What Is OpenTelemetry?

OpenTelemetry is an open source observability framework. It offers vendor-agnostic or vendor-neutral APIs, software development kits (SDKs) and other tools for collecting telemetry data from cloud-native applications and their supporting infrastructure to understand their performance and health.

Managing performance in today's complex, distributed environment is extremely difficult. Telemetry data is critical for helping DevOps and IT groups understand these systems' behavior and performance. To gain a complete picture of their services' and applications' behavior, they need to instrument all their frameworks and libraries across programming languages.

However, no commercial vendor has a single instrument or tool to collect data from all of an organization's applications. This lack results in data silos and other ambiguities that make troubleshooting and performance-issue resolution more difficult.

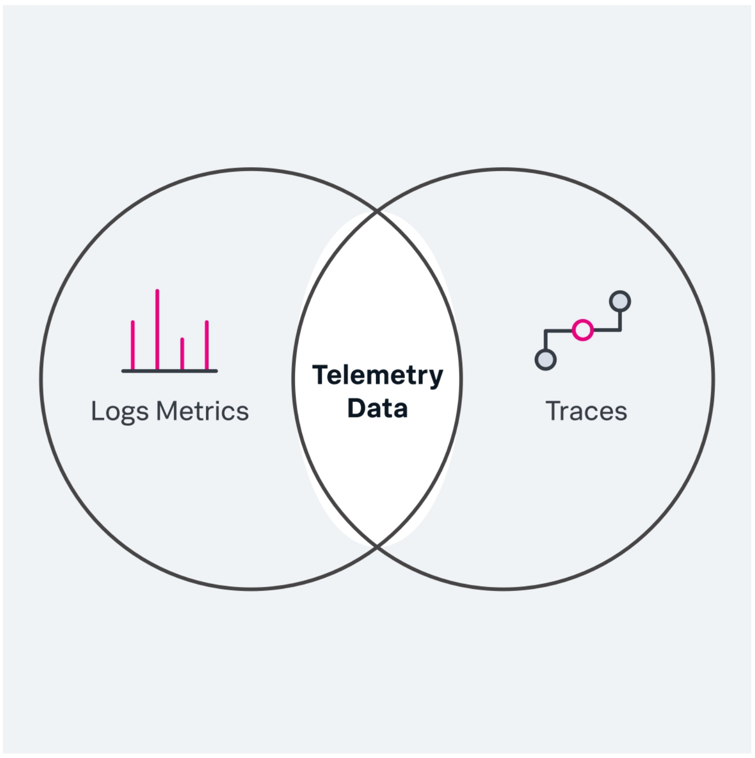
OpenTelemetry is important because it standardizes the way telemetry data is collected and transmitted to backend platforms. It bridges visibility gaps by providing a common format of instrumentation across all services. Engineers don't have to re-instrument code or install different proprietary agents every time a backend platform is changed. OpenTelemetry will continue to work, too, as new technologies emerge, unlike commercial solutions, which will require vendors to build new integrations to make their products interoperable.

In the sections that follow, we'll take a closer look at OpenTelemetry, how it works and how it helps organizations achieve the observability in their distributed systems needed to meet their business goals.

What is telemetry data?

Telemetry data is the output collected from system sources in observability. Analyzed together, this output provides a view of the relationships and dependencies within a distributed system. There are three primary data classes used, often called "the three pillars of observability": logs, metrics and traces.

Logs: A log is a text record of an event that happened at a particular time. Whenever a block of code gets executed, a log entry is produced, recording the time the event occurred and providing a “payload” of context about that event. Log data comes in three formats: plain text, structured and unstructured. Plain text is the most common, but structured logs — which include additional metadata and are more easily queried — are becoming increasingly popular. Conversely, unstructured logs are harder to parse. Logs are typically the source of truth for what the application is doing. It's where you look when something goes wrong in a system, and developers rely on them to troubleshoot their code and verify its execution. A failure in a distributed system usually has a series of underlying causes (sometimes called root causes), and logging provides us with fine-grain details when certain code blocks get executed.
Metrics: A metric is a numeric value measured over an interval of time. It includes specific attributes such as timestamp, name of an event and the value of that event. Unlike logs, metrics are structured by default, which makes them easier to query. It also optimizes them for storage, which lets you retain more metrics for longer periods, giving a better perspective of a system’s historical trends. Metrics work best for large data sets or data collected at regular intervals when you want an answer to a specific question. We tend to see metrics aggregated over time, which in our modern systems is crucial to allow us to analyze and respond to issues. Metrics power alerts, either in aggregate or as a single point, which is often the first indicator of problems in our systems.
Traces: A trace represents the end-to-end journey of a request through a distributed system. Many operations are performed on a single request as it moves through a system. Each operation gets encoded with important data — called a “span” — that includes trace identifier, span identifier, the name of the operation, a start and end timestamp, logs, events and other indexed information. By viewing a trace, known as distributed tracing, you can track a complete execution path and identify which part of the code is causing issues like errors, latency concerns or resource availability. Traces also can generate metrics, particularly in a form known as RED (rate, errors and duration). Critically, traces also provide context for determining which instances of the other two pillars are most relevant for troubleshooting a particular issue.



