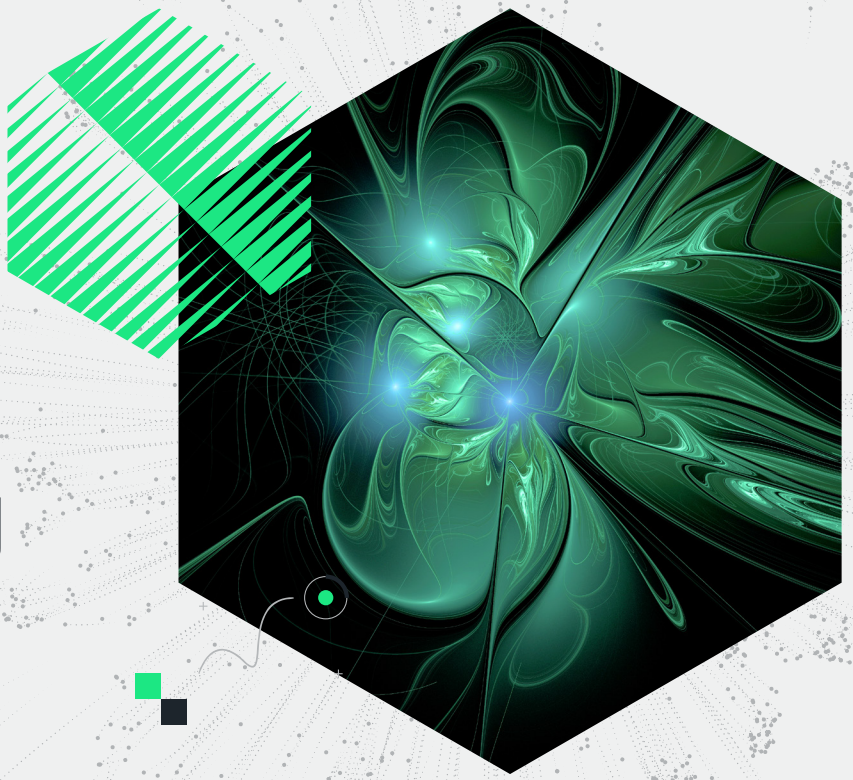
new relic



Why Distributed Tracing is Essential for APM

Reduce MTTR in complex distributed application environments

eBook / November 2022

Contents

Cutting through the complexity

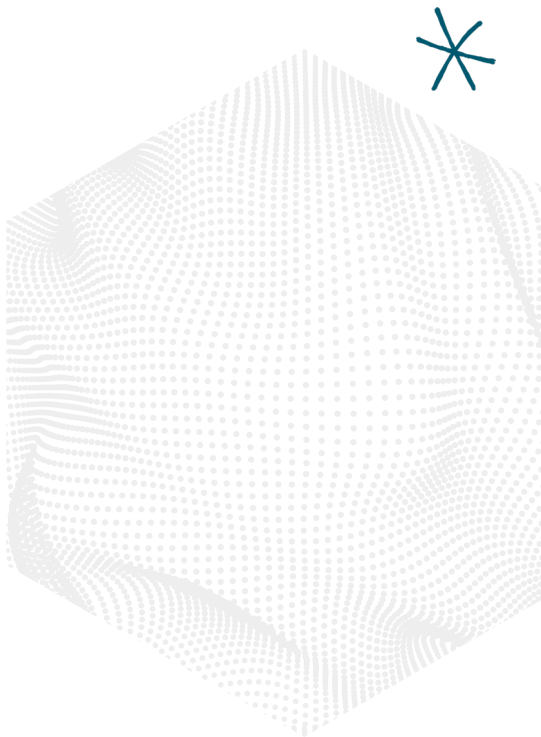
Modern software environments and architectures like microservices have the potential to accelerate application development. But in many organizations, software engineering teams face a complex environment, which makes it difficult to diagnose and resolve performance issues and errors before they impact reliability and the customer experience.

Microservices environments can include dozens to hundreds of services, making it hard to determine request paths and diagnose issues. And application performance monitoring (APM) burdens only increase with orchestration, automation, and CI/CD for frequent software deployments. Without the proper monitoring instrumentation, organizations risk their teams having to search for answers in distributed systems repeatedly, which increases mean time to resolution (MTTR) and takes time from innovative software development.

Observability cuts through software complexity and provides end-to-end visibility that enables teams to solve problems faster, work smarter, and create better digital experiences for their customers. Observability creates context and actionable insights by, among other things, combining four essential types of observability data: metrics, events, logs, and traces (MELT).

Traces—more precisely, distributed traces—are essential for software teams that have transitioned (or are considering a move) to the cloud and have adopted microservices architectures. That's because distributed tracing is the best way to understand quickly what happens to requests as they transit through the microservices that make up distributed applications.

Business leaders, DevOps engineers, product owners, site reliability engineers (SREs), software team leaders, or other stakeholders can use distributed tracing to find bottlenecks or errors and gain an edge with faster troubleshooting.



