Engineer

Observability

ein System von außen anhand ausgesendeter Daten verstehen

Was es ist

Observability beschreibt, in welchem Maß ein Team von außen verstehen kann, was ein laufendes System tut: allein anhand seiner ausgesendeten Daten, ohne es zu öffnen oder vorher zu wissen, was schiefging. Die Einführung des OpenTelemetry-Projekts formuliert es als Fragen. Ein beobachtbares System lässt Fragen zu, ohne dass man seine inneren Abläufe kennen muss, auch solche, an die beim Schreiben der Software niemand dachte. Die Kurzform der Frage, die es beantworten können sollte, lautet dort: „Warum geschieht das?“

Observability ist damit eine Systemeigenschaft, kein Produkt. Ein Werkzeug kann Daten speichern und anzeigen. Zuerst müssen sie aber aus der Anwendung kommen. OpenTelemetry sagt es ausdrücklich: Ein beobachtbares System braucht instrumentierte Komponenten, also Code, der Signale aussendet. Daraus folgt ein praktischer Maßstab. Eine Anwendung gilt als richtig instrumentiert, wenn ihre vorhandenen Signale Entwicklern erlauben, ein Problem zu untersuchen, ohne zuerst die Instrumentierung zu ändern.

Diese Daten heißen Telemetrie. OpenTelemetry gliedert sie in Signale:

- **Traces** halten den Weg einer Anfrage durch das System fest. Ein Trace besteht aus Spans, jeweils einer einzelnen Arbeitseinheit mit Name, Beginn, Ende und Attributen wie HTTP-Methode und Route. Der erste Span ist die Wurzel und deckt die Anfrage vom Anfang bis zum Ende ab.
- **Metriken** sind zur Laufzeit erfasste und über die Zeit aggregierte Messungen: Fehlerrate, Anfragerate, Prozessorauslastung.
- **Logs** sind Ereignisaufzeichnungen mit Zeitstempel. Viele gehören zu keiner bestimmten Anfrage.
- **Baggage** sind mit einer Anfrage übertragene Kontextinformationen, die andere Signale verwenden können.

Die Signalseite führt auch Events, eine besondere Art von Log, und Profile auf. Profile zeichnen den Ressourcenverbrauch auf Codeebene auf. Beide werden als in Entwicklung oder im Vorschlagsstadium beschrieben.

Aus drei Datenhaufen wird durch Korrelation etwas, das ein Team befragen kann. OpenTelemetry überträgt Kontext zwischen Diensten. Ein Span in einem Dienst weiß dadurch, zu welchem Trace er gehört, und die IDs des aktiven Traces und des aktiven Spans können jedem Logeintrag mitgegeben werden. Jedes Signal trägt außerdem eine Ressource: Attribute zur Identifizierung des erzeugenden Objekts, etwa Dienstname, Host oder Container. Über diese gemeinsamen Schlüssel kann ein Team von einer Metrik zu den Anfragen dahinter und von einer Anfrage zu ihren Logzeilen wechseln.

Observability und Monitoring überschneiden sich, sind aber nicht dasselbe. Monitoring achtet auf vorher als wichtig festgelegte Zustände und alarmiert, sobald einer eintritt. Observability ist das, womit das Team nach dem Alarm arbeiten kann, wenn niemand die Ursache vorhergesehen hat.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Observability?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Monitoring und Alarme melden sich, solange ein Fehler noch klein ist; Kennzahlen zeigen, wie oft das gelingt.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Das erste Observability-Dashboard aufgebaut — der erste Einblick des Unternehmens in die eigenen Systeme

Das erste Observability-Dashboard aufgebaut Kontakt aufnehmen →

Warum es wichtig ist

Moderne Systeme scheitern auf schwer vorhersehbare Weise. Eine Anfrage kann ein Gateway, mehrere Dienste, eine Warteschlange und eine Datenbank durchlaufen, bevor sie beantwortet wird. Ein Fehler an einer Stelle kann sich als Langsamkeit an einer anderen zeigen. Die Einführung nennt verteiltes Tracing für solche Systeme unerlässlich: Ihre Probleme sind oft nicht deterministisch oder zu kompliziert, um sie auf dem Rechner eines Entwicklers nachzustellen. Lässt sich ein Fehler nicht reproduzieren, ist die währenddessen aufgezeichnete Telemetrie der einzige vorhandene Nachweis.

Dashboards und Alarme beantworten vorhergesehene Fragen. Das ist wertvoll und deckt bereits bekannte Fehler ab. Schwierig sind die neuen Vorfälle. Dann lautet die Frage nicht, ob die Fehlerrate stieg, sondern warum Anfragen aus einer Region über eine bestimmte Version zu einem Zahlungsanbieter zu scheitern begannen. Muss man dafür neuen Code schreiben, ihn bereitstellen und auf die Wiederkehr des Problems warten, dauert der Ausfall so lange wie dieser ganze Durchlauf.

