start today.**

   As cloud native environments continue to evolve, the adoption of observability becomes increasingly essential. It is a game changer that equips engineering leadership with the tools and methodologies to tame the complexity of their distributed systems. By embracing observability, they can harness the full potential of modern engineering practices, overcome challenges, and drive innovation with confidence. It is an ongoing journey that requires continuous exploration, experimentation and collaboration among teams.

Checklist when considering observability solutions in RFPs

Checklist for Observability Solutions	Vendor A	Vendor B	Vendor C
Ability to ingest/analyze/visualize data volume at scale?			
OpenTelemetry-based Instrumentation			
Real time monitoring and alerting			
Scalability and performance			
Distributed tracing and automatic service dependency mapping			
Centralized logging and log analysis			
Advanced analytics and anomaly detection			
Collaborative incident response and post-incident analysis			
Integration and compatibility with existing systems			
Customizability for business metrics and metrics, traces, logs			
Integrated support for real-user monitoring and synthetic monitoring			

Learn More.

Solve problems in seconds with the only full-stack, analytics-powered observability solution.

Schedule Demo

splunk>

23-265351-Splunk-Ess Guide to O11y for Cloud Native Environments-EB-106