THE COMPLETE GUIDE TO

Achieving Observability in Complex Modern Applications

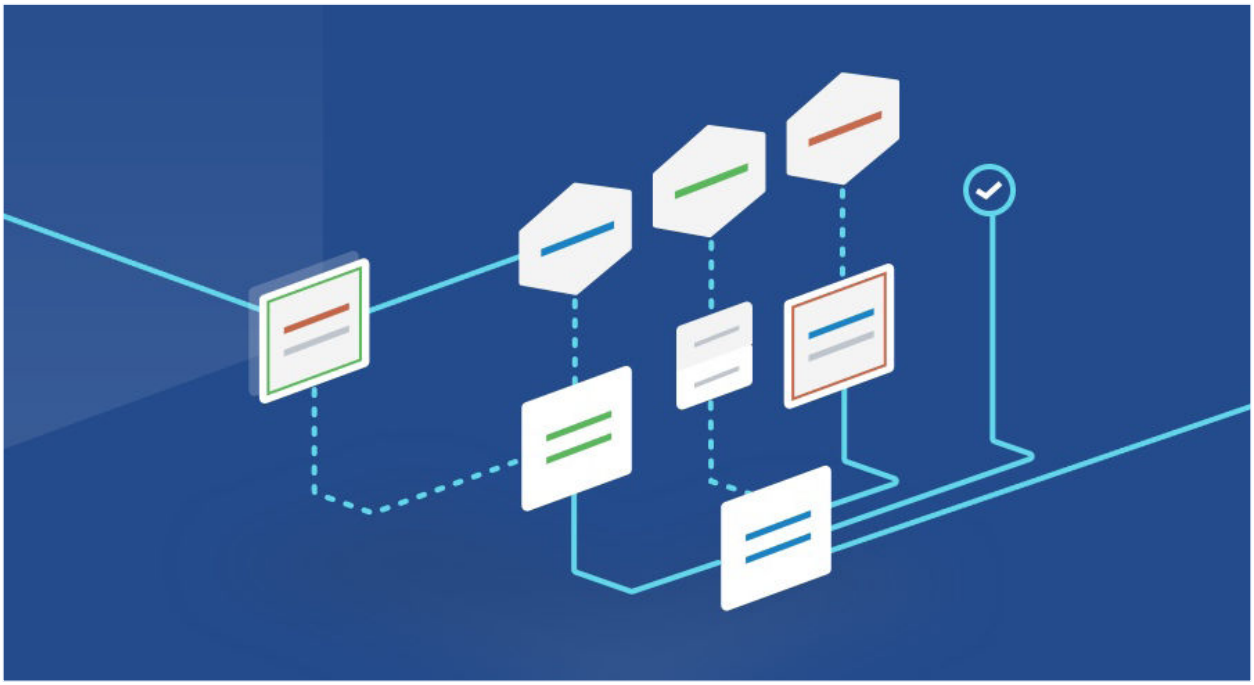
Rollbar

Table of Contents

The State of Application Architecture and Deployment

Software applications have changed radically in just a few years. Even in cases where application functionality has not changed, application architectures and deployments often look very different than they did at the start of this decade.

Modern and cloud-native applications are composed of a complex web of microservices, rather than monolithic binaries. In many cases, they are deployed using infrastructure such as REST services, containers, and serverless functions, all of which add significantly more complexity and layers to the software stack. Applications are sometimes written using multiple languages, a feature that was difficult to implement prior to the microservices age. And because of the continuous delivery paradigm, application updates arrive at dizzying speed.



From the perspective of developers and users, most of these changes are welcome. They lead to software that is more agile, to more efficient coding practices, and faster delivery of new features into production.

The Observability Challenge

However, modern application architectures and deployment processes also create challenges, particularly for DevOps teams that are responsible for managing applications across the development lifecycle.

Chief among these challenges is observability.

Observability refers to the ability to identify and interpret changes to software and infrastructure on an ongoing basis. It also entails the ability to respond to undesirable changes quickly in order to guarantee application performance without slowing down delivery.