Tracing the path through distributed systems

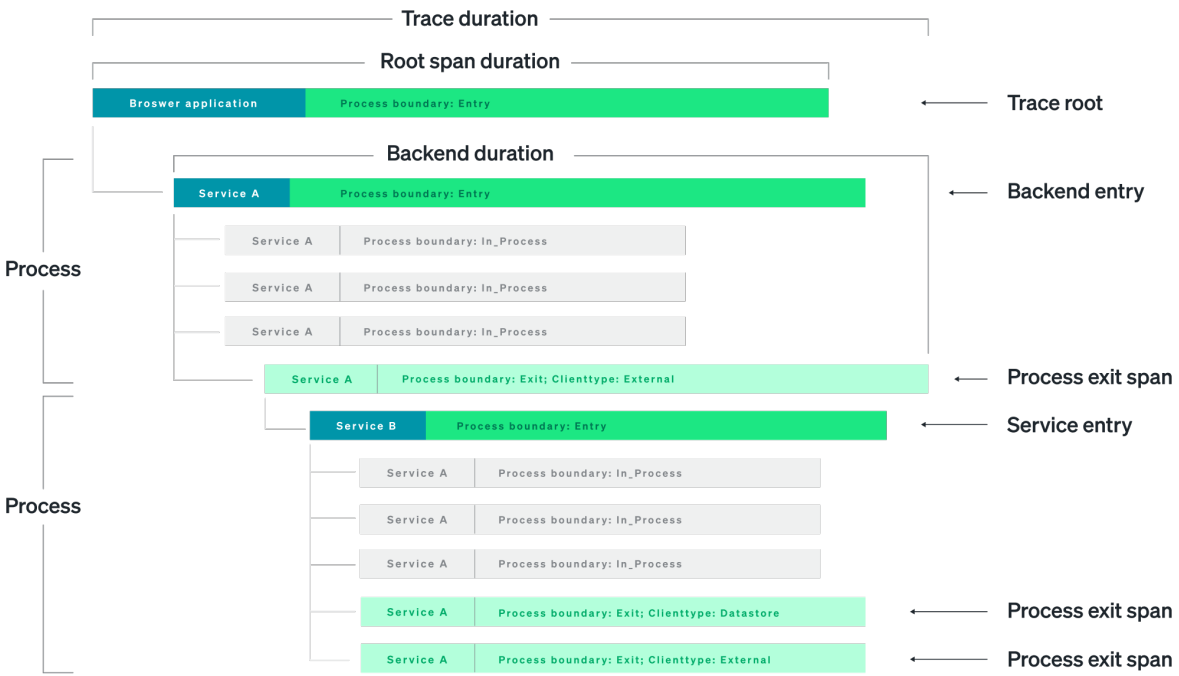
Distributed tracing is now table stakes for operating and monitoring modern application environments. When teams monitor software and system performance for observability, tracing is a way to monitor and analyze requests as they propagate through a distributed environment and hop from service to service.

Distributed tracing is the ability to trace a solution to track and observe service requests as they flow through distributed systems by collecting data as the requests go from one service to another. The trace data helps teams understand the flow of requests through the microservices environment and pinpoint where failures or performance issues occur in the system—and why.

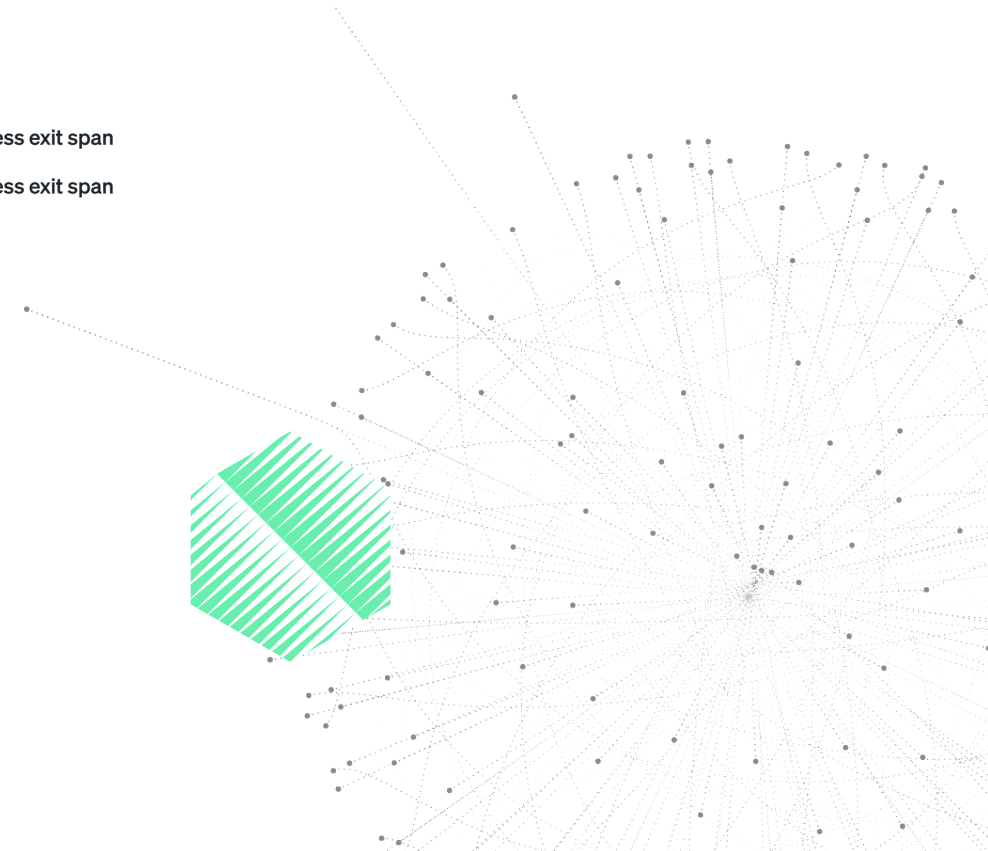
When teams instrument systems for distributed tracing, all transactions generate trace telemetry, from the frontend user to the backend database calls. For example, when customers click on a cart to make a purchase in an e-commerce application, that request travels through several distinct frontend and backend services across multiple containers, serverless environments, virtual machines, different cloud providers, on-premises (on-prem), or any combination of these. The request might include the inventory service to ensure there is inventory available, payment service, and shipping service. And ultimately the request completes and comes back to the user. Every time a request hops from one service to another, it emits a span with tracing telemetry. Once the request finishes, spans are stitched together to create a complete trace of the request's journey through the system.

