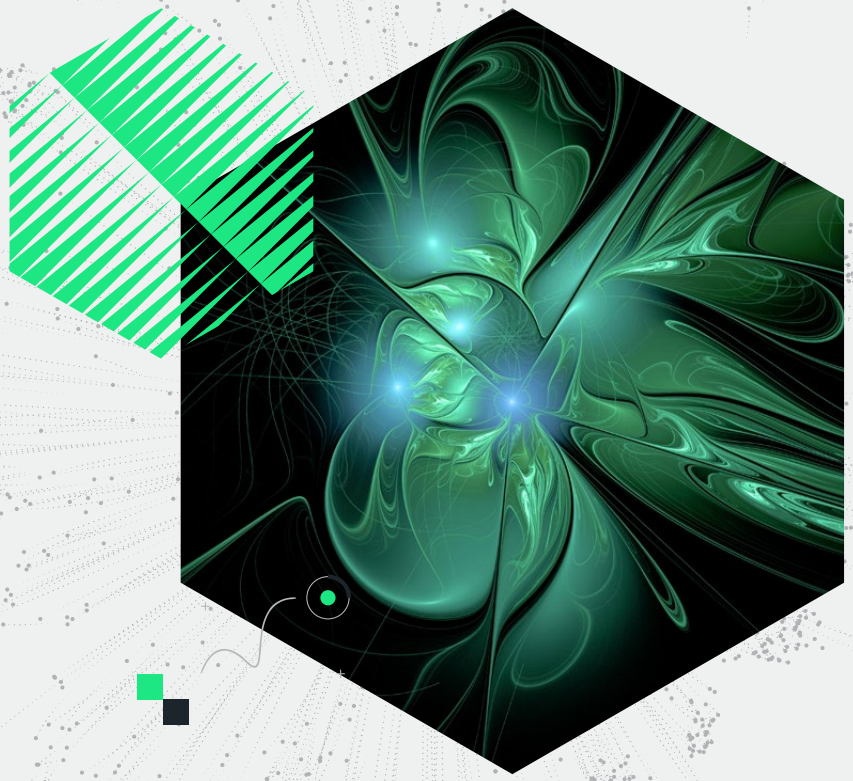
new relic



Distributed Tracing: Rückgrat von APM

Von verteilten Anwendungsumgebungen und verkürzten Lösungszeiten

E-Book / November 2022

Inhalt

Kein Platz für Komplexität

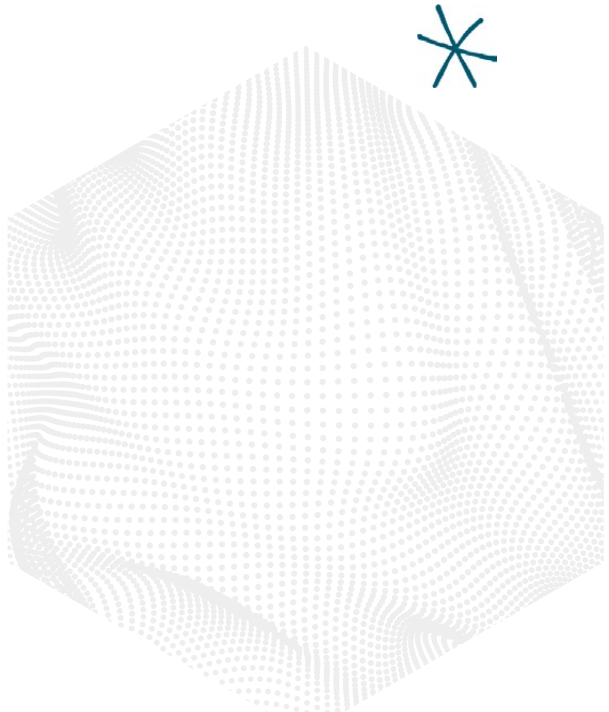
Moderne Software-Umgebungen etwa auf Basis von Microservices-Architekturen gelten als Potenzialgeber für mehr Dynamik in der Anwendungsentwicklung. Doch für ihre Protagonisten – die Teams vom Software-Engineering – erweisen sie sich allzu oft eher als Bremse. Denn sie stehen vor einer Umgebung, die ihnen die Diagnose von Performance-Problemen und Fehlern erheblich erschwert. Gleiches gilt entsprechend für deren Behebung, bevor sie sich auf Stabilität und Kundenerlebnis auswirken können.

Das Problem: Microservices-Umgebungen setzen sich aus dutzenden, mitunter auch hunderten Einzelservices zusammen. Die Rückverfolgung von Abfragepfaden für Problemdiagnosen ist daher mit enormem Aufwand verbunden. Orchestrierung, Automatisierung und im Zuge von CI/CD-Methodiken verkürzte Deployment-Zyklen erweitern den Aufgabenkatalog rund um Application Performance Monitoring (APM) dabei noch zusätzlich. Fehlt es hier Instrumentierung, können in verteilten Systemen wie diesen erst vielfach wiederholte Diagnosen Antworten auf Problemstellungen und wichtige Fragen liefern – ein Hemmschuh für Lösungszeit und Software-Innovation gleichermaßen.

Observability hilft Software-Teams dabei, diese Komplexität anhand einer Methodik für End-to-End-Transparenz zu entflechten, mit der sie Probleme schneller beheben, fundierte Entscheidungen treffen und bessere Kundenerlebnisse gestalten. Möglich wird dies durch Kontext und konkret umsetzbare Insights, generiert aus einem Verbund verschiedenster Informationen, in deren Mittelpunkt die vier Observability-Grundessenzen Metrics, Events, Logs und Traces (MELT) stehen.

Dabei bilden Traces bzw. deren Variante in Form von Distributed Traces ein immens wichtiges Element, wenn es um Cloud-Initiativen und in diesem Kontext etwa Microservices-Architekturen geht. Denn über sie wird es möglich, Anfragen innerhalb der Microservices, die hinter verteilten Anwendungen stehen, schnell und effizient nachzuvollziehen.

So liefert Distributed Tracing Aufschluss über Bottlenecks ebenso wie Fehlerquellen, gibt so von Leadership, DevOps-Teams und Produktverantwortlichen bis hin zu Site Reliability Engineers (SRE) und anderen Stakeholdern dem gesamten Team vom Software-Engineering starke Troubleshooting-Prozesse und damit mehr Wettbewerbskraft an die Hand.