Observability behält auch die Nutzer im Blick. Die Einführung definiert Zuverlässigkeit danach, ob der Dienst tut, was die Nutzer erwarten. Ihr Beispiel: Ein Laden kann ständig erreichbar und trotzdem unzuverlässig sein, wenn der Klick, mit dem schwarze Schuhe in den Warenkorb gelegt werden sollen, nicht immer genau das bewirkt. Eine Hostmetrik würde das nie zeigen. Telemetrie einzelner Anfragen kann es. Aus ihr entstehen Service Level Indicators: Messungen aus Nutzersicht, die Service Level Objectives dann mit den Anforderungen des Geschäfts verbinden.

Schließlich ändern sich die Kosten eines Vorfalls. Ein Team, das eine langsame Anfrage bis genau zu dem Aufruf verfolgen kann, der sie verzögert hat, verbringt seine Zeit mit dem Beheben statt mit Vermutungen. Die anschließende Nachanalyse kann Belege statt Erinnerungen zitieren.

Wie es in der Praxis aussieht

Nehmen wir einen erfundenen Onlineshop. Seit 14:00 dauert ein kleiner Teil der Bezahlvorgänge länger als zwei Sekunden. Nichts ist ausgefallen, kein Alarm wurde ausgelöst und dafür wurde kein Dashboard gebaut. So würde ein Team mit verknüpften Signalen vorgehen. Alle folgenden Namen, IDs und Zahlen sind für das Beispiel erfunden.

Der erste Schritt ist das Latenzhistogramm für den Bezahlvorgang, gemessen in Sekunden. Im OpenMetrics-Textformat kann eine Klasse nach einem `#` ein Exemplar tragen: einen Verweis auf Daten außerhalb der Metrik, meist die ID eines Traces aus dieser Klasse. Zwei Klassen lauten:

```
checkout_bucket{le="1.0"} 95
checkout_bucket{le="2.5"} 97 # {trace_id="3f1c9a77e2b04d18a6c5e0b9d4127f60"} 2.3
```

Der zweite Schritt folgt dieser Trace-ID. Der Trace zeigt Span für Span, wo die 2,3 Sekunden geblieben sind:

```
POST /checkout                      2.30 s   service checkout
  SELECT cart                       0.04 s   service checkout
  POST /authorise                   2.21 s   service payments
    event: provider timeout, retrying
  INSERT order                      0.03 s   service checkout
```

Der dritte Schritt holt die Logzeilen mit derselben Trace-ID. OpenTelemetrys Spezifikation für Logs in anderen Formaten als seinem eigenen Protokoll sieht vor, die IDs in JSON als Felder auf oberster Ebene mit den Namen `trace_id` und `span_id` festzuhalten, kleingeschrieben und hexadezimal kodiert:

```
{
    "timestamp": 1793110051.402,
    "severity": "WARN",
    "body": "provider timeout after 2000 ms, retrying",
    "provider": "acquirer-b",
    "trace_id": "3f1c9a77e2b04d18a6c5e0b9d4127f60",
    "span_id": "a91c04e7d2f3b658"
}
```

Drei Stationen, und die nicht vorhergesehene Frage hat eine Antwort: Bei einem von zwei Zahlungsanbietern traten Zeitüberschreitungen auf. Jeder dorthin geleitete Bezahlvorgang wartete auf einen erneuten Versuch. Alle jüngeren Traces lassen sich nun nach demselben Attribut, `provider`, gruppieren, um das Muster vor jeder Codeänderung zu bestätigen.

Nichts davon funktioniert, wenn die Dienste sich nicht identifizieren. OpenTelemetry-SDKs setzen das Ressourcenattribut `service.name` standardmäßig auf `unknown_service`. Die Dokumentation empfiehlt, es ausdrücklich zu setzen, im Code oder über eine Umgebungsvariable. Weitere Ressourcenattribute lassen sich genauso ergänzen:

```
env OTEL_SERVICE_NAME=checkout \
    OTEL_RESOURCE_ATTRIBUTES=deployment.environment.name=production \
    ./checkout
```

Auf dieser Website zeigen drei Portfolioeinträge den schrittweisen Aufbau von Observability. Einer beschreibt das erste Observability-Dashboard eines Unternehmens. Es erfasste CPU-, Speicher- und Datenträgermetriken entfernter Linux-Server über SSH. Eine zweite Fassung wurde später auf Grafana und Prometheus aufgebaut. Die Arbeit zur Senkung des Betriebsrisikos mit Grafana und Prometheus instrumentierte die wesentlichen Dienste und wählte aussagekräftige statt bloß gut aussehender Kennzahlen. Ein späterer Eintrag, Fehlerüberwachung und Tracing in eigener Hand, ergänzte OpenTelemetry-Tracing über OTLP. Eine langsame Anfrage lässt sich damit durch Frontend, API und Datenbank verfolgen, statt ihre Ursache zu erraten. Daneben bereinigt eine Fehlerpipeline die Nutzdaten vor der Speicherung.