With distibuted tracing, teams can:

- Trace the path of a request as it travels across a complex system.
- Understand upstream and downstream service dependencies.
- Discover the latency of the components along that path.
- Understand where bottlenecks are occurring in the request path.
- See and analyze where errors happen in the transaction at the individual service level.



Scatter chart and waterfall visualization showing how much time each request took on each step across application services



When to use traces

In general, distributed tracing is the best way for DevOps, operations, software, and SRE teams to get answers to specific questions quickly in environments where the software is distributed or relies on serverless architectures. As soon as a request involves a handful of microservices, having a way to see how all the different services are working together is essential.

Trace data provides context for what is happening across the application as a whole and between services and entities. If there were only raw events for each service in isolation, there would be no way to reconstruct a single chain between services for a particular transaction.

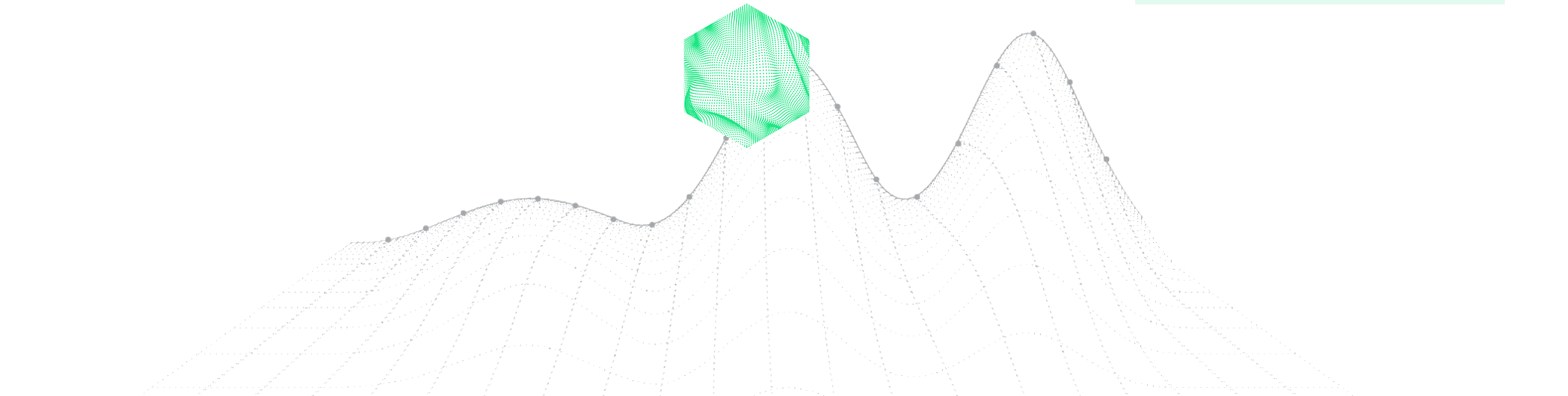
Because applications often call multiple other applications depending on the task they're trying to accomplish, they also often process data in parallel. So the call chain can be inconsistent and timing unreliable for correlation. The only way to ensure a consistent call chain is to pass trace context between each service to identify a single transaction uniquely through the entire chain.

This means teams should use distributed tracing to get answers to questions such as:

- What is the health of the services that make up a distributed system?
- What is the root cause of errors and defects within a distributed system?
- Where are performance bottlenecks that could impact the customer experience?
- Which services have problematic or inefficient code that teams should prioritize for optimization?

A quick guide to distributed tracing terminology:

- A **transaction** is the function and method calls that make up that unit of work in a software application. It starts when the method is called and ends when the method returns or errors out.
- A **request** is how applications, microservices, and functions talk to one another.
- A **trace** is performance data about requests as they flow through microservices.
- A **span** represents operations or segments that are part of a trace.
- A **root span** is the first span in a trace.
- A **child span** is a subsequent span, which can be nested.



How traces work

Stiched-together traces form special events called spans, which help track a causal chain through a microservices ecosystem for a single transaction. To accomplish spans, each service passes correlation identifiers, known as trace context, to each other. This trace context is used to add attributes to the span.

