Engineer

Zustandsprüfung

automatischer Test, ob ein Dienst funktioniert

Was es ist

Eine Zustandsprüfung ist ein kleiner, regelmäßig wiederholter Test, der einen laufenden Dienst fragt, ob er seine Aufgabe erfüllen kann. Dazu gehört eine Regel, was bei einer verneinenden Antwort geschieht. Der Test kann eine HTTP-Anfrage an einen eigens dafür vorgesehenen Pfad, ein Befehl im Container oder der Versuch sein, eine TCP-Verbindung zu einem Port aufzubauen. Erst die Regel macht aus einer Messung eine Steuerung: Eine fehlgeschlagene Prüfung kann einen Server aus der Verteilung nehmen, einen Container neu starten, ein Deployment stoppen oder jemanden wecken.

Docker baut die Idee ins Image ein. Die Anweisung `HEALTHCHECK` in einem Dockerfile benennt einen Befehl, den Docker in festen Abständen im Container ausführt. Exitstatus 0 bedeutet gesund, 1 bedeutet ungesund; 2 ist reserviert und darf nicht verwendet werden. Ein Container mit einer Prüfung beginnt im Zustand `starting`, wird nach der ersten bestandenen Prüfung `healthy` und nach einer festgelegten Anzahl aufeinanderfolgender Fehlschläge `unhealthy`. Standardmäßig wird alle 30 Sekunden geprüft, mit einem Timeout von 30 Sekunden und drei Wiederholungsversuchen. Überschreitet ein Lauf sein Timeout, zählt er als Fehlschlag und sein Prozess wird beendet. Eine optionale Startphase gibt einer langsamen Anwendung Zeit zum Hochfahren; Fehlschläge darin werden nicht gezählt. Ein Dockerfile enthält eine solche Anweisung. Bei mehreren gilt die letzte; `HEALTHCHECK NONE` schaltet eine vom Basisimage geerbte Prüfung aus.

Kubernetes teilt die Frage in drei Probes mit jeweils eigener Folge auf. Eine Startup-Probe fragt, ob die Anwendung fertig gestartet ist. Bis sie erfolgreich ist, werden die beiden anderen zurückgehalten. Schlägt sie weiter fehl, wird der Container beendet und gemäß seiner Neustartregel behandelt. Eine Liveness-Probe fragt, ob der Container neu gestartet werden sollte. Sie ist für Zustände wie einen Deadlock gedacht, bei dem der Prozess läuft, aber nicht vorankommt.

Eine Readiness-Probe fragt, ob der Container Verkehr erhalten soll. Schlägt sie fehl, wird nichts neu gestartet: Die Adresse des Pods wird aus den Endpunkten aller passenden Services entfernt. Anfragen gehen an die anderen Replikate, bis die Probe wieder erfolgreich ist.

Jede Probe nutzt einen von vier Mechanismen. Eine `httpGet` -Probe ist bei jedem Statuscode von 200 bis ausschließlich 400 erfolgreich. Eine `tcpSocket` -Probe ist erfolgreich, wenn der Port eine Verbindung annimmt, auch wenn diese sofort wieder geschlossen wird. Eine `grpc` -Probe ist erfolgreich, wenn die gRPC-Zustandsprüfung des Dienstes mit `SERVING` antwortet. Eine `exec` -Probe führt einen Befehl im Container aus und ist bei Exitstatus 0 erfolgreich. Ohne andere Vorgabe prüft Kubernetes alle 10 Sekunden, wartet eine Sekunde auf eine Antwort und handelt nach drei aufeinanderfolgenden Fehlschlägen.

Load Balancer ergänzen eine zweite Art. Die obigen Prüfungen sind aktiv: Etwas sendet eine Testanfrage, unabhängig davon, ob gerade Nutzer da sind. Eine passive Prüfung beobachtet stattdessen den wirklichen Verkehr. So arbeitet Open-Source-nginx: Endet eine Anfrage an einen Upstreamserver mit einem Fehler, markiert nginx ihn als ausgefallen und meidet ihn eine Zeit lang. Mit dem Standardwert 1 für `max_fails` genügt ein einziger gescheiterter Versuch. Die nginx-Dokumentation führt Anwendungszustandsprüfungen als Funktion der kostenpflichtigen NGINX-Plus-Abonnements auf. Eine passive Prüfung kostet nichts zusätzlich, erfährt aber erst durch die gescheiterte Anfrage eines Nutzers vom defekten Server. Über einen Server ohne Verkehr sagt sie nichts aus.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Zustandsprüfung?“ 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

Ein laufender Prozess ist noch kein funktionierender Dienst. Die Docker-Referenz nennt den typischen Fall: Ein Webserver steckt in einer Endlosschleife und nimmt keine neuen Verbindungen mehr an, obwohl sein Prozess weiterläuft. Alles, was nur das Vorhandensein des Prozesses überwacht, etwa eine systemd-Unit mit

Neustartregel, sieht kein Problem. Nur eine Frage durch denselben Eingang, den Nutzer verwenden, kann den Unterschied erkennen. Genau darum geht es bei einer Zustandsprüfung: Sie ersetzt „Läuft er?“ durch „Antwortet er?“.

Zwei Entscheidungen machen eine Prüfung nützlich oder gefährlich. Die erste ist, was sie prüft: eine in Code geschriebene Definition von „funktioniert“. Ein Handler, der 200 zurückgibt, ohne etwas anzusprechen, belegt, dass das Webframework läuft. Ein Handler, der zusätzlich die Datenbank abfragt, belegt auch deren Erreichbarkeit und schlägt fehl, sobald sie nicht erreichbar ist. Die zweite Entscheidung betrifft die Folge eines Fehlschlags und damit dessen Wirkungsradius. Ein Replikat aus der Verteilung zu nehmen ist günstig und umkehrbar. Ein Neustart verwirft seine laufende Arbeit und seine gefüllten Caches. Scheitert jedes Replikat im selben Moment an derselben Prüfung, verwandelt der Neustart aller Replikate eine langsame Abhängigkeit in einen vollständigen Ausfall.