Observability is a relatively new term within the DevOps lexicon, which originated from the study of control systems. It has been used in the aerospace industry for years since it's critical that pilots can determine and control the state of the plane even in challenging flying conditions with poor visibility.

In the software field, observability may appear to be merely a jargony way to refer to what DevOps engineers have traditionally called monitoring, but it's actually different. As Cindy Sridharan has persuasively written, observability and monitoring complement one another, but are not alternative terms for the same process.

Observability means that a system's output is sufficient to determine its state. Monitoring a system does not necessarily mean that you have all the information you need to determine the system's state. This is critically important when you're troubleshooting a problem. If your system is essentially a black box and you cannot determine its state, it becomes very difficult to troubleshoot the system. In order to achieve observability, you need to instrument your system with sufficient outputs through logs, events, or other data that can be used by engineers to troubleshoot problems.

Monitoring tools can help you to achieve observability, but they don't give the full picture. You may need to use several systems and resources, including log events, APM, and distributed tracing. Furthermore, you need to design your system in such a way that it can be observed, as well as instrument outputs that give you enough data to troubleshoot problems.

The goal of observability is to gain continuous understanding of the state of an application in order to get to the root of complex performance problems, foresee and prevent future issues, and map the complex relationships between infrastructure, microservices, source code, and software delivery processes.

Modern Observability Challenges

Maintaining observability in a modern software environment is difficult because of new trends in software architecture and deployment. The greater the complexity of infrastructure and application architectures, the harder it becomes to troubleshoot performance and reliability issues. Unless this challenge is properly managed, it leads to a loss of observability, which in turn increases **mean time to resolution (MTTR)** of performance incidents. Ultimately, poor observability degrades the user experience and undercuts the agility and velocity of the continuous delivery process. This is bad for the DevOps team, and bad for business.

Achieving observability has always required careful planning and the implementation of tools that extend beyond monitoring. However, observability is particularly challenging in today's software ecosystem for the following reasons.

- **Full-stack applications**. It is common today for applications to be composed of server-side and client-side components. In some cases, there may be multiple applications' services running in each of these locations. Observability requires understanding all parts of the stack.

- **Microservices**. Microservices make applications more agile, but they also mean that an application is comprised of many more moving parts. In addition, the dependencies between microservices are often complex, with the result that the root of a problem that affects one microservice may lie with a different microservice. Root-cause analysis is therefore especially difficult in a microservices architecture.

- **Multi-language applications**. Microservices make it easier to write an application using multiple programming languages because different microservices can be implemented in different languages. This is advantageous from a development standpoint, but it means that observability tools must support multiple languages.

- **Complex hosting architectures**. Today's applications are often not deployed in a single location. They may span multiple public clouds, or rely on a hybrid cloud deployment strategy that mixes public and private infrastructure. These complex deployment infrastructures mean that there are more deployment regions to observe. What's more, each region may require different observability tools or processes.

- **Continuous delivery**. Continuous delivery has many benefits, but it leads to rapidly changing software configurations. Observability processes must keep pace, which means that continuous observability is the only effective way to manage a continuously delivered application.

- **Many-layered stacks**. The infrastructure stacks that deploy applications often have many layers. For example, an application may run inside containers, which run inside virtual machines, which run on top of host servers in a public cloud. Each of these layers needs to be observed in order to achieve observability.

- **Legacy applications**. Not all applications have modern architectures or deployment processes. Complete observability requires supporting legacy applications as well, adding more complexity to the process.

Why Monitoring Falls Short of Observability

The typical organization attempts to respond to the challenges described above with a strategy that centers on monitoring. Such an approach falls short of delivering the business value that only observability can bring.

That is because, as noted above, monitoring and observability are not the same thing. Monitoring typically involves tools that deliver the following functionality:

- **Application Performance Monitoring (APM)**. APM entails identifying performance slowdowns or failures within an application, then addressing them. APM typically focuses on runtime rather than earlier stages of the application lifecycle. APM helps to improve the performance of running applications, but it does not address deeper issues, such as coding or architectural problems that cause an application to perform suboptimally.