Timestamp	EventType	TraceID	SpanID	ParentID	ServiceID	Duration
11/8/2022 15:34:23	Span	2ec68b32	aaa111	bbb111	Vending Machine	23
11/8/2022 15:34:22	Span	2ec68b32	bbb111	aaa111	Vending Machine Backend	18
11/8/2022 15:34:20	Span	2ec68b32	ccc111	bbb111	Credit Card Company	15
11/8/2022 15:34:19	Span	2ec68b32	ddd111	ccc111	Issuing Bank	3

Example of a distributed trace composed of the spans in a credit card transaction

In the table above, the timestamp and duration data shows the credit card company has the slowest service in the transaction with 12 of the 23 seconds—more than half the time for this entire trace.

How did we get 12 seconds? The span to contact the issuing bank is called a child span. The span to contact the credit card company is its parent. So, if the bank request took three seconds, the credit card company took 15 seconds, and you subtract the child from the parent, it took 12 seconds to process the credit card transaction.

Connecting the dots

As organizations began moving to distributed applications, they quickly realized they needed a way to have visibility into individual microservices in isolation and the entire request flow. This migration is why distributed tracing became a best practice for gaining needed visibility into what was happening. And combining traces with the other three essential types of telemetry data—metrics, events, and logs—gives teams a complete picture of their software environment and performance for end-to-end observability.

Distributed tracing also requires trace context. This requirement means assigning a unique ID to each request, assigning a unique ID to each step in a trace, encoding this contextual information, and passing (or propagating) the encoded context from one service to the next as the request makes its way through an application environment. This process lets the distributed tracing tool correlate each step of a trace, in the correct order, along with other necessary information to monitor and track performance.

A single trace typically captures data about:

- Spans (service name, operation name, duration, and other metadata)
- Errors
- Duration of important operations within each service (such as internal method calls and functions)
- Custom attributes

W3C Trace Context has become the standard for propagating trace context across process boundaries. It lets all tracers and agents that conform to the standard participate in a trace, with trace data propagated from the root service to the terminal service. Many observability vendors, including New Relic, fully support the W3C Trace Context standard.

Why organizations need distributed tracing

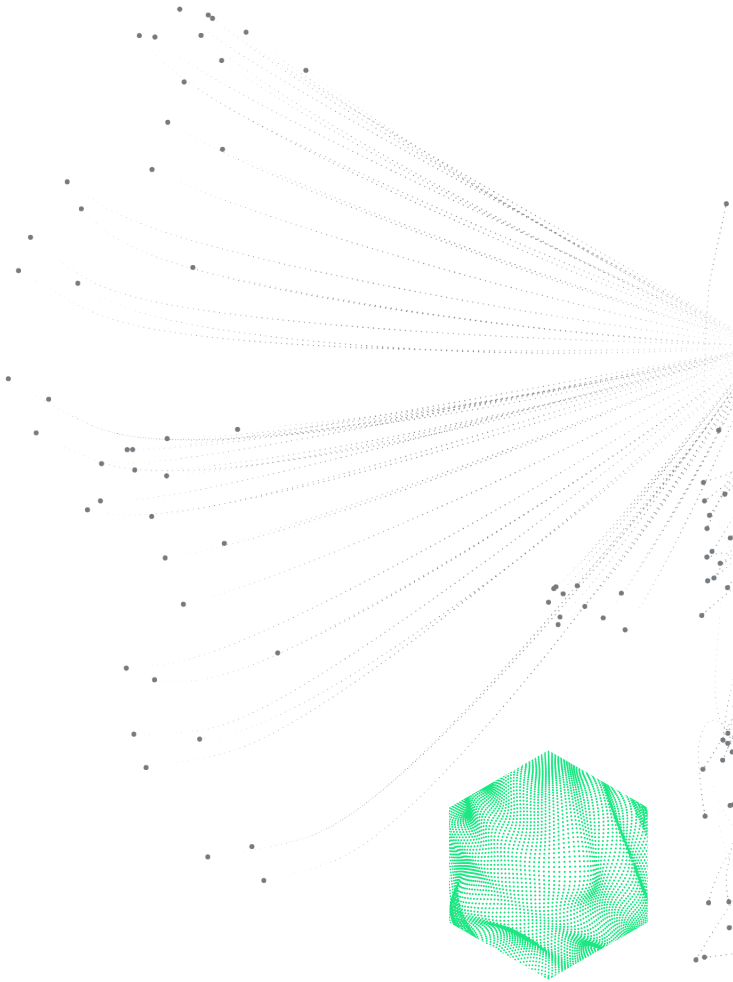
As new technologies and practices—cloud, microservices, containers, serverless functions, DevOps, site reliability engineering, and more—increase velocity and reduce the friction of getting software from code to production, they also introduce new challenges:

- More points of failure within the application stack
- Increased MTTR due to the complexity of the application environment
- Less time for teams to innovate because they need more time to diagnose problems

For example, a slow-running request might impact the experience of a set of customers. That request is distributed across multiple microservices and serverless functions. Several teams own and monitor the various services involved in the request, and none have reported any performance issues with their microservices. Without a way to view the performance of the entire request across the different services, it's nearly impossible to pinpoint where and why the high latency is occurring and which team should address the issue. As part of an end-to-end observability strategy, distributed tracing addresses the challenges of modern application environments.