Ein roter Faden durchs Labyrinth verteilter Systeme

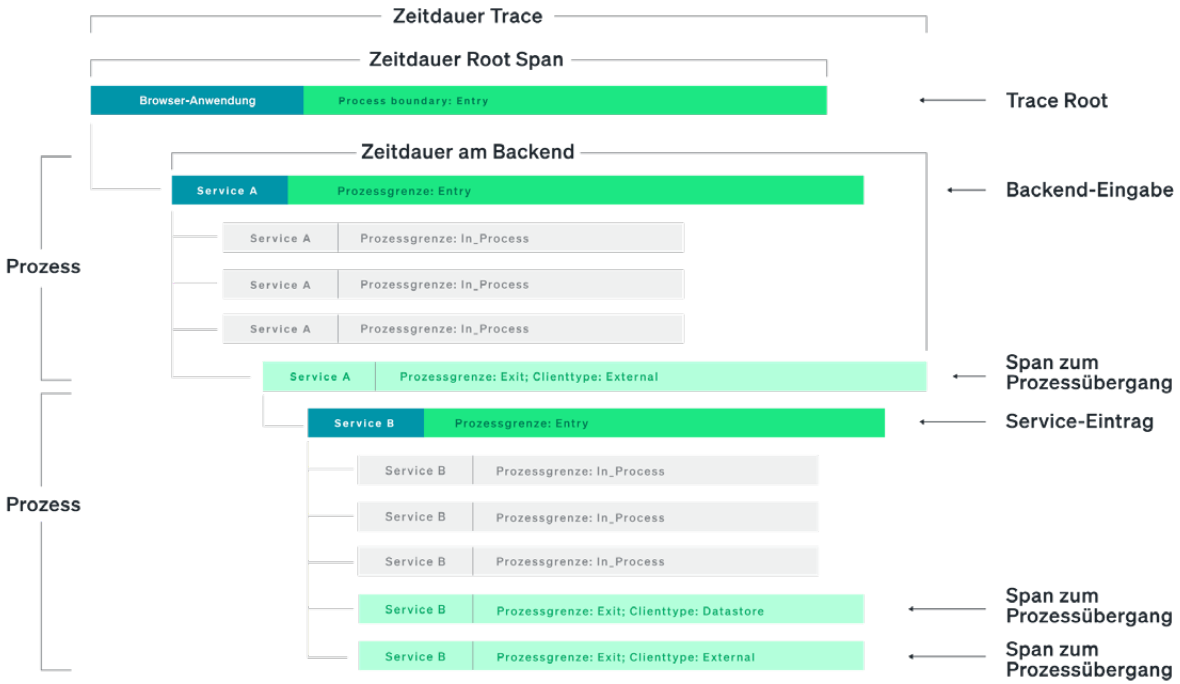
Anhand dieser Analogie lässt sich die tragende Rolle von Distributed Tracing für Betrieb und Monitoring moderner Anwendungsumgebungen gut beschreiben: Via Tracing wird gewissermaßen der Faden gespannt, über den sich nachvollziehen lässt, wie Anfragen durch verteilte Umgebungen von einem Service zum nächsten übergehen.

Konkret umgesetzt wird dies durch die Erfassung der Daten, die Service-Anfragen beim Durchlaufen der verschiedenen Entitäten innerhalb verteilter Systeme generieren. Hierdurch werden deren Wege durch die Microservice-Umgebung und so auch die Punkte nachvollziehbar, an denen Fehler oder Performance-Probleme aufgetreten sind – ebenso wie ihre Ursache.

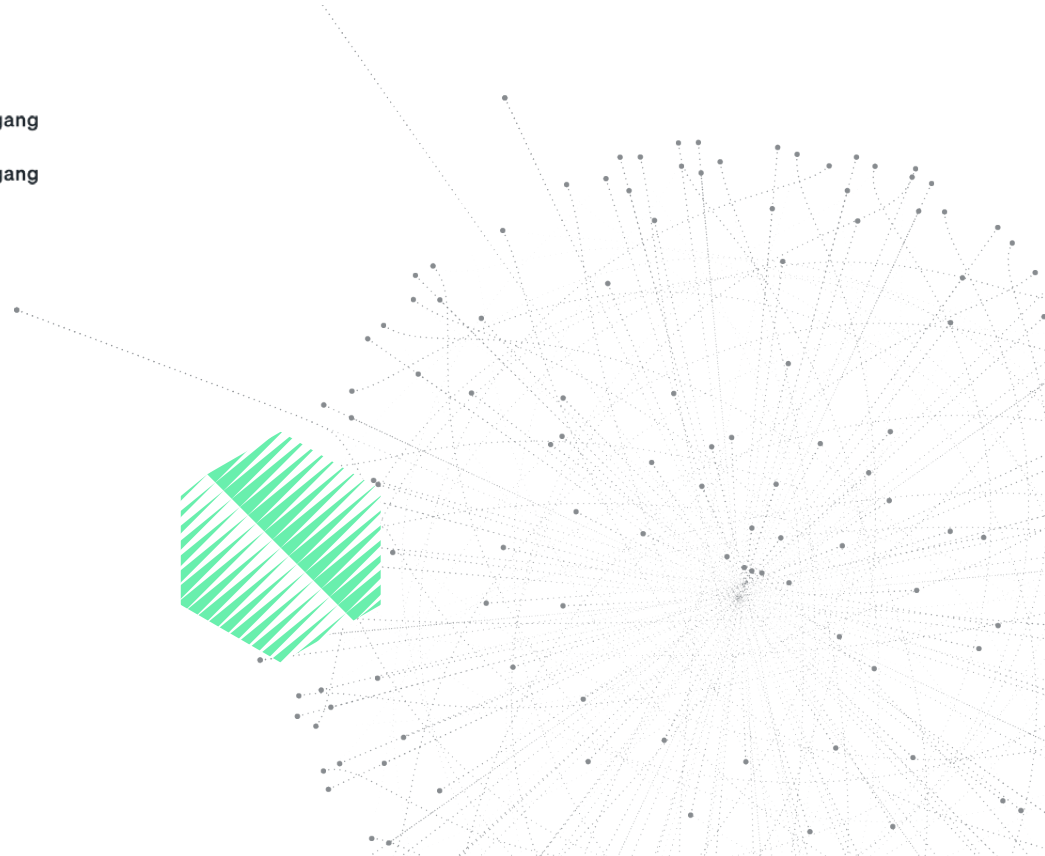
Werden Systeme für Distributed Tracing instrumentiert, werden von Frontend-Benutzer:innen bis hin zu Datenbankaufrufen im Backend Telemetriedaten zu sämtlichen Transaktionen generiert. So etwa beim Abschluss eines Kaufs in einer E-Commerce-Anwendung durch Klicken auf den Einkaufskorb: Eine solche Anfrage durchläuft diverse eindeutige Services im Front- wie auch im Backend, die über verschiedenste Container und Serverless-Umgebungen auf virtuellen Maschinen, Cloud- und On-Prem-Infrastruktur oder beliebigen Kombinationen aus diesen verteilt sind. Dabei umfasst die Anfrage Stationen wie den Service zur Abfrage des Warenbestands sowie zur Zahlung- und Versandabwicklung und wird abschließend wieder zurückgeleitet. Auf den Stationen, die die Anfrage dabei von einem Service zum anderen durchläuft, hinterlässt sie Spans mit Tracing-Telemetrie. Diese werden nach Abschluss zusammengeführt und so die komplette Trace abgebildet, die die Anfrage auf ihrem Weg durch das System generiert hat.

