Observability is the ability to measure the internal states of a system by examining its outputs. A system is considered "observable" if the current state can be estimated by only using information from outputs, namely sensor data. While it might seem like a recent buzzword, the term originated decades ago with control theory (which is about describing and understanding self-regulating systems). However, it has increasingly been applied to improving the performance of distributed IT systems. In this context, observability uses three types of telemetry data — metrics, logs and traces — to provide deep visibility into distributed systems and allow teams to get to the root cause of a multitude of issues and improve the system's performance.

Over the last several years, enterprises have rapidly adopted cloud-native infrastructure services, such as AWS, in the form of microservice, serverless and container technologies. Tracing an event to its origin in these distributed systems requires thousands of processes running on the cloud, on-premise or both. But conventional monitoring techniques and tools struggle to track the many communication pathways and interdependencies in these distributed architectures.

Observability allows teams to monitor modern systems more effectively and helps them to find and connect effects in a complex chain and trace them back to their cause. Further, it gives system administrators, IT operations analysts and developers visibility into their entire architecture.

In this article, we'll take a closer look at observability: what it is, what it takes to implement and the benefits you can expect your organization to gain from it.

What is the difference between monitoring and observability?

Monitoring and observability are distinct concepts that depend on each other. Monitoring is an action you perform to increase the observability of your system. Observability is a property of that system, like functionality or testability.

Specifically, monitoring is the act of observing a system's performance over time. Monitoring tools collect and analyze system data and translate it into actionable insights. Fundamentally, monitoring technologies, such as application performance monitoring (APM), can tell you if a system is up or down or if there is a problem with application performance. Monitoring data aggregation and correlation can also help you to make larger inferences about the system. Load time, for example, can tell developers something about the user experience of a website or an app.

Observability, on the other hand, is a measure of how well the system’s internal states can be inferred from knowledge of its external outputs. It uses the data and insights that monitoring produces to provide a holistic understanding of your system, including its health and performance. The observability of your system, then, depends partly on how well your monitoring metrics can interpret your system's performance indicators.

Another important difference is that monitoring requires you to know what’s important to monitor in advance. Observability lets you determine what’s important by watching how the system performs over time and asking relevant questions about it.

Why is observability important?

Observability is important because it gives you greater control over complex systems. Simple systems have fewer moving parts, making them easier to manage. Monitoring CPU, memory, databases and networking conditions is usually enough to understand these systems and apply the appropriate fix to a problem.

Distributed systems have a far higher number of interconnected parts, so the number and types of failure that can occur is higher too. Additionally, distributed systems are constantly updated, and every change can create a new type of failure. In a distributed environment, understanding a current problem is an enormous challenge, largely because it produces more “unknown unknowns” than simpler systems. Because monitoring requires “known unknowns,” it often fails to adequately address problems in these complex environments.

Observability is better suited for the unpredictability of distributed systems, mainly because it allows you to ask questions about your system's behavior as issues arise. “Why is X broken?” or “What is causing latency right now?” are a few of the questions that observability can answer.

What is observability in containers and microservices?

Observability in containers and microservices exposes the state of applications in production so developers can better identify and resolve performance issues.

Container services (such as Docker, Kubernetes and others) and microservices address the increased risk of downtime and other issues related to monolithic software, in which any change to the single codebase affects the entire application and its dependencies. Containers and microservices break applications down into independent services, allowing developers to modify and redeploy a particular service rather than the whole application.

However, a container-based architecture introduces new challenges. Interdependent microservices are typically scattered across multiple hosts, and as the infrastructure scales, so does the number of microservices in production. This makes it difficult for developers to know what's currently running in production, leading to longer delivery cycles, downtime and other issues.

Observability addresses these challenges, providing visibility into distributed systems that help developers better understand an app's performance and availability. In the event of a failure, it provides the control needed to pinpoint and debug or fix the problem quickly.

What are the primary data classes used in observability and how are they used?

The primary data classes used in observability are logs, metrics and traces. Together they are often called "the three pillars of observability."

Logs: A log is a text record of an event that happened at a particular time and includes a timestamp that tells when it occurred and a payload that provides context. Logs come in three formats: plain text, structured and binary. Plain text is the most common, but structured logs — which include additional data and metadata and are easier to query — are becoming increasingly popular. Logs are also typically the first place you look when something goes wrong in a system.

Metrics: A metric is a numeric value measured over an interval of time and includes specific attributes such as timestamp, name, KPIs and value. Unlike logs, metrics are structured by default, which makes it easier to query and optimize for storage, giving you the ability to retain them for longer periods.

Traces: A trace represents the end-to-end journey of a request through a distributed system. As a request moves through the host system, every operation performed on it — called a "span" — is encoded with important data relating to the microservice performing that operation. By viewing traces, each of which includes one or more spans, you can track its course through a distributed system and identify the cause of a bottleneck or breakdown.

Working with these data classes doesn't guarantee observability, particularly if you're working with them independently of each other or are using different tools for each function. Rather, you'll achieve a successful approach to observability by integrating your logs, metrics and traces within a single solution. When you do this, you not only understand when problems occur, you can immediately shift the focus to understanding why those problems are occurring.

How do I implement observability?