By deeply understanding the performance of every service—both upstream and downstream—software teams can more effectively and quickly:

- Identify and resolve issues to minimize the impact on the customer experience and business outcomes.
- Measure overall system health and understand the effect of changes on the customer experience.
- Prioritize high-value areas for improvement to optimize the digital customer experiences.
- Innovate continuously with confidence to outperform the competition.



Gaining visibility into the data pipeline

Distributed tracing requires the reporting and processing of tracing telemetry. The volume of trace data can grow exponentially over time as the volume of requests increases and as teams deploy more microservices within the environment.

For this reason, many organizations use data sampling to manage the complexity and cost associated with transmitting trace activity. Ideally, the sampled data represents the characteristics of the larger data population.

Software teams need the flexibility to choose head- or tail-based sampling to meet the monitoring requirements for each application.

Efficient head-based sampling

Head-based sampling collects and stores trace data randomly while the root (first) span is processed to track and analyze what happens to a transaction across all of the services it touches. Typically, head-based sampling happens within the agent responsible for collecting trace telemetry by randomly selecting which traces to sample for analysis. The sampling decisions occur before traces are complete. Because there is no way to know which trace might encounter an issue, teams might miss traces that contain unusually slow processes or errors.

Head-based sampling works well to provide an overall statistical sampling of requests through a distributed system. It does a good job of catching traces with errors or latency in applications with a lower volume of transactions and environments with a mix of monolith and microservices-based architectures. Head-based sampling is an efficient way to sample a vast amount of trace data in real time, and there is little to no impact on application performance.

Advantages of head-based sampling

- Works well for applications with lower transaction through put
- Fast and simple to get up and running
- Appropriate for blended monolith and microservice environments where monoliths still reign supreme
- Little-to-no impact on application performance
- A low-cost solution for sending tracing data to third-party vendors
- Statistical sampling provides adequate transparency into the distributed system

Limitations of head-based sampling

- Traces are sampled randomly
- Sampling happens before a trace has fully completed its path through many services, so there is no way to know upfront which trace may encounter an issue
- In high-throughput systems, traces with errors or unusual latency might be sampled out and missed

Actionable traces with tail-based sampling

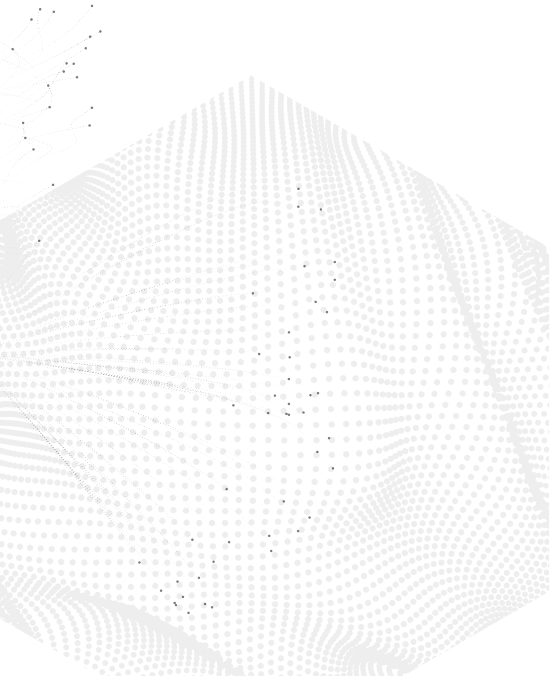
Distributed tracing with tail-based sampling helps software teams troubleshoot issues in highly distributed, high-volume systems where teams must observe all trace telemetry and sample the traces that contain errors or unusual latency. Tail-based sampling collects all information about that trace when it's complete.

Tail-based sampling is less of a nice to have and more of a requirement when teams need the highest level of granularity for troubleshooting.

Some organizations need their distributed tracing tool to observe and analyze every span—every hop between services—and surface the most actionable traces for troubleshooting because downtime could cost millions of dollars, especially during peak events.

For example, an organization with an average span load of three million spans per minute sees spikes of 300 million spans per minute when it launches a new product. Traditional head-based sampling is inadequate for this type of organization with a high transaction volume.

Not every trace is equal. To choose the best sampling method, teams should evaluate based on the use case and cost-over-benefit analysis and consider the monitoring needs of each application.