Mit Distributed Tracing wird Folgendes möglich:

- Nachzeichnen von Abfragepfaden innerhalb komplexer Systeme
- Klarheit über Upstream- und Downstream-Abhängigkeiten von Services
- Bestimmung der Latenz der an einem Pfad beteiligten Komponenten
- Aufdecken von Bottlenecks im Abfragepfad
- Identifikation und Analyse von Transaktionsfehlern auf Ebene einzelner Services



Punktwolke und Wasserfalldiagramm zur Visualisierung der Verarbeitungszeit der einzelnen Anfragen durch die verschiedenen Anwendungsservices



Darum braucht es Traces

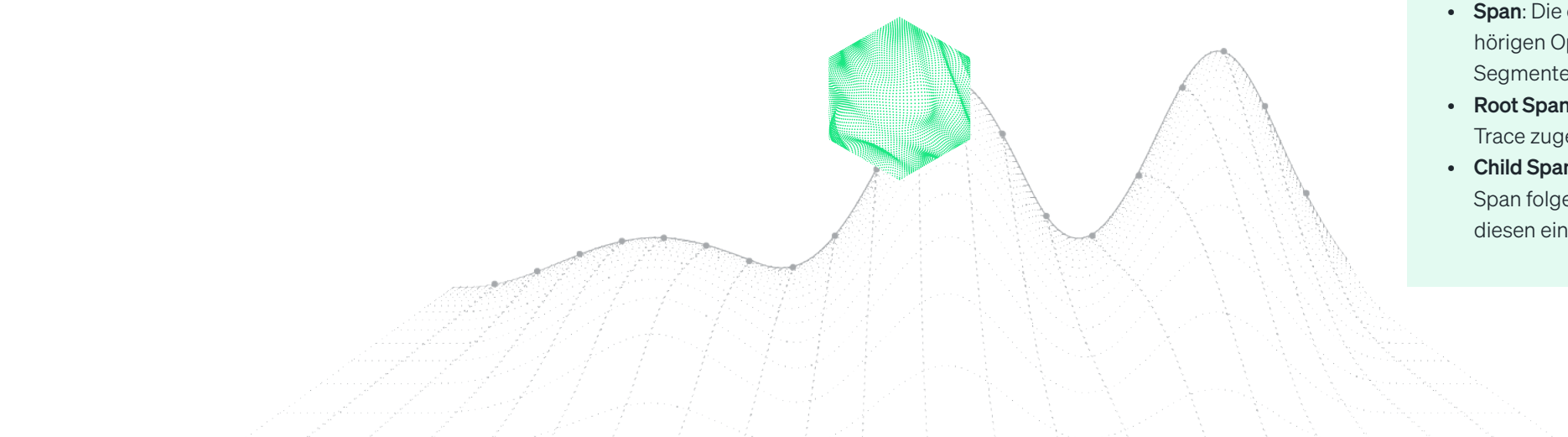
Ganz allgemein ist Distributed Tracing die beste Methode, um schnell und zuverlässig Antworten auf spezifische Fragen in Umgebungen zu erhalten, in denen Software verteilt oder auf Serverless-Architekturen ausgeführt wird. Denn sobald Anfragen sich nur auf eine Handvoll Microservices verteilen, ist Transparenz für die Interaktion zwischen den einzelnen Services unabdingbar.

Trace-Daten liefern Kontext zu den Vorgängen innerhalb der Gesamtanwendung sowie zwischen den einzelnen ihr zugrunde liegenden Services und Entitäten. Würden dagegen nur die Event-Daten zu den einzelnen Services unzusammenhängend erfasst, ließe sich die Kette der verschiedenen Operationen hinter einer Transaktion nicht lückenlos rekonstruieren.

Je nach Task, den eine Anwendungen ausführt, muss dieser auch andere aufrufen. Damit verbunden ist häufig eine parallele Verarbeitung von Daten, infolge derer keine durchgängige Kette zwischen den einzelnen Aufrufen entsteht. Zudem lässt sich womöglich auch über die Zeitstempel kein verlässlicher Zusammenhang herstellen. Konsistenz in der Anrufkette gewährleisten kann daher nur die Übergabe von Trace-Kontext zwischen den einzelnen Services. Denn so wird es möglich, jede Transaktion innerhalb der Kette eindeutig zu identifizieren.

So hilft Distributed Tracing dabei, Aufschluss über Aspekte wie die folgenden zu erhalten:

- Health-Status der Services innerhalb eines verteilten Systems
- Ursache von Fehlern und anderen Problemen innerhalb eines verteilten Systems
- Performance-Bottlenecks mit potenziellen Auswirkungen auf das Benutzererlebnis
- Services mit problematischem oder ineffizientem Code, deren Optimierung es zu priorisieren gilt



Wichtige Begrifflichkeiten rund um Distributed Tracing:

- **Transaktion**: Die von der Einleitung bis zum Abschluss einer Transaktion von einer Software-Anwendung ausgeführten Funktions- und Methodenaufrufe. Das Ende einer Transaktion wird entweder durch Rückgabe der den Methoden zugehörigen Abfragen oder durch einen Fehler markiert.
- **Abfrage**: Über sie erfolgt die Kommunikation zwischen Anwendungen, Microservices und Funktionen.
- **Trace**: Die Performance-Daten, die zu Abfragen auf ihrem Weg durch die verschiedenen Microservices erfasst werden.
- **Span**: Die einer Trace zugehörigen Operationen oder Segmente.
- **Root Span**: Der erste einer Trace zugehörige Span.
- **Child Span**: Der auf den Root Span folgende, ggf. auch in diesen eingebettete Span.