Kubernetes warnt genau davor. Laut Dokumentation müssen Liveness-Probes so konfiguriert werden, dass sie tatsächlich einen Fehler anzeigen, von dem sich die Anwendung nicht selbst erholen kann. Eine falsche Implementierung kann Kaskadenausfälle verursachen: Container werden unter hoher Last neu gestartet, Clientanfragen schlagen fehl, und noch mehr Arbeit landet bei den verbleibenden Pods. Der folgende Rat lautet, die beiden Fragen zu trennen. Hängt eine Anwendung zwingend von Backenddiensten ab, sollte ihre Liveness-Probe erfolgreich sein, solange die Anwendung selbst gesund ist. Ihre Readiness-Probe prüft zusätzlich jedes erforderliche Backend. Verkehr meidet dann einen Pod, der nur mit Fehlern antworten könnte, und der Pod wird nicht wegen eines anderswo liegenden Fehlers beendet.

Zustandsprüfungen dienen auch dem Monitoring. Ein vom Load Balancer abgefragter Endpunkt lässt sich ebenso von außen abfragen. Die bei seinem Ausfall ausgelöste Warnung ist oft das erste Zeichen eines Vorfalls. Als Irakli Nadareishvili den Internet-Draft Health Check Response Format for HTTP APIs schrieb, gab es keinen Standard für die Antwort eines solchen Endpunkts. Die meisten Anwendungen verließen sich allein auf den HTTP-Statuscode oder auf ein eigenes Format. Der Entwurf schlug einen JSON-Nachrichtentext mit dem Medientyp `application/health+json` und einem verpflichtenden Feld `status` mit den Werten `pass`, `fail` oder `warn` vor. `pass` und `warn` gehören dabei zu einem Statuscode 2xx oder 3xx, `fail` zu 4xx oder 5xx. Seine letzte Fassung vom Oktober 2021 lief im April 2022 ab. Die IETF führt ihn als nicht gebilligt und ohne formellen Status. Als gemeinsames Vokabular ist er nützlich, als Standard nicht.

Wie es in der Praxis aussieht

In einem Dockerfile reist die Prüfung mit dem Image, das sie prüft. Hier fragt das eigene kleine Probeprogramm der Anwendung einen Pfad auf localhost ab und endet mit 0 oder 1:

```
HEALTHCHECK --interval=15s --timeout=3s --start-period=60s --retries=3 \
  CMD ["/app/bin/probe", "http://127.0.0.1:8080/livez"]
```

Docker führt die Probe alle 15 Sekunden aus und gibt jedem Lauf drei Sekunden. Fehlschläge in der ersten Minute zählen nicht, es sei denn, eine Prüfung war bereits erfolgreich. Auf Docker Engine 25.0 oder neuer laufen die Prüfungen in dieser Minute alle fünf Sekunden, dem Standard-Startintervall, damit ein schneller Start schnell bemerkt wird. Nach drei aufeinanderfolgenden Fehlschlägen wird der Container als `unhealthy` markiert. Die ersten 4.096 Bytes der Probeausgabe werden für `docker inspect` aufbewahrt, und jeder Zustandswechsel löst ein `health_status`-Ereignis aus. Die Referenz beschreibt einen Zustand, keine Aktion: Etwas anderes muss darauf reagieren.

In Kubernetes können die drei Probes verschiedene Pfade erhalten, damit jede ihre eigene Frage stellt:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # up to 2 minutes to start
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # about a minute before a restart
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` antwortet für den Prozess allein. `/readyz` prüft zusätzlich die Datenbank und alles andere, ohne das eine Anfrage nicht auskommt. Die Startup-Probe verwendet wie von Kubernetes empfohlen den Liveness-Pfad und lässt 24 × 5 Sekunden zu, bevor der Container aufgegeben wird. Nach ihrem Erfolg übernimmt Liveness mit einer langen und Readiness mit einer kurzen Zündschnur: Etwa zehn Sekunden mit Fehlschlägen nehmen den Pod aus dem Service, eine Minute startet ihn neu. Die Vorgaben ließen drei Fehlschläge im Abstand von zehn Sekunden zu. Das ist für eine Anwendung, die zwei Minuten zum Laden ihrer Daten braucht, zu wenig. Die eigene Regel der Dokumentation lautet: Ein Container, dessen Start üblicherweise länger dauert als die anfängliche Verzögerung plus failureThreshold × periodSeconds, braucht eine Startup-Probe.

Eine Zustandsprüfung lässt sich auch umkehren. Beim Totmannschalter für die Hostüberwachung vom September 2026 prüft sich ein Host mit 464 MB Arbeitsspeicher, auf dem eine Forge und ein Webserver laufen, alle fünfzehn Minuten mit zufälliger zeitlicher Streuung selbst. Ein kleines Skript misst verfügbaren Arbeitsspeicher und freien Plattenplatz. Es pingt einen externen Dienst nur an, wenn beide Werte oberhalb ihrer Schwellen liegen. Schweigen ist der Alarm. Ein Host ohne verbleibenden Arbeitsspeicher, ohne Netzwerk oder mit gescheitertem Start sendet dasselbe Signal wie ein lediglich ungesunder Host. Deshalb fiel die Wahl auf diese Form statt auf einen Agenten, der einen Status meldet. Die Prüfung liest verfügbaren statt freien Arbeitsspeicher, weil der Kernel ungenutzten Speicher mit Cache füllt. Sechs Variablennamen, die „free“ sagten, wurden korