Engineer

Prometheus

Datenbank, die Metriken abruft und speichert

Was es ist

Prometheus ist ein quelloffenes Monitoring-System mit einer eigenen Zeitreihendatenbank im Kern: Alle paar Sekunden fragt es die beobachteten Programme über HTTP nach ihren aktuellen Zahlen, speichert jeden Wert mit einem Zeitstempel und erlaubt, die gespeicherten Daten abzufragen, grafisch darzustellen und für Alarme zu verwenden. Der Name bezeichnet sowohl den Server allein als auch ebenso häufig die kleine Gruppe von Komponenten, die um ihn herum entstanden ist.

Entwickelt wurde es bei SoundCloud, wo die Arbeit 2012 begann; heute gehört es zur Cloud Native Computing Foundation. Es kam 2016 als zweites von der Stiftung aufgenommenes Projekt nach Kubernetes hinzu und erreichte 2018 den Status „Graduated“. Prometheus 3.0, angekündigt im November 2024, war die erste Hauptversion seit sieben Jahren. Der Code ist überwiegend in Go geschrieben und unter der Apache-2.0-Lizenz veröffentlicht.

Alles, was es speichert, ist eine Zeitreihe, bezeichnet durch eine Metrik und einen Satz von Labels: Schlüssel-Wert-Paare, die angeben, welche Instanz einer Sache gemessen wurde, etwa `api_http_requests_total{method="POST", handler="/messages"}`. Jeder Messpunkt dieser Reihe ist ein 64-Bit-Gleitkommawert oder ein natives Histogramm mit einem Zeitstempel in Millisekunden. Ein anderer Wert in irgendeinem Label ergibt eine andere Reihe. Das macht die Daten leicht nach Host, Route oder Statuscode auswertbar und das System zugleich leicht überlastbar.

Programme stellen ihre Zahlen an einem HTTP-Endpunkt bereit, gewöhnlich unter `/metrics`, entweder über eine in den Code einkompilierte Clientbibliothek oder über einen Exporter: einen kleinen Adapter, der ein System übersetzt, das Prometheus nicht direkt instrumentieren kann, etwa einen Linux-Host oder eine

Datenbank. Der Server findet seine Ziele über Diensterkennung oder eine statische Liste und ruft ihre Metriken nach einem Zeitplan ab. Ein Stapeljob, der vor dem nächsten Abruf endet, kann seine Daten stattdessen an ein separates Push-Gateway senden. Alarmregeln laufen auf dem Server; ausgelöste Alarme gehen an Alertmanager, der Duplikate entfernt, Alarme gruppiert, Stummschaltungen anwendet und sie an E-Mail, PagerDuty, OpsGenie und ähnliche Empfänger weiterleitet.

Als Alternativen werden meist Push-basierte Sammler in der Tradition von Graphite und StatsD, allgemeine Zeitreihendatenbanken wie InfluxDB und gehostete Metrikdienste betrachtet. Grafana ist die übliche Oberfläche davor. Prometheus speichert Zahlen statt Ereignisse; eine Protokollzeile oder ein Trace gehört daher woanders hin. Das Projekt sagt ausdrücklich, dass Zuverlässigkeit Vorrang vor Vollständigkeit hat: Eine Abrechnung pro Anfrage, bei der jede Zählung exakt stimmen muss, ist eine Aufgabe für ein anderes System.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Prometheus?“ 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

Bei Prometheus ist zu entscheiden, ob man eine Metrikdatenbank selbst betreibt oder mietet und wie groß sie im Eigenbetrieb sein muss.

Für den Betrieb fallen keine Lizenzgebühren an. Die Kosten bestehen aus einem Rechner und den Arbeitsstunden der Person, die ihn betreibt und instand hält. Ein gehosteter Dienst nimmt diese Arbeit ab und ersetzt sie durch eine Gebühr, die mit den eingesandten Daten wächst. Jedes neue Label wird damit zu einer Rechnungsposition. Für einige Dutzend Rechner ist ein selbst betriebener Server meist der billigere und einfachere Weg. Die Rechnung verschiebt sich, wenn die Zahl der Reihen in die Millionen geht und das Team lieber zahlt, als Personal dafür vorzuhalten.