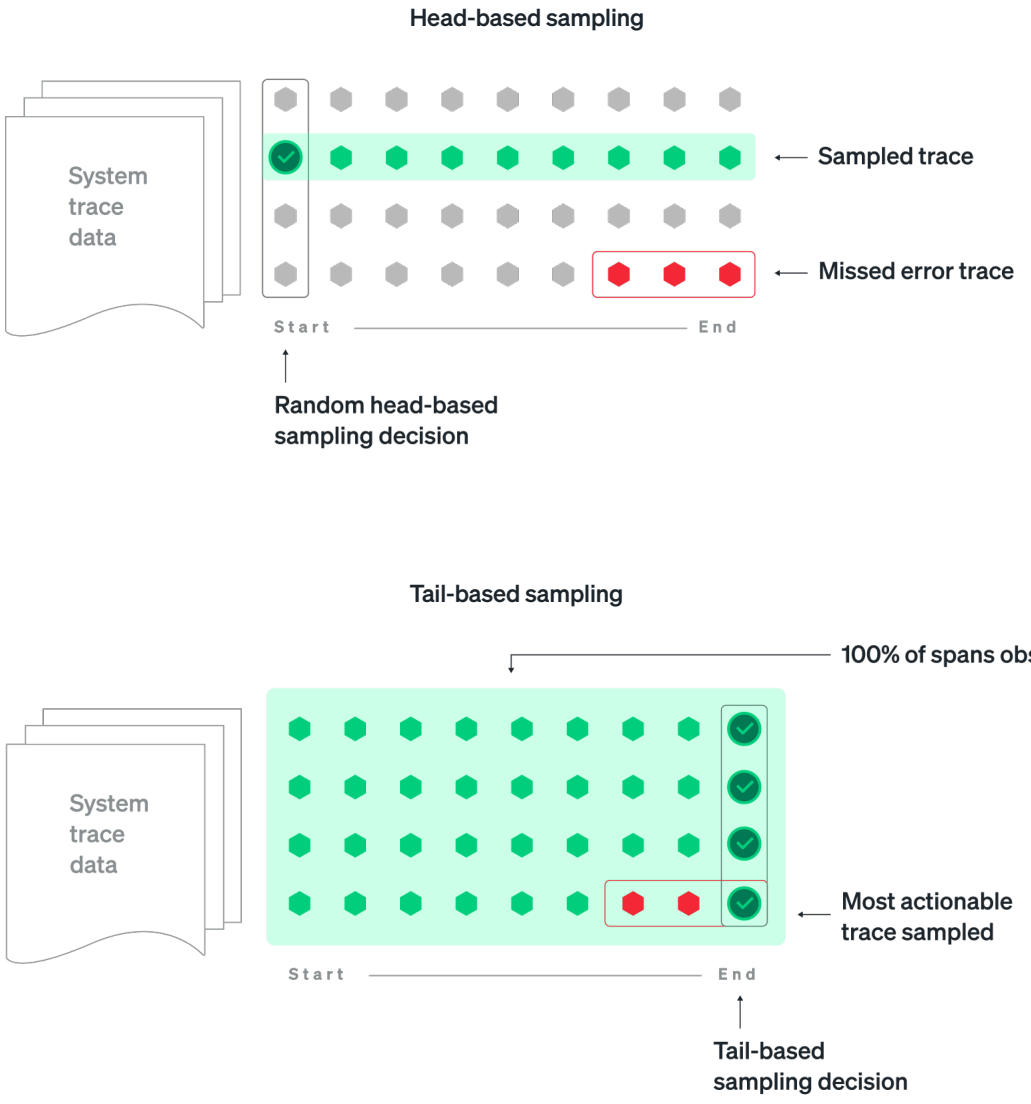


Advantages of tail-based sampling

- Observes and analyzes 100% of traces
- Samples after traces are fully completed
- Visualizes traces with errors or uncharacteristic slowness more quickly

Limitations of tail-based sampling

- May require additional gateways, proxies, and satellites to run sampling software
- Requires some toil to manage and scale third-party software in some cases
- Incurs additional costs for transmitting and storing more data



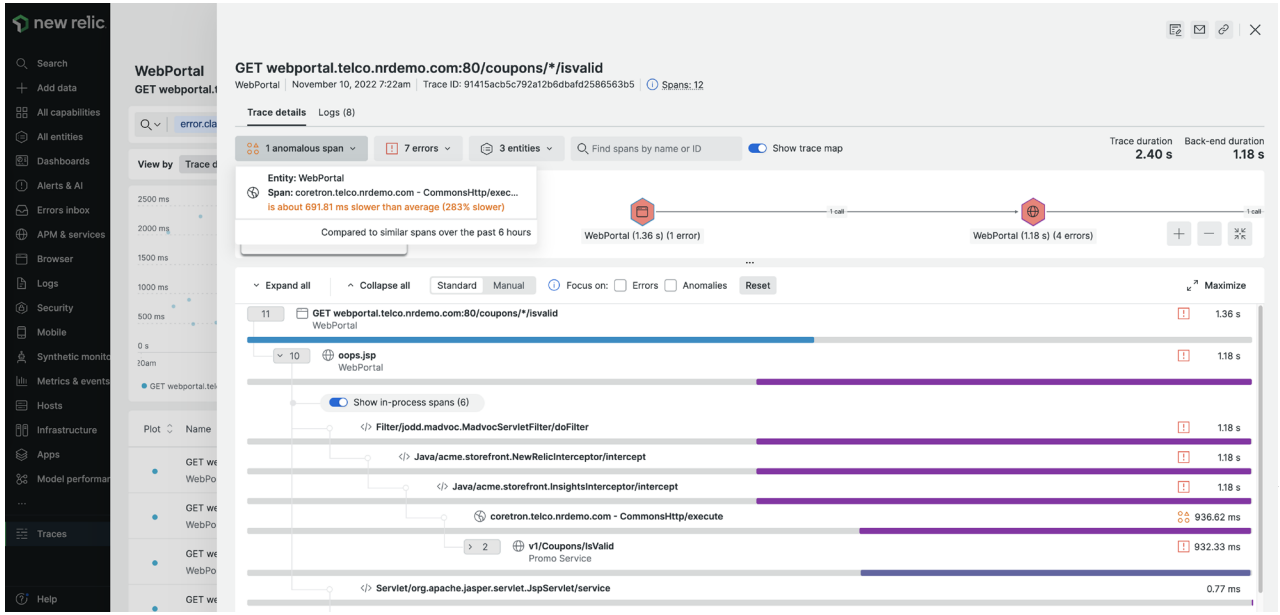
Traditional head-based sampling (top) and tail-based sampling (bottom)