Das Konzept hinter Traces

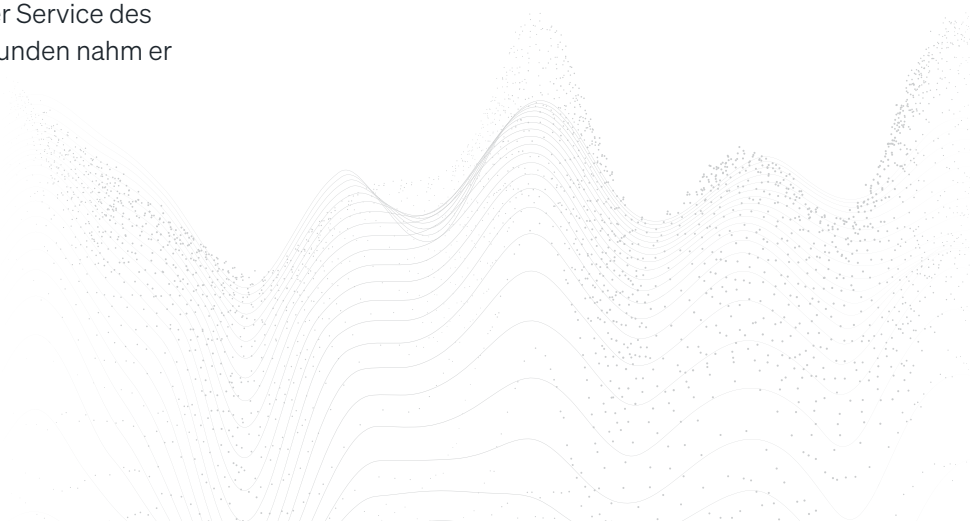
Traces bilden die Elemente, aus deren Zusammenführung die Event-Kategorie Span entsteht. Über diese wiederum lässt sich die kausale Kette nachzeichnen, die hinter einer bestimmten Transaktion innerhalb eines Microservices-Ökosystems steht. Gebildet werden Spans durch die Korrelation von IDs, die die einzelnen Services im Zuge der Transaktion aneinander weitergeben. Anhand dieser Informationen, Trace-Kontext genannt, lassen sich den einzelnen Spans Attribute zuweisen.

Timestamp	EventType	TraceID	SpanID	ParentID	ServiceID	Zeitspanne
8.11.2022, 15:34:23	Span	2ec68b32	aaa111	bbb111	Frontend der Verkaufsstelle	23
8.11.2022, 15:34:22	Span	2ec68b32	bbb111	aaa111	Backend der Verkaufsstelle	18
8.11.2022, 15:34:20	Span	2ec68b32	ccc111	bbb111	Kreditkarten-institut	15
8.11.2022, 15:34:19	Span	2ec68b32	ddd111	ccc111	Ausgebende Bank	3

Distributed Trace, wie sie sich etwa aus den Spans einer Kreditkartentransaktion zusammensetzt

Gemäß der in der Tabelle oben aufgeführten Daten zu Timestamp und Zeitdauer erweist sich der Service des Kreditkarteninstituts als der langsamste innerhalb der Transaktion: Mit 12 von insgesamt 23 Sekunden nahm er mehr als die Hälfte der Zeit in Anspruch, die für diese Trace erfasst wurde.

Doch warum 12 Sekunden?
Die Kontaktaufnahme mit der ausgebenden Bank bildet den sogenannten Child Span. Diesem übergeordnet ist der Span zur Anfrage beim Kreditkarteninstitut, da dieses zuerst kontaktiert werden muss. Nun wurden für die Anfrage bei der Bank ja 3 Sekunden benötigt und für die beim Kreditkarteninstitut 15. Die Differenz aus Child und übergeordnetem Span ergibt dann schließlich 12 Sekunden, die zur Verarbeitung der Kreditkartentransaktion benötigt wurden.



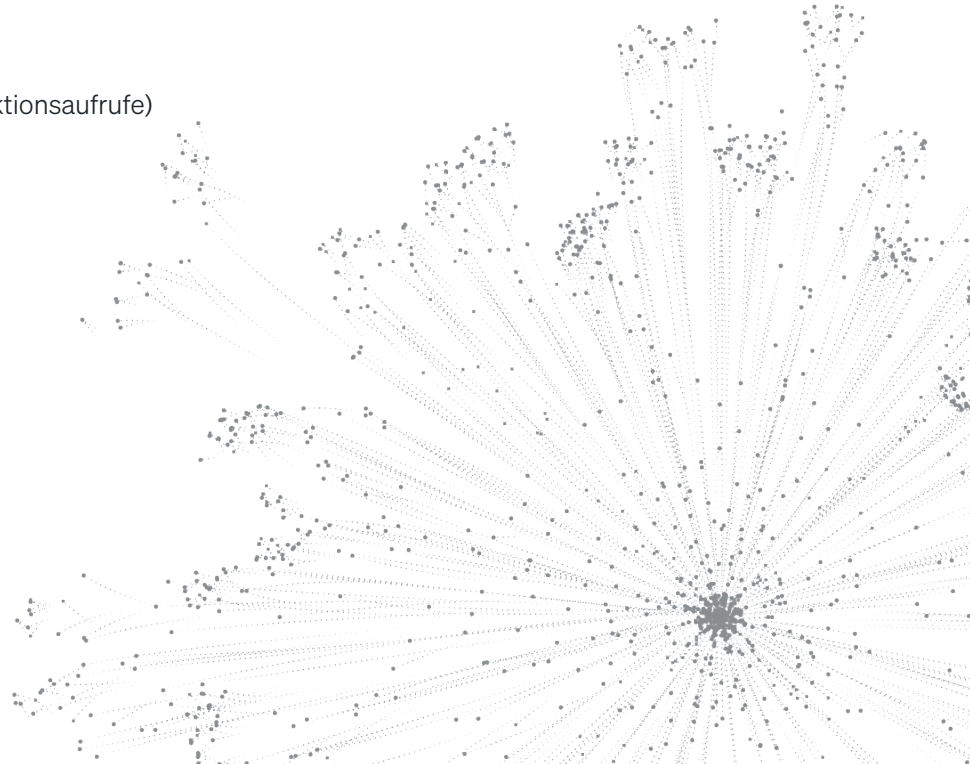
Vom Kettenglied zum Kontext

Im Zuge der Einführung verteilter Anwendungen wurde schnell klar: Wenn es darum ging, die Vorgänge innerhalb der einzelnen ihnen zugehörigen Microservices nachvollziehbar zu machen, stießen klassische Konzepte an ihre Grenzen. Gleiches galt für den Anfrage-Flow als Ganzes. Distributed Tracing konnte hier nicht nur Abhilfe schaffen, sondern etablierte sich auch als Best Practice in diesem Kontext. Im Verbund mit dem Dreigespann aus Metrics, Events und Logs bildet Distributed Tracing zudem das elementare vierte Glied der Telemetrie-Datentypen, aus denen End-to-End-Observability für die gesamte Software-Umgebung wie auch für ihre Performance entsteht.

Für die Umsetzung entscheidend ist dabei der Trace-Kontext, zusammengesetzt aus jeweils einer eindeutigen ID für jede Anfrage sowie für die einzelnen einer Trace zugehörigen Schritte. In dieser kodierten Form können die Kontext-Details dann auf dem Weg, den eine Anfrage durch die Anwendungsumgebung nimmt, von einem Service zum nächsten übergeben werden. Ein Distributed-Tracing-Tool kann anhand dieser Details wiederum die einzelnen Schritte einer Trace in der korrekten Reihenfolge nachzeichnen und diese zudem mit anderen zum Performance-Monitoring und -Tracking relevanten Details in Zusammenhang setzen.