Die Rechnung zum Speicherbedarf fällt günstiger aus, als die meisten erwarten. Laut Speicherdokumentation des Projekts liegt der Durchschnitt bei einem bis zwei Bytes je Messpunkt. Tausend Reihen, alle 15 Sekunden abgerufen und über die voreingestellten 15 Tage aufbewahrt, ergeben 86,4 Millionen Messpunkte, ungefähr 86–173 MB auf dem Datenträger. Der Platz auf dem Datenträger wird selten knapp. Der Engpass ist der Arbeitsspeicher: Der gerade gefüllte Block liegt dort, durch ein Write-ahead-Log gegen einen Absturz geschützt. Jede aktive Reihe benötigt dort etwas Platz, unabhängig davon, ob sie je jemand abfragt.

Jeder Server ist bewusst eigenständig. Er hängt von keinem Netzwerkspeicher ab und antwortet daher weiter, während die von ihm beobachtete Infrastruktur ausfällt, also genau dann, wenn er am meisten gebraucht wird. Der Preis dafür: Der lokale Speicher ist weder in einem Cluster organisiert noch repliziert. Hohe Verfügbarkeit bedeutet zwei identische Server, die dieselben Ziele abfragen. Wer mehr als einige Wochen Historie benötigt, muss eine Kopie mit `remote_write` an einen dafür ausgelegten Langzeitspeicher senden.

Der Abruf statt des Empfangs hat noch einen praktischen Vorteil. Weil der Server selbst fragt, bemerkt er, wenn ein Ziel nicht mehr antwortet, und hält dies fest, indem `up` auf null fällt. Außerdem kann ein Entwickler dieselbe Seite `/metrics` im Browser öffnen und genau das sehen, was der Server sieht. Widersprüche zwischen Grafik und Anwendung lassen sich dadurch schnell klären.

Wie es in der Praxis aussieht

In der Arbeit der Gründerin erscheint Prometheus als Sammler hinter einem Dashboard: einmal für eine Gruppe von Datenbankservern und einmal als zweite Fassung einer Ansicht, die das ganze Büro verfolgte.

Im Datenbankfall ging es um zehn virtuelle Server mit Ubuntu 20.04 LTS, PostgreSQL 14 und MS SQL Server 2019. Prometheus und Grafana verfolgten CPU, Arbeitsspeicher, Datenträger-I/O sowie Abfrageleistung in Echtzeit, neben automatisierten Sicherungen, Firewallregeln und eigenen Skripten zur Protokollanalyse. Über alle zehn Server lag die Verfügbarkeit bei 99,9 %. Datenbankhosts eignen sich gut für regelmäßige Abrufe: Die Zahlen, die Probleme ankündigen, etwa ein voller werdender Datenträger oder eine langsamer werdende Abfrage, verändern sich über Stunden. Messpunkte im Abstand von wenigen Sekunden zeigen diese Entwicklung, lange bevor ein Nutzer sie bemerkt.

Die Büroansicht zeigt den Schritt von einem selbst geschriebenen Sammler zu Prometheus. In einer früheren Rolle baute die Gründerin das erste Observability-Dashboard einer Organisation. Es las zunächst Messwerte zu CPU, RAM, Datenträgern, Temperatur und mehr von entfernten Linux-Servern über SSH aus und überwachte Docker-Container auf dieselbe Weise. Eine zweite Fassung wurde mit Prometheus und Grafana neu aufgebaut, ausgerichtet auf die mit den DevOps- und Entwicklungsteams gewählten Metriken: CPU-Auslastung, Speicherverbrauch, Dienstverfügbarkeit und API-Latenz, mit Alarmen bei Schwellenüberschreitungen. Mit dem Dashboard erkannten und behoben die Teams Störfälle 40 % schneller. SSH-Abfragen sind für eine Handvoll Rechner ein guter Anfang. Ein Sammler mit Labels und Abfragesprache lässt dieselbe Ansicht weiter wachsen.