Individually, logs, metrics and traces serve different purposes, but together they provide the comprehensive detailed insights needed to understand and troubleshoot distributed systems.

How is OpenTelemetry used, and why is it important to DevOps?

OpenTelemetry is used to collect telemetry data from distributed systems in order to troubleshoot, debug and manage applications and their host environment. It offers an easy way for IT and developer teams to instrument their code base for data collection and make adjustments as the organization grows.

OpenTelemetry collects several classes of telemetry data, including logs, metrics and traces, and exports them to backend platforms for processing. Analysis of this telemetry data allows teams to bring coherence to multi-layered ecosystems. This makes it far easier to observe the systems' behavior and address any performance issues.

The OpenTelemetry framework includes several components:

Language-specific OpenTelemetry APIs — Java, Python, JavaScript and more — to instrument code for data collection.
Exporters to enable the transmission of telemetry data to the backend observability platform of your choice.
Language-specific OpenTelemetry SDKs to provide a bridge between the APIs and the exporters.
The OpenTelemetry Collector to provide a vendor-agnostic implementation for receiving, processing and exporting telemetry data.

OpenTelemetry is important to DevOps because the data it provides simplifies alerting, troubleshooting and debugging applications. While telemetry data has always been used to understand system behavior, general network complexity has made collecting and analyzing tracing data more difficult. Tracking the cause of a single incident using conventional methods in these labyrinthine systems can take hours or days.

But OpenTelemetry improves observability in these systems by bringing together traces, logs and metrics from across applications and services in a correlated manner. Further, the open source project removes roadblocks to instrumentation so organizations can get down to the business of vital functions such as application performance monitoring (APM) and others. The net result is greater efficiency in identifying and resolving incidents, better service reliability and reduced downtime.

What are the benefits of OpenTelemetry?

OpenTelemetry offers several benefits — here are the top three.

Consistency: The practice of collecting telemetry data from applications existed before OpenTelemetry, but it was a lot more difficult. Finding the right collection of instrumentation was challenging, and once you landed on a particular solution or vendor you were contract-bound to them. That solution was unlikely to be consistent from one application to another, making it nearly impossible to get a holistic understanding of an application’s performance.

OpenTelemetry, on the other hand, provides a consistent path for capturing telemetry data and transmitting it to a backend without changing instrumentation, offering a de facto standard for adding observability to cloud-native apps. Developers and IT can now devote more time to creating new app features rather than wrestling with their instrumentation.

Simpler choice: Before OpenTelemetry, organizations had to choose between OpenTracing or OpenCensus, each of which offered a different approach to achieving observability. Because OpenTelemetry merges the code of those two frameworks, you get the best of both in a single solution. And there’s no risk from switching to OpenTelemetry if you were previously using one or the other. OpenTelemetry is backward compatible with both.

Streamlined observability: OpenTelemetry lets developers view application usage and performance data from any device or web browser. This convenient interface makes it easy to track and analyze observability data in real time.

Of course, the greatest benefit is gaining the observability necessary to achieve business goals. OpenTelemetry consolidates the telemetry data needed to determine if systems are functioning properly, understand where issues may be compromising performance and fix root causes, potentially before service is interrupted — resulting in greater stability and reliability for supporting business processes.

How does OpenTelemetry relate to AI?