- **Infrastructure monitoring**. By monitoring infrastructure, you identify hardware and software failures that can lead to performance or availability problems for an application. Infrastructure monitoring helps to keep applications and data available to users, but it does little more than that.

- **Incident management**. In some cases, monitoring tool sets provide incident management functionality, which helps DevOps teams coordinate responses to application or infrastructure problems. Incident management is helpful from an organizational perspective, but its primary purpose is to facilitate responses to issues, not provide deeper visibility into applications and software environments.

In short, monitoring involves finding and responding to problems within host infrastructure or applications when they are running. This is valuable, but it does not help to manage the broader set of performance issues, process inefficiencies and user-experience problems that can occur throughout the software delivery lifecycle.

To address the latter challenges, organizations need observability. Observability provides an understanding of the application at all stages of delivery, as well as continuous visibility into the relationship between the application, the software environment that hosts it, and the infrastructure it runs on.

In these ways, a fully observable system empowers DevOps teams to build more efficient applications, maximize performance and optimize decisions about infrastructure and process design across the organization. The ultimate result is a better experience for users and greater value to the business (which spends less time and money to deliver quality software).

Achieving Effective Observability

The observability challenges discussed above can be met, but only with proper preparation and investment. Effective observability for modern applications demands a comprehensive set of tools and software development practices.

Full-Stack Error Tracking

Error tracking enables DevOps teams to trace an application performance problem or crash back to its source and identify which parts of the application source code should be updated in order to fix the issue.

Error tracking tools can be used to enhance monitoring and incident management by providing deep visibility into an application, which runtime monitoring alone cannot achieve. For example, error tracking solutions provide insight into request parameters, local variables, and telemetry on user behavior before the error occurred. In addition, error tracking can be used at earlier stages of the delivery pipeline to vet applications before they are released into production.

Leading error tracking vendors: Rollbar, Crashlytics

Distributed Tracing

Distributed tracing tools track a transaction across the multiple services and infrastructure layers through which it passes in order to reach its destination. Distributed tracing is particularly important in microservices-based, full-stack application deployments because such deployments are complex.

Distributed tracing can help to identify which component of an application environment is causing a performance problem or failure. Like error tracking, it provides a deeper level of visibility that complements monitoring data and is useful when the root cause of a problem is not apparent.

Leading distributed tracing solutions: HTrace, Zipkin

APM

As noted above, APM tools monitor applications for failures or slowdowns. They are typically used in production environments, although it is possible to deploy APM tools earlier in the development lifecycle as well.

APM tools should be used to identify application problems that could impact users. On their own, however, APM tools often do not provide enough information to identify the root cause of a performance issue, especially in modern, complex application environments.

Leading APM solutions: New Relic, Instana

Infrastructure Monitoring

Infrastructure monitoring refers to the process of checking for failures in hardware or software infrastructure that could degrade application performance. Infrastructure failures could range from a broken hard disk to a virtual machine that has stopped responding.

Like APM, infrastructure monitoring is useful for maintaining service levels in a production environment, as well as for preventing problems within testing and staging infrastructure that could slow software delivery.

Leading infrastructure monitoring solutions: Nagios, Zabbix

Log Aggregation and Analysis

Log aggregation tools collect logs from various sources and enable analysis from a centralized location. They are useful because modern software environments and applications produce a variety of different logs—ranging from authentication and access to error and operational logs— and log data is typically stored in a variety of locations.

Log aggregation and analysis are particularly useful for identifying overarching trends that span across an application environment, as well as performing post-mortems after a failure. Real-time log analysis can also help to detect security threats and other problems, although log analysis on its own is not enough to keep software environments stable in real time.

Leading log aggregation tools: Loggly, Sumo Logic

Incident Management

Incident management systems help engineers to coordinate responses to performance and availability issues. They are designed mainly to orchestrate communication and the sharing of resources.

Incident management systems do not generate the data that DevOps teams need to respond to problems; they simply help to organize it and make the right data available to the right people when an incident occurs.