Analysis and visualization

Collecting trace data is a waste of time if software teams don't have an easy way to analyze and visualize the data across complex architectures. A comprehensive observability platform allows teams to see all their telemetry and business data in one place. It also provides the context they need to derive meaning, take the right action quickly, and work with the data in meaningful ways.

A distributed trace visualization ideally has a tree-like structure. The visualization should include child spans that refer to one parent span and lets teams see which spans have high latency and errors within a trace. It also helps teams understand the exact error details and what services are slow, with detailed attributes to find issues and fix them quickly.

Observability vendors like New Relic use this visualization structure for troubleshooting and analysis.



New Relic distributed tracing

Addressing the management burden

Troubleshooting distributed systems is a classic needle-in-a-haystack problem, and instrumenting systems for tracing then collecting and visualizing the data can be labor-intensive and complex to implement. Fully managed software-as-a-service (SaaS) solutions allow teams to eliminate the burden of deploying, managing, and scaling third-party gateways or satellites for data collection.

The New Relic observability platform makes it easy to instrument applications with a single agent deployment for almost any programming language and framework. Teams can also use open-source tools and open instrumentation standards to instrument environments. OpenTelemetry is considered the standard for open-source instrumentation and telemetry collection.

The New Relic platform also offers a fully managed tail-based sampling service that observes and analyzes 100% of spans across a distributed system and provides visualizations for traces with errors or unusual latency, so teams can quickly identify and troubleshoot issues.

The platform observes every span and provides metrics, error data, and essential traces in a single view. It provides critical insights by saving the most actionable data to the New Relic platform. The result is unparalleled visibility into distributed systems, enabling teams to understand the impact of downstream latency or errors with detailed metrics and then drill down to the saved trace data for the most relevant traces.

Distributed tracing is included with New Relic APM, with low-latency and low-cost data transfer from New Relic agents, instrumentation within serverless functions, or any other data source, including third-party instrumentation.

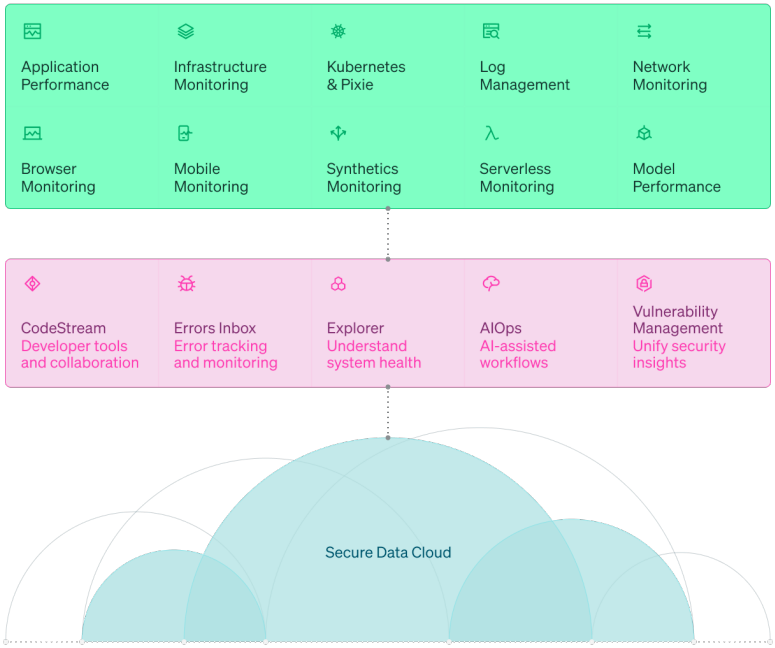
With New Relic, you can:

- Enjoy a fully managed cloud-local service that scales on-demand.
- Observe and analyze 100% of the traces across distributed systems.
- Visualize the most actionable traces that contain errors or unusual latency.
- Eliminate the toil of deploying, managing, supporting, and scaling third-party gateways or satellites in environments.
- Leverage its full support of open instrumentation and standards for trace telemetry.
- Reduce the cost of data egress charges from proximity to cloud workloads.
- Troubleshoot more efficiently.
- Reduce mean time to detection (MTTD) and MTTR with high-fidelity, actionable traces.
- Empower engineers and developers to focus on more important work, such as developing new features.

Heads or tails? You don’t need to flip a coin

New Relic offers flexible options for distributed tracing so teams can make head- or tail-based sampling decisions at the application level. For critical applications where teams need to observe and analyze every trace, they can select tail-based sampling without worrying about managing sampling infrastructure.