OpenTelemetry can feed collected data into an AI engine to automatically produce actionable insights. As an example, the AI engine can continuously analyze data that OpenTelemetry captures along with other useful and desirable data and look for anomalies throughout the full stack without any human intervention. The AI can reveal and process billions of dependencies within a distributed system in fractions of a second and detect hidden activity. It can also automatically identify the source of issues and, when possible or desirable, fix them before they impact end users. Through this continuous process, the AI will learn what the “normal” state of the system is and adapt its responses to improve performance over time, including the potential for predicting problems before they arise.

Essentially, AI integration increases the value of OpenTelemetry by reducing the manual effort behind using observability to analyze conditions and making it easier to get actionable insights.

What are the origins of OpenTelemetry?

OpenTelemetry is the combination of two overlapping open source distributed tracing projects, OpenTracing and OpenCensus, merged into a single project.

OpenTracing, hosted by the Cloud Native Computing Foundation (CNCF), was an effort to provide a standardized API for tracing. Among other things, it let developers embed instrumentation in commonly used libraries or their own custom code without getting locked into a particular vendor. It was a unique concept at the time of its implementation. Though OpenTracing gave developers much-needed flexibility, it had limited applications and inconsistent implementations because it focused solely on tracing.

Google developed OpenCensus based on its internal tracing platform. It was eventually open-sourced, and Microsoft, along with other vendors and contributors, got involved and began evolving the standard. OpenCensus was a set of multi-language libraries that collected metrics about application behavior and could transfer that data to the backend analysis platform of the developer's choice to help with debugging. It could also trace messages, requests and services from their source to their destination. However, there was no API available to embed OpenCensus into code, so developers had to use community-built automatic instrumentation agents for the task.

OpenTelemetry came about when the two projects agreed to merge the codebases of the OpenTracing and OpenCensus projects, incorporating the strengths of each under CNCF control. Currently in beta, OpenTelemetry offers one set of APIs, libraries, agents and collector services for capturing distributed traces and metrics (and soon logs) from any application, then analyzing them using popular observability tools.

What is OpenTracing?

OpenTracing are vendor-agnostic API specifications, frameworks and libraries (not owned by any specific company) that help developers easily instrument tracing into their code base. OpenTracing simplifies the distributed tracing process by providing a standard mechanism developers can use to instrument their own code without being locked into a particular tracing vendor.

Distributed tracing, also called distributed request tracing, is a method for monitoring applications built on microservices architecture. Programmers use tracing, along with other forms of logging, to collect information about an application's behavior. Developers use this information for debugging, and system administrators, tech support and others use it to troubleshoot apps and services.

What is observability?

Observability is the practice of measuring the state of a system by its outputs. It is a term that originated in control theory, which is concerned with describing and understanding how self-regulating systems operate. Increasingly, organizations are adding observability to distributed IT systems to understand and improve their performance. In this context, observability uses telemetry data to gain deep visibility into these systems and enable teams to answer a multitude of questions about the systems' behavior.

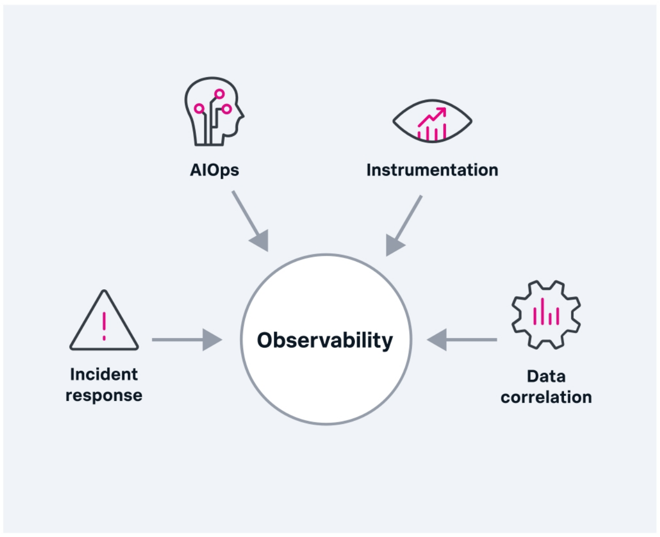
Managing distributed systems is challenging because of their high number of interdependent parts connected via loosely coupled communication paths. This increases the number and types of failures they're subjected to. It also makes it hard to understand problems in a distributed system's current state, let alone predict future issues. Essentially, distributed systems produce more "unknown unknowns" than simpler systems. Because conventional monitoring is predicated on "known unknowns," it struggles to identify problems in these complicated environments.