So umfasst eine einzelne Trace in der Regel folgende Daten:

- Spans (Name des Service und der Operation, Ausführungsdauer sowie zusätzliche Metadaten)
- Fehler
- Ausführungsdauer wichtiger Operationen innerhalb jedes Service (z. B. interne Methoden- und Funktionsaufrufe)
- Custom-Attribute



Die Übermittlung von Trace-Kontext über Prozessgrenzen hinweg erfolgt heute zumeist via W3C Trace Context. Der Standard ermöglicht es Tracing-Tools und Agents, Kontext-Daten vom Root- bis zum Terminal-Service zu einer Trace beizusteuern. Der Standard wird von einer Vielzahl von Observability-Anbietern unterstützt, so auch von New Relic.

Warum Distributed Tracing?

Cloud, Microservices, Container oder Serverless bieten neue Technologie-Konzepte, die im Tandem mit modernen Delivery-Methodiken wie DevOps oder Site Reliability Engineering (SRE) dazu beitragen, Software-Code schneller und nahtloser in die Produktion zu überführen. Zugleich gehen mit ihnen auch neue Herausforderungen einher. Dazu gehören:

- Mehr Fehlerquellen innerhalb des Anwendungs-Stacks
- Verzögerte Problembehebung infolge der Komplexität der Anwendungsumgebung
- Gebremste Innovation aufgrund von hohem Zeitaufwand für Problemdiagnosen

Dies macht etwa ein Beispiel deutlich, bei dem eine langsam ausgeführte Abfrage das Kundenerlebnis beeinträchtigt: Diverse Microservices und Serverless-Funktionen sind an der Anfrage beteiligt, doch von den Teams, die mit diesen jeweils betraut sind, registriert kein einziges Performance-Probleme in seinem jeweiligen Verantwortungsbereich. Was ihnen fehlt, ist ein lückenloses Bild der Anfrage-Performance über sämtliche Services hinweg. Wo oder warum es zu einem Latenzanstieg kam, ist daher praktisch unmöglich festzustellen. Und so nimmt auch keiner Notiz von dem Problem oder nimmt sich ihm an. Daher braucht es in einer modernen Anwendungsumgebung wie dieser Distributed Tracing, eingebettet in eine End-to-End-Methodik für Observability.

Denn damit wird die Performance jedes Service klar nachvollziehbar – Upstream ebenso wie Downstream. So agieren die verantwortlichen Teams nicht nur schneller, sondern auch effektiver. Bemerkbar macht sich dies wie folgt:

- Effizientere Erkennung und Behebung von Problemen, dadurch Reduzierung von Business-Risiken durch CX-Schwächen
- Präzise Daten zur System-Health und somit Klarheit zur ihren Auswirkungen auf das Kundenerlebnis
- Punktgenaue Priorisierung der kritischsten und potenzialstärksten Bereiche zur Optimierung digitaler Kundenerlebnisse
- Stärkung der Wettbewerbsfähigkeit durch konsequente Innovationskultur

Transparenz für die Daten-Pipeline

Damit Distributed Tracing und mit ihm transparentes Reporting möglich wird, braucht es vor allem eines: Die Verarbeitung von Tracing-Telemetrie und damit von Daten. Nehmen aber die Anfragevolumina etwa infolge der Erweiterung der Umgebung um zusätzliche Microservices zu, steigt deren Menge schnell in exponentiellem Maße an.

Entsprechend komplexer und zudem kostenintensiver wird in diesem Zuge auch die Übermittlung der Tracing-Aktivität. Als Mittel dagegen setzen viele Unternehmen auf Daten-Sampling – was durchaus probat ist, solange die entsprechende Datenauswahl repräsentativ für den Gesamtbestand an Daten ist.

Gewährleisten lässt sich dies abhängig von den Monitoring-Anforderungen der jeweiligen Anwendung entweder mittels head- oder tailbasiertem Sampling. Wichtig ist daher, dass in dieser Hinsicht Flexibilität bei der Wahl besteht.

Headbasiertes Sampling: Für Traces mit Effizienz

Beim headbasierten Sampling wird die Auswahl der Span-Daten zum Zeitpunkt der Verarbeitung des Root Span gebildet. Als erster Span innerhalb einer Trace ermöglicht dieser sowohl die Identifikation der Services, die an einer Transaktion beteiligt sind, als auch die Analyse der dabei erfolgten Vorgänge. Das Sampling selbst erfolgt in der Regel innerhalb des Agents zur Erfassung der Trace-Telemetrie, der die Datenauswahl nach dem Zufallsprinzip bestimmt. Da die Sampling-Entscheidungen dabei jedoch noch vor Abschluss der Traces erfolgen, lässt sich nicht bestimmen, bei welcher Trace womöglich ein Problem vorliegt. So wäre es möglich, dass Traces aus dem Raster fallen, bei denen Prozesse ungewöhnlich langsam ausgeführt wurden oder Fehler aufgetreten sind.

Zu verachten ist headbasiertes Sampling aber dennoch nicht. Denn das Verfahren liefert einen Datenauszug, der statistisch relevant und somit für einen Gesamtüberblick der Abfragen innerhalb eines verteilten Systems äußerst nützlich ist. Für Anwendungen mit geringeren Transaktionsvolumina oder Umgebungen, in denen Monolith- und Microservices-Architekturen parallel betrieben werden, ist dies beispielsweise bereits ausreichend, um Traces mit Fehlern oder Latenzen effektiv aufzuspüren. Zudem ist das Verfahren äußerst effizient: Selbst aus immensen Trace-Volumina lassen sich damit Datenauszüge in Echtzeit erfassen, dies ganz ohne oder nur mit minimalen Auswirkungen auf die Anwendungs-Performance.