New Relic is the only observability vendor that gives software teams the flexibility to select distributed tracing with head-based sampling or fully managed tail-based sampling. With less to manage, there’s more room for innovation and gaining a competitive advantage.



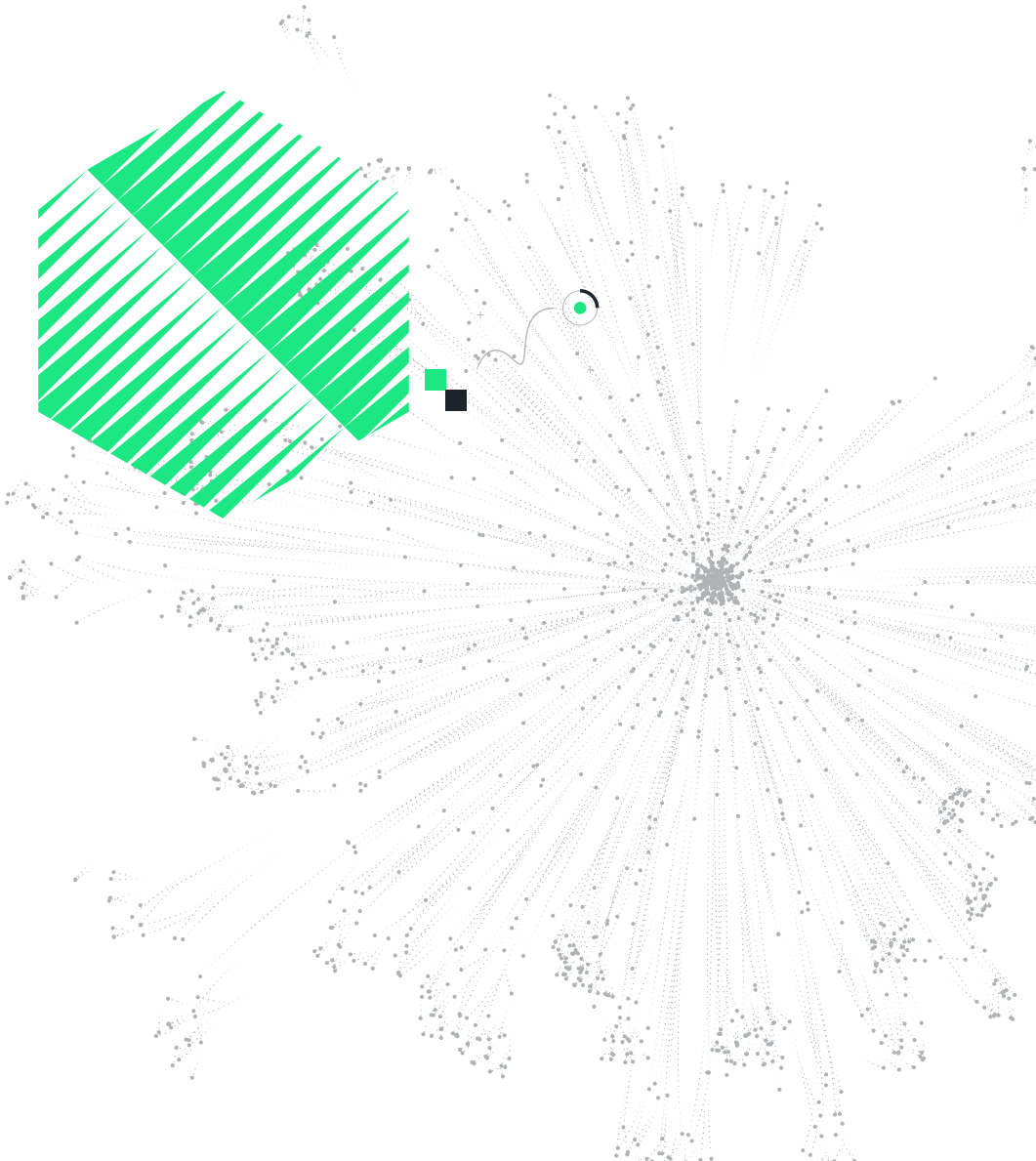
The New Relic observability platform incorporates log management, APM, distributed tracing, infrastructure monitoring, serverless monitoring, mobile monitoring, browser monitoring, synthetic monitoring, Kubernetes monitoring, and more.

Next steps

To begin using New Relic APM with distributed tracing, sign up for a free account today. Free accounts include 100 GB/month of data ingest, one full platform user, and unlimited basic users.

Already have a New Relic account? Getting started with New Relic APM distributed tracing is easy; just use our latest APM agent. Learn about distributed tracing setup options.

Get Started Now



About New Relic

As a leader in observability, New Relic empowers engineers with a data-driven approach to planning, building, deploying, and running great software. New Relic delivers the only unified data platform with all telemetry—metrics, events, logs, and traces—paired with powerful full-stack analysis tools to help engineers do their best work with data, not opinion.

Delivered through the industry's first usage-based pricing that's intuitive and predictable, New Relic gives engineers more value for their money by helping improve planning cycle times, change failure rates, release frequency, and MTTR. This helps the world's leading brands and hyper-growth startups to improve uptime, reliability, and operational efficiency and deliver exceptional customer experiences that fuel innovation and growth.

new relic.