To achieve observability you need proper tooling of your systems and apps to collect the appropriate telemetry data. You can make an observable system by building your own tools, using open source software or buying a commercial observability solution. Typically, there are four components involved in implementing observability:

- **Instrumentation:** These are measuring tools that collect telemetry data from a container, service, application, host and any other component of your system, enabling visibility across your entire infrastructure.
- **Data correlation:** The telemetry data collected from across your system is processed and correlated, which creates context and enables automated or custom data curation for time series visualizations.
- **Incident response:** These automation technologies are intended to get data about outages to the right people and teams based on on-call schedules and technical skills.
- **AIOps:** Machine learning models are used to automatically aggregate, correlate and prioritize incident data, allowing you to filter out alert noise, detect issues that can impact the system and accelerate incident response when they do.

What are the criteria for good observability tools?

Regardless of whether you choose to build your own or use open source or commercial solutions, all observability tools should:

Integrate with current tools: If your observability tools don't work with your current stack, your observability efforts will fail. Make sure they support the frameworks and languages in your environment, container platform, messaging platform and any other critical software.

Be user-friendly: If your observability tools are hard to learn or use, they won't get added to workflows — preventing your observability initiative from getting off the ground.

Supply real-time data: Your observability tools should provide the relevant insights via dashboards, reports and queries in real time so teams can understand an issue, its impact and how to resolve it.

Support modern event-handling techniques: Effective observability tools should be able to collect all relevant information from across your stacks, technologies, and operating environments; separate valuable signals from the noise, and add enough context so that teams can address it.

Visualize aggregated data: Observability tools should surface insights in easily digestible formats, such as dashboards, interactive summaries and other visualizations that users can comprehend quickly.

Provide context: When an incident arises, your tools should provide enough context for you to understand how your system's performance has changed over time, how the change relates to other changes in the system, the scope of the issue and any interdependencies of the affected service or component. Without context at the level that observability can provide, incident response is crippled.

Use machine learning: Your tools should include machine learning models that automate data processing and curation, so you can detect and respond to anomalies and other security incidents faster.

Deliver business value: Make sure you're evaluating your observability tool against metrics important to your business, like deployment speed, system stability and customer experience.

What are the benefits of observability in DevOps?

Observability allows DevOps developers to understand an application's internal state at any given time and have access to more accurate information about system faults in distributed production environments. A few key benefits include:

Better visibility: Sprawling distributed systems often make it hard for developers to know what services are in production, whether application performance is strong, who owns a certain service or what the system looked like before the most recent deployment. Observability gives them real-time visibility into production systems that can help remove these impediments.

Better alerting: Observability helps developers discover and fix problems faster, providing deeper visibility that allows them to quickly determine what has changed in the system, debug or fix the issues and determine what, if any, problems those changes have caused.

Better workflow: Observability allows developers to see a request's end-to-end journey, along with relevant contextualized data about a particular issue, which in turn streamlines the investigation and debugging process for an application,optimizing its performance.

Less time in meetings: Historically, developers would have to track down information through third-party companies and apps to find out who was responsible for a particular service or what the system looked like days or weeks before the most-recent deployment.

With effective observability, this information is readily available.

Accelerated developer velocity: Observability makes monitoring and troubleshooting more efficient, removing the main friction point for developers. The result is increased speed of delivery and more time for DevOps staff to come up with innovative ideas to meet the needs of the business and its customers.

Benefits of Obervability

What are the benefits of observability in software engineering?

As with DevOps, observability benefits software engineers by providing insights into the entire infrastructure, allowing them to see how it changes because of a problem, as new software is deployed, or as it is scaled up or down.

Who benefits from observability?

Individual developers and software engineers benefit from observability because of the visibility it provides into their entire architecture, from third-party apps and services to their own. This not only enables them to more easily fix and eventually prevent problems, it also fosters a greater understanding of system performance and how it shapes a better customer experience. Both developers and engineers then have more time for strategic initiatives that benefit the business.

Teams also benefit because observability offers a shared view of the environment, providing a more comprehensive understanding of its architecture, health and performance over time. Observability allows developers, operators, engineers, analysts, project managers and other team members to access the same insights about services, customers and other system elements. Also, observability creates more accurate post-incident reviews, as all parties can examine documented records of real-time system behavior instead of piecing events together from siloed, individual sources. Data — not opinions — will help your teams understand why incidents occurred so they can better prevent and handle future incidents.

The business, however, might benefit the most. Observability allows you to make changes to your apps and services without compromising the stability of your systems by giving you the tools to understand what’s working and what’s not, pinpoint any issues that crop up and quickly improve or resolve them. New features combined with less downtime translate to happier customers and a more robust bottom line.

The Bottom Line: Get insight into your infrastructure

Observability is more than just a buzzword — it's an important and useful approach to understanding the state of your entire infrastructure. The cloud, containerization, microservices and other technologies have made systems more complex than they've ever been. While the net result of these tools is positive, working within, troubleshooting and managing these systems is fraught with difficulties. More interactive parts lead to a greater variety of problems, which, when they occur, are harder to detect and fix.

Fortunately, these distributed systems produce a wealth of telemetry data that provide a clearer understanding of their performance, if you can harness it. Effective observability tools provide all the instrumentation and analytic horsepower you need to capture and contextualize your system's output and deliver the insights required to thrive in the world of modern distributed systems.

More resources

- DFTT #1. Observability for the real world
- DFTT #3: What Does Modern Observability Need?
- If Observability is Everything, Where Are You?
- In Observability, RED is the New Black
- Using Observability as a Proxy for Customer Happiness
- DFTT #4: What Culture of Observability?
- Does Observability Throw You for a Loop? Part One: Open with Observability
- Does Observability Throw You for a Loop? Part Two: Close with Controllability
- DFTT #6: What are the Myths of Observability?