Argumente für headbasiertes Sampling

- Probates Mittel für Anwendungen mit geringerem Transaktionsdurchsatz
- Schnell und bei minimalem Aufwand einsatzbereit
- Angemessen für Umgebungen, in denen Monolith-Anwendungen gegenüber Microservices überwiegen
- Keine oder nur minimale Auswirkungen auf die Anwendungs-Performance
- Kosteneffiziente Lösung zur Übermittlung von Tracing-Daten an Dritte
- Sampling mit statistischer Relevanz, das adäquate Visibility für verteilte Systeme liefert

Nachteile von headbasiertem Sampling

- Auswahl der Trace-Daten nach Zufallsprinzip
- Trace-Sampling vor Durchlauf einer Trace durch die Gesamtheit der Services, dadurch keine direkte Bestimmung der Auftrittspunkte von Problemen möglich
- Traces mit Fehlern oder ungewöhnlicher hoher Latenz potenziell nicht im Sample-Datensatz erfasst

Tailbasiertes Sampling: Für maximalen Trace-Kontext

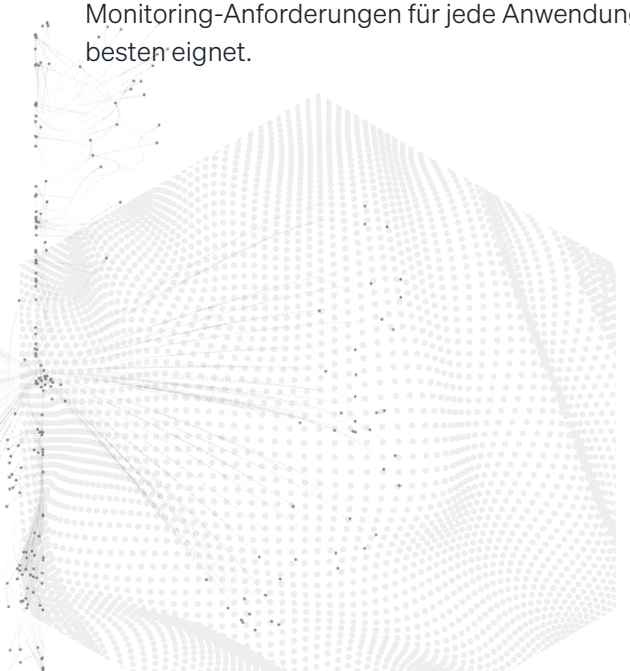
Geht es um die Problembehebung in hochgradig verteilten Systemen, in denen besonders hohe Transaktionsvolumina anfallen, fällt die Wahl eher auf tailbasiertes Sampling. Trace-Telemetrie lässt sich damit in seiner Gesamtheit erfassen – einschließlich aller Traces, die Fehler und ungewöhnliche Latenzwerte aufzeigen. Erreicht wird dies, da Details zu Traces bei diesem Verfahren erst nach ihrem Abschluss erfasst werden.

Dabei ist tailbasiertes Sampling mehr Pflicht als Kür, wenn im Troubleshooting-Prozess ein Höchstmaß an Granularität vonnöten ist.

Denn durch die Erfassung und Analyse ausnahmslos aller Spans – und damit jeder einzelnen Service-Interaktion – liefert tailbasiertes Sampling maximale Kontext-Klarheit. Unabdingbar ist dies etwa für Unternehmen, bei denen Downtime nicht nur Kosten in Millionenhöhe verursachen würde, sondern aufgrund enormer Traffic-Spitzen bei besonderen Events auch deutlich wahrscheinlicher wird.

So verzeichnen diese womöglich üblicherweise rund 3 Millionen Spans pro Minute, die bei einem Produkt-Launch jedoch direkt auf 300 Millionen hochschnellen. Bei Transaktionsvolumina dieser Größenordnung lässt sich mit headbasiertem Sampling schlicht kein effektives Troubleshooting mehr gewährleisten.

Nun trifft dies aber freilich nicht auf jede Trace zu. Daher gilt es, anhand von Use Case, Kosten-Nutzen-Analyse und Monitoring-Anforderungen für jede Anwendung einzeln abzuwägen, welche Sampling-Methode sich jeweils am besten eignet.



Argumente für tailbasiertes Sampling

- Observability und Analyse für 100 % aller Traces
- Sampling von Traces nach ihrem vollständigem Abschluss
- Schnellere Visualisierung von Traces, die Fehler oder ungewöhnlich lange Verarbeitungszeiten aufweisen

Nachteile von tailbasiertem Sampling

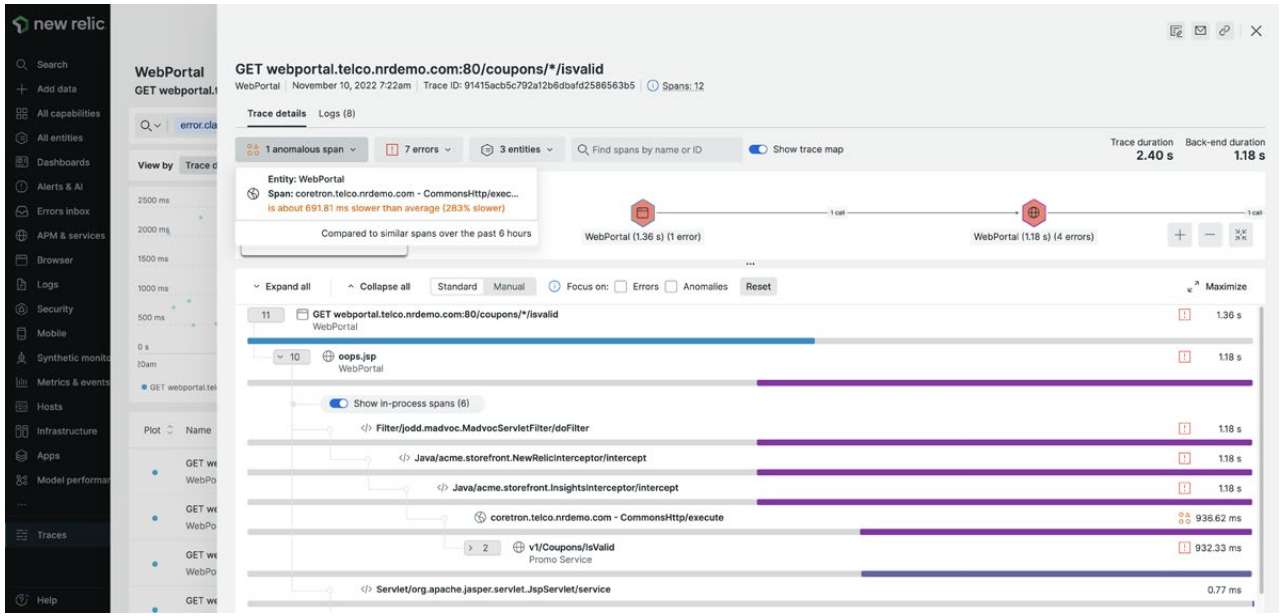
- Ausführung von Sampling-Software erfordert u. U. Gateways, Proxies und Satellites
- In manchen Fällen vergleichsweise aufwendige Verwaltung und Skalierung von Drittanbieter-Software
- Erhöhter Kostenaufwand aufgrund höherer Mengen an übermittelten und gespeicherten Daten

Klassisches headbasiertes Sampling (obere Abbildung) und tailbasiertes Sampling (Abbildung darunter)

Analyse und Visualisierung

Die Erfassung der Trace-Daten aus komplexen Architekturen allein bringt Software-Teams jedoch mehr als unnötigen Zeitaufwand. Sie müssen sie auch einfach und unkompliziert analysieren und visualisieren können. In einer umfassenden Observability-Plattform erhalten sie dazu neben sämtlicher Telemetrie auch alle zugehörigen Business-Daten zentral in einer Ansicht. Zudem gibt sie ihnen den Kontext an die Hand, der nötig ist, um relevante Einblicke aus den Daten zu ziehen und so mit adäquaten Maßnahmen zu reagieren und fundierte Entscheidungen zu treffen.