Wo es schiefgeht

Kaufen statt einbauen. Ein neues Werkzeug ohne nützliche eingehende Daten ist ein leerer Raum. Observability entsteht durch Instrumentierung im Code und in den Bibliotheken. Planen Sie deshalb auch die Entwicklungszeit für Spans, Attribute und Logfelder ein, nicht nur die Lizenz.

Drei Signale, drei Inseln. Metriken in einem Werkzeug, Logs in einem anderen und Traces in einem dritten ohne gemeinsamen Schlüssel lassen das Team Zeitstempel von Bildschirm zu Bildschirm kopieren. Schreiben Sie Trace- und Span-IDs in jeden Logeintrag, hängen Sie Exemplare an Latenzhistogramme, soweit der Stack das unterstützt, und geben Sie allen Signalen dieselben Ressourcenattribute.

Jeder Dienst heißt unknown_service. Das ist der SDK-Standard. Eine Flotte mit diesem gemeinsamen Namen lässt sich nicht auseinanderhalten. Setzen Sie `OTEL_SERVICE_NAME` für jeden Dienst. Die Ressource wird beim Erzeugen der Tracer- und Metrikprovider angehängt und lässt sich danach nicht ändern. Sie muss deshalb beim Start stimmen.

Nur die Ränder sind instrumentiert. Zero-Code-Instrumentierung ist ein guter Anfang. Sie erfasst, was an den Rändern einer Anwendung geschieht: in ihren Bibliotheken und ihrer Laufzeitumgebung. Die Fragen im Vorfall betreffen meist das Geschäft: welche Kundenstufe, welcher Anbieter, welcher Feature-Flag-Wert. Dafür müssen Attribute im Code ergänzt werden. OpenTelemetry nennt codebasierte Instrumentierung eine Ergänzung zu Zero-Code. Beide können gleichzeitig laufen.

Instrumentieren während des Vorfalls. Brauchte die Antwort eine neue Logzeile und eine Bereitstellung, war das System nach dem eigenen Maßstab der Einführung nicht richtig instrumentiert. Machen Sie „Was mussten wir ergänzen?“ zur festen Frage jeder Vorfallnachbesprechung und behandeln Sie jede Antwort als Folgeaufgabe.

Unbegrenzte Details im falschen Signal. Nutzer-IDs, volle URLs und Anfragekörper gehören in Spans und Logs, nicht in Metriklabels, wo jeder neue Wert eine weitere Zeitreihe erzeugt. Der OpenMetrics-Standard begrenzt die Labelnamen und -werte eines Exemplars zusammen auf 128 Zeichen, gerade damit Exemplare nicht Span-Daten transportieren. Er erlaubt dem Empfänger, Exemplare ganz zu verwerfen. Prüfen Sie, ob die Verknüpfungen auf dem gesamten Weg vom Exporter bis zum Bildschirm erhalten bleiben, bevor Sie sich auf sie verlassen.

Telemetrie verrät Informationen. Spans und Logs halten Anfrageinhalte fest. Dazu können Adressen, Namen und Token gehören. Entscheiden Sie, welche Attribute aufgezeichnet werden dürfen, entfernen Sie den Rest vor der Speicherung und schützen Sie Telemetrie mit derselben Sorgfalt wie die von ihr beschriebene Datenbank.

Weiterführende Lektüre

- Einführung in Observability — die Definition des OpenTelemetry-Projekts mit Zuverlässigkeit, SLIs, SLOs, Spans und verteilten Traces.
- Signale — Traces, Metriken, Logs, Baggage und die noch in Entwicklung befindlichen Signale.
- Instrumentierung — codebasierte und Zero-Code-Instrumentierung sowie die Verknüpfung der Signale durch Kontextübertragung.
- Ressourcen — Ressourcenattribute, der Standard `unknown_service` und die zugehörigen Umgebungsvariablen.
- Trace-Kontext in Nicht-OTLP-Logformaten — die stabile Spezifikation für Trace- und Span-IDs in syslog, Klartext- und JSON-Logs.
- OpenMetrics 1.0 — der Standard für das Ausgabeformat mit Exemplaren und Empfehlungen zur Kardinalität.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Observability?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/observability/