Observability is better suited for this unpredictability. It allows for greater control over these complex modern systems and makes their behavior easier to understand. Teams can more easily identify broken links in a complex environment and trace them back to their cause. Observability also allows developers to take an exploratory approach to system failures by asking questions like "Why is X broken?" or "What is causing latency right now?"

To achieve observability, you have to prepare your system and apps to collect the appropriate telemetry data. You can purchase a commercial observability platform, use an open source solution or build your own. In any case, there are four components to observability:

Instrumentation: Measuring tools collect telemetry data from a CPU, container, service, application and any other component of your system that produces data.

This provides and maintains visibility across your entire infrastructure as you add new components and data types. Auto-instrumentation is crucial to rapid implementation.
Data correlation: The telemetry data collected from your system is processed and correlated. This creates context and enables automated or custom data curation for the observability tool's visualizations.
Incident response: Automated technologies forward data about outages to the relevant people and teams.
AIOps: Machine learning automatically aggregates, correlates and prioritizes incident data. This filters out alert noise, letting teams detect issues and accelerate incident response..



How does OpenTelemetry compare to observability?

OpenTelemetry is a means of achieving observability. Organizations need to add instrumentation across their infrastructure to gain end-to-end visibility. This is challenging to do with a single commercial solution, but OpenTelemetry standardizes the instrumentation needed to collect the telemetry data and enables observability.

What is the future of OpenTelemetry?

The OpenTelemetry project is still in its early stages. It's currently in beta and only supports traces and metrics; log support is in the initial planning stages. Project teams are working on stabilizing OpenTelemetry's core components, integrating automated instrumentation through agents, adding language compatibility and improving metrics capabilities. From there, we should see the general availability of a production-quality release. OpenTelemetry is one of CNCF's biggest open source projects (in addition to Prometheus and Kubernetes), and is receiving considerable support from the technology community. OpenTelemetry will eventually be the dominant observability framework in the cloud-native telemetry landscape.

What are different types of distributed deployments?

Distributed deployments can range from tiny, single department deployments on local area networks to large-scale, global deployments. In addition to their size and overall complexity, organizations can consider deployments based on the size and capacity of their computer network, the amount of data they'll consume, how frequently they run processes, whether they'll be scheduled or ad hoc, the number of users accessing the system, capacity of their data center and the necessary data fidelity and availability requirements.

Based on these considerations, distributed deployments are categorized as departmental, small enterprise, medium enterprise or large enterprise. While there are no official taxonomies delineating what separates a medium enterprise from a large enterprise, these categories represent a starting point for planning the needed resources to implement a distributed computing system. Distributed systems can also evolve over time, transitioning from departmental to small enterprise as the enterprise grows and expands.

Why do we need distributed systems now?

Modern computing wouldn't be possible without distributed systems. They're essential to the operations of wireless networks, cloud computing services and the internet. If distributed systems didn't exist, neither would any of these technologies.

But do we still need distributed systems for enterprise-level jobs that don't have the complexity of an entire telecommunications network? In most cases, the answer is yes. Distributed systems provide scalability and improved performance in ways that monolithic systems can't, and because they can draw on the capabilities of other computing devices and processes, distributed systems can offer features that would be difficult or impossible to develop on a single system.

This includes things like performing an off-site server and application backup — if the master catalog doesn’t see the segment bits it needs for a restore, it can ask the other off-site node or nodes to send the segments. But ultimately, virtually everything you do now with a computing device takes advantage of the power of distributed systems, whether that’s sending an email, playing a game or reading this article on the web.

The Bottom Line: Digital interconnectedness makes OpenTelemetry essential

Over the last two decades, digital transformation has created new ways of working, and mobile and web apps are more important than ever.

Data collection is a critical driver of these cloud-native applications, but many organizations aren’t equipped for it. Fifty-seven percent of organizations across industries say that the volume of data is growing faster than their ability to keep up with it, and 47% say their organization will fall behind this rapid growth of data volume. And while a majority believe data is essential for overall success and innovation, a full two-thirds say that half or more of their data is dark — a 10% increase over the previous year.

OpenTelemetry is the key to getting a handle on your telemetry and fueling the comprehensive visibility you need to improve your observability practices. It provides tools to collect data from across your technology stack, without getting bogged down in tool-specific deliberations. Ultimately, it helps facilitate the healthy performance of your applications and vastly improves business outcomes.