Für Distributed Traces ist dabei die Visualisierung in einer Baumstruktur ideal, dies einschließlich Child Spans, die auf den ihnen zugehörigen übergeordneten Span verweisen. Dadurch wird direkt ersichtlich, welche Spans innerhalb einer Trace hohe Latenz und/oder Fehler aufweisen. Wichtig sind zudem detaillierte Attribute zu diesen: Um welchen Fehler es sich handelt, an welchen Services es genau hakt. In dieser Form strukturiert, sehen Software-Teams alles direkt, können umgehend an den richtigen Stellschrauben drehen – eine Ansicht, die ihnen etwa Anbieter von Observability-Technologien wie New Relic an die Hand geben.



Distributed Tracing mit New Relic

Klare Methodik statt Management-Bürden

Mit der Fehlersuche verhält es sich in verteilten Systemen ohnehin schon wie mit der klassischen Nadel im Heuhaufen. Doch auch die Vorarbeit – die Instrumentierung der Systeme zum Tracing und zur Erfassung und Visualisierung der Daten – ist ein komplexes und potenziell enorm arbeitsintensives Unterfangen. Erleichterung bringen hier Software-as-a-Service-Angebote: Entsprechende SaaS-Lösungen decken von Deployment und Management bis hin zur Skalierung von Drittanbieter-Gateways und -Satellites zur Datenerfassung sämtliche Arbeitsschritte in einem Managed-Komplettservice ab.

So auch die Observability-Plattform von New Relic: Zur Instrumentierung von Anwendungen ist damit nicht mehr nötig als ein einzelner Agent – dies zudem für nahezu alle Programmiersprachen und -Frameworks. Unterstützt werden hierzu außerdem Open-Source-Tools und offene Standards. So etwa OpenTelemetry, der als einer der wichtigsten Open-Source-Standards für Instrumentierung und Telemetrie-Erfassung gilt.

Auch umfasst die New Relic Plattform tailbasiertes Sampling im Rahmen eines Managed-Komplettservice, der übergreifend über das gesamte verteilte System 100 % der Spans durch Observability und Analysen abdeckt. Zur einfacheren Problemerkennung und -behebung ist dabei auch die Visualisierung von Traces möglich, die Fehler und ungewöhnliche Latenzen aufweisen.

Überführt werden sämtliche Spans dabei in eine umfassende Komplettansicht: Zugehörige Metrics, Fehlerdetails und wichtige Traces sind darin komplett zentral einsehbar, für kritische Insights werden zudem die relevantesten Daten direkt in der Plattform gespeichert. Software-Teams erhalten damit einen völlig neuen Grad der Transparenz: Über detaillierte Metrics erhalten sie zunächst ein Bild davon, wie sich Downstream-Latenzen und Fehler auswirken. Ausgehend davon können sie über die hinterlegten Trace-Daten dann ins Detail gehen und die relevantesten Traces auswerten.

Distributed Tracing ist im Rahmen von New Relic APM komplett out of the box verfügbar. Dies mit niedriger Latenz und kosteneffizienter Datenübertragung von New Relic Agents, außerdem mit Instrumentierung innerhalb von Serverless-Funktionen sowie bei Bedarf auch beliebiger anderen Datenquellen etwa auch von Drittlösungen.

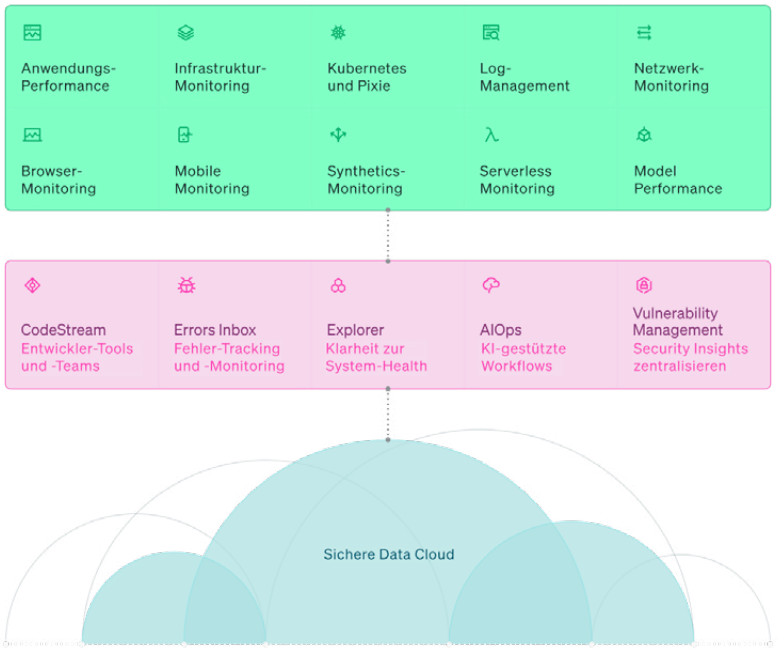
Konkrete Vorteile mit New Relic:

- Cloudbasierter Managed-Komplettservice mit On-Demand-Skalierung
- Observability und Analysen für 100 % aller Traces, übergreifend über sämtliche verteilten Systeme hinweg
- Visualisierung der relevantesten Traces mit Fehlern oder niedriger Latenz
- Kein Aufwand für Deployment, Management, Support und Skalierung von externen Gateways oder Satellites
- Umfassende Unterstützung für Open-Source-Instrumentierung und -Standards für Trace-Telemetrie
- Reduzierung der für Daten-Egress von Proximity- zu Cloud-Workloads anfallenden Kosten
- Effizientere Problembehebung
- Verkürzte Erkennungs- und Lösungszeit dank detaillierten, relevanten Traces
- Mehr Zeit für Engineering- und Dev-Teams für strategisch wichtigere Aufgaben wie z. B. Feature-Entwicklung

Sampling head- oder tailbasiert – oder einfach beides?