Wo es schiefgeht

Der Speicherbedarf steigt nach einer Version ohne wichtige Änderungen. Jemand hat ein Label mit unbegrenzt vielen möglichen Werten ergänzt: eine Nutzer-ID, eine E-Mail-Adresse, einen vollständigen URL-Pfad oder eine Anfrage-ID. Jeder neue Wert erzeugte eine neue Reihe, und der Server hält nun Millionen, wo zuvor Tausende lagen. Die Benennungsrichtlinie warnt genau davor. Beschränken Sie Labelwerte auf kleine, bekannte Mengen, schreiben Sie Details einzelner Anfragen in Logs oder Traces und finden Sie die größten Verursacher mit einer Abfrage wie dieser:

```
topk(10, count by (__name__) ({__name__=~".+"}))
```

Ein Diagramm zeigt Einbrüche oder Spitzen, die nie auftraten. Ein Zähler steigt nur an und fällt beim Neustart des Prozesses auf null. Den Rohwert grafisch darzustellen oder von Hand zu subtrahieren macht aus jedem Neustart einen Einbruch oder Anstieg. `rate()` versteht Rücksetzungen. Berechnen Sie deshalb zuerst die Rate und summieren Sie danach, über ein Zeitfenster von mehreren Abrufintervallen:

```
sum by (job) (rate(http_requests_total{status=~"5.."}[5m]))
```

Eine Aufzeichnungsregel, die den Zähler vor der Ratenberechnung summiert, versteckt die Rücksetzungen in der Summe und bringt die falschen Einbrüche zurück.

Ein Dienst fällt aus, und sein Fehleralarm bleibt grün. Eine Regel wie eine Fehlerrate oberhalb einer Schwelle wird über die vorhandenen Reihen ausgewertet. Verschwindet das Ziel, verschwinden auch seine Reihen. Der Ausdruck liefert ein leeres Ergebnis, und der Alarm hebt sich im ungünstigsten Moment selbst auf. Alarmieren Sie für jedes Abrufziel bei `up == 0`, verwenden Sie `absent()` für jede Reihe, die immer vorhanden sein muss, und lassen Sie etwas außerhalb des Servers prüfen, ob der Server selbst läuft.

Die Zahlen des letzten Quartals fehlen, wenn jemand danach fragt. Lokal gespeicherte Daten werden standardmäßig 15 Tage aufbewahrt und liegen nur als eine einzige Kopie auf einem einzigen Datenträger vor. Als Fenster auf das jüngste Verhalten funktioniert diese Speicherung, als Archiv enttäuscht sie. Erhöhen Sie die Aufbewahrungsdauer mit Bedacht, senden Sie dauerhaft benötigte Daten über `remote_write` an Langzeitspeicher und halten Sie Zahlen, von denen Geld abhängt, in einer maßgeblichen Datenbank fest.

Weiterführende Lektüre

- Prometheus im Überblick — die eigene Beschreibung des Projekts, seiner Komponenten und der Fälle, für die es nicht passt.
- Datenmodell — Metriknamen, Labels und Messpunkte sowie der Grund, warum jeder Labelwert eine neue Reihe beginnt.
- Metriktypen — Zähler, Momentanwert, Histogramm und statistische Zusammenfassung und die Fragen, die jeder Typ beantworten kann.
- Grundlagen der Abfrage — PromQL-Selektoren, Bereichsvektoren und die Typen, die ein Ausdruck zurückgeben kann.
- Speicherung — Zweistundenblöcke, Write-ahead-Log, Aufbewahrung und die Remote-Write-Schnittstelle.
- Metriken und Labels benennen — Basiseinheiten, das Suffix `_total` und die Warnung vor Labels mit hoher 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 Prometheus?“ 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/prometheus/