Leading incident management tools: PagerDuty, VictorOps

Best Practices for Achieving Observability

In order to leverage observability tools effectively, organizations should integrate the following practices into their software delivery processes:

- **Good system architecture and development practices**. It's critical that your system capture enough data to troubleshoot problems. Monitoring solutions only track what they can see. Logging solutions require you to add meaningful log statements to your code. Bugs should not surface to users without also tracking an error or warning on the backend.

- **Balance performance with visibility**. Swallowing exceptions may prevent a crash and improve application performance from the user's perspective, but doing so reduces observability unless you log or track the error. Additionally, it's great to have automatic retries and failover in your microservices architecture, but you should track when they happen in order to troubleshoot spikes in latency and dead letter queues. All of these examples require your team to design your architecture and your software in a way that enables it to be observed.

- **Shift-left processes**. The "shift-left" paradigm refers to the practice of performing tasks earlier in the delivery chain, rather than waiting until software is in production. By shifting processes such as error tracking and APM to the left (while still performing them in production, too), organizations gain earlier insights into software issues that may impact

the user experience. Earlier insights lead to more effective observability by enabling teams to detect and respond to problems while still in integration or development environments. It's more cost-effective to track debug-level information in these environments, making problems easier to address.

- **Continuous communication**. To the extent feasible, everyone on the DevOps team should understand the tools that are used to achieve observability, and should communicate constantly about observability insights provided by these tools. Not every team member needs to be an expert in every tool; that would not be realistic. However, everyone on the team should understand the basics of the observability toolset and its goals, and information should not be siloed or handed off manually from one part of the team to another. Otherwise, your system is effectively non-observable to people tasked with running it.

- **Prioritization policies**. A fully observable system will, by design, generate more data, so you need a way to prioritize what matters from a business or user-experience perspective. Not all performance problems or application errors are of equal seriousness, and in most cases they cannot all be addressed. In order to avoid overwhelming engineers with a never-ending stream of alerts, organizations should have policies in place that help to define which types of problems receive priority. You also need support from your monitoring tools to help you rank problems according to priority.

Ultimately, achieving observability requires cultural change. This is one of the differentiators between observability and monitoring: Whereas monitoring can be achieved by adopting the right tools, achieving observability entails rethinking the ways in which your team interacts with the data its tools collect, reacts to problems and shares information.

In a culture centered on observability, engineers don't just collect data for data collection's sake. Nor do they design applications to maximize uptime and performance at the expense of visibility and continuous improvement. Instead, they strike a balance between these various priorities in order to achieve observability, which empowers them to deliver the best overall user experience.

Conclusion

Applications have evolved. So have the tools and processes that DevOps teams need to achieve observability into their applications. If your organization is performing only monitoring, and lacks the tools and processes necessary for effective observability, your software delivery chain is not driving as much value for the business as it could.

With observability, you gain the insights necessary not just to identify and respond to problems after your applications have been deployed, but to also trace issues to their source and enhance your software so that problems do not recur.

Monitoring tools also empower your organization to take full advantage of next-generation infrastructure technologies and software architecture techniques while keeping your team focused on developing a great product.

Ultimately, an observable system enables your DevOps team to provide the best possible user experience. Happy users are what generate business and drive long-term success.

Ready to get started with Rollbar?

Visit our website for more tips on using Rollbar to accelerate the transition to achieving observability in your organization, or if you are interested in learning more about the product.

We also offer a fully featured free trial if you wanted to try it hands-on and get started right away.

Should you have any questions or feedback, please contact our team at sales@rollbar.com

“At 2 AM, when I get a notification from Rollbar, I am grateful to have this much visibility into our app so we can quickly address the issue before it impacts customers.”

Ian Chan
Director of Engineering at Branch

“Rollbar allows us to go from alerting to impact analysis and resolution in a matter of minutes. Without it, we would be flying blind.”

Arnaud Ferreri
Engineering Team Lead, Instacart

“There are so many emotional pains developers and operators have from all kinds of hideous errors they have shipped. What if you could make that go away? That’s what Rollbar does.”

Rob Zuber
CTO of CircleCI