New Relic stellt Software-Teams in Sachen Distributed Tracing nicht vor die Wahl. Vielmehr stehen Optionen für head- und tailbasierte Sampling-Entscheidungen bis auf Anwendungsebene zur Verfügung. So können Sie für kritische Anwendungen, die Observability für ausnahmslos alle Traces erfordern, einfach tailbasiertes Sampling festlegen. Um die Verwaltung der zugehörigen Infrastruktur brauchen sie sich dabei nicht kümmern.

Denn New Relic ist der einzige Observability-Anbieter, der neben headbasiertem Sampling auch einen Managed-Komplettservice für tailbasiertes Sampling zur Verfügung stellt. Dies bedeutet Distributed Tracing bei weniger Verwaltungsaufwand für Software-Teams – und damit mehr Zeit für Innovation, die das Unternehmen im Wettbewerb voranbringt.



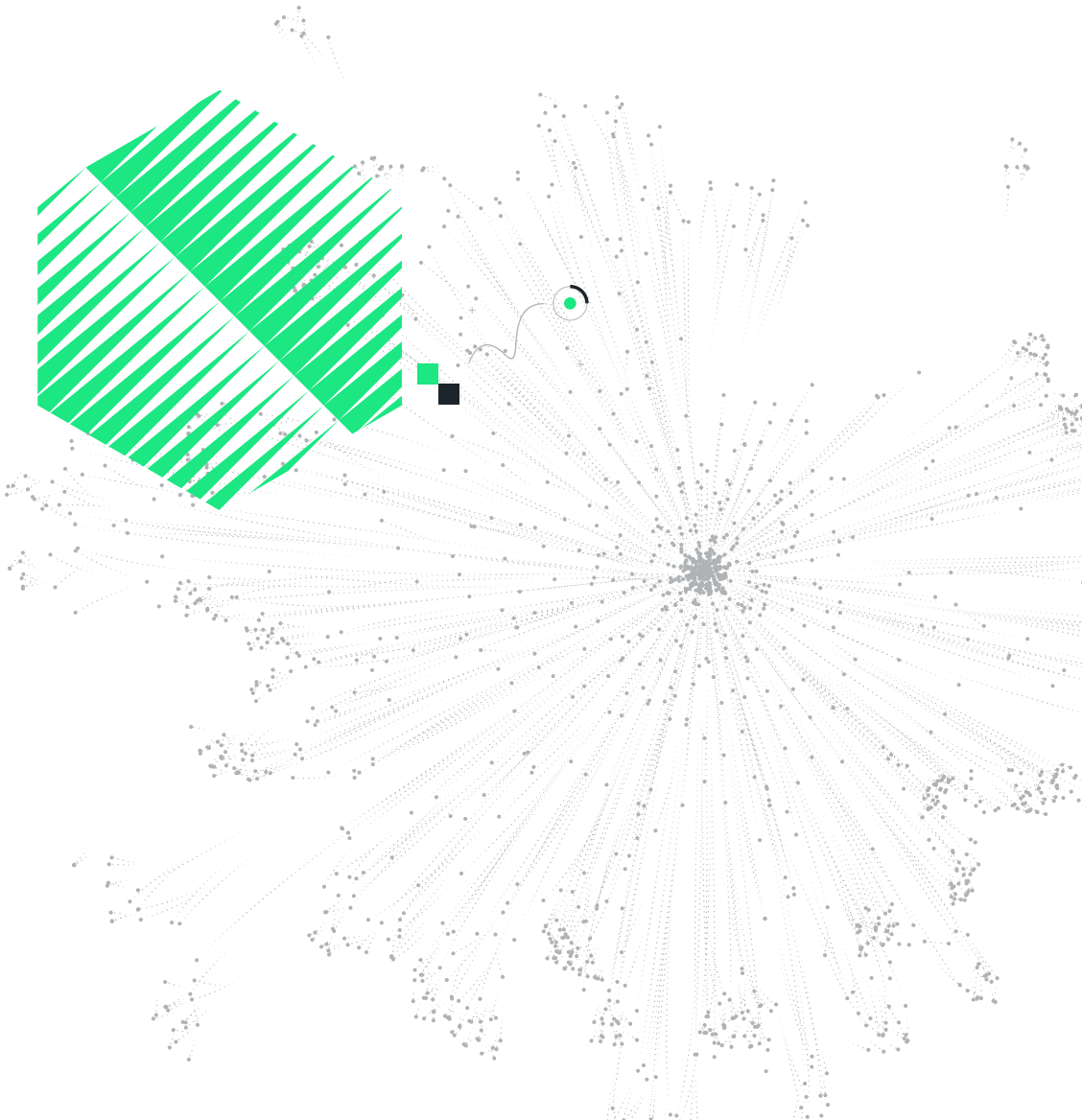
New Relic vereint von Log-Management, APM und Distributed Tracing bis zu Monitoring für Infrastruktur, Serverless, Mobile, Browser, Synthetics und Kubernetes alles in einer umfassenden Observability-Plattform.

Nächste Schritte

New Relic APM und Distributed Tracing können Sie noch heute für sich nutzen. Registrieren Sie sich dazu einfach hier. Das New Relic Einstiegskonto ist komplett kostenlos – dies mit 100 GB zur Datenerfassung, einer Komplettlizenz und unbegrenzten Basic-Lizenzen.

Sie haben bereits ein Konto für New Relic? Mit dem APM-Toolset in New Relic nutzen Sie Distributed Tracing so einfach wie unkompliziert. Alles, was Sie dafür benötigen, ist die aktuelle Version unseres APM-Agent – alle Optionen zur Einrichtung sind zudem umfassend dokumentiert.

Jetzt starten



Über New Relic

Als führender Anbieter von Observability-Technologien unterstützt New Relic die globale Engineering-Community mit einer datenfundierten Methodik für das gesamte Software Lifecycle – von der Konzept-Phase bis zur operativen Umsetzung. In New Relic erhalten Entwickler:innen die einzige Plattform zur Erfassung sämtlicher Telemetriedaten: Metrics, Events, Logs und Traces. Im Verbund mit umfassenden Analyse-Tools für den gesamten Stack leitet sie New Relic in kürzester Zeit von einer grundlegenden Situationsanalyse zur genauen Problemursache.

Mit dem branchenweit ersten klar planbaren verbrauchsbasierten Kostenmodell setzt New Relic auf absolute Transparenz in jeder Hinsicht und liefert der Engineering-Community so diverse Vorteile, von optimierter Cycle-Planung bis hin zu besseren Ergebnissen bei der Rate änderungsbedingter Ausfälle, der Release-Frequenzen und Lösungszeiten. Diese Möglichkeiten nutzen branchenführende Weltmarken wie auch Hypergrowth-Start-ups für bessere System-Uptime und -Stabilität, mehr Effizienz und optimale UX für ihre Endkund:innen – und für mehr Innovationschancen und Wachstum für sich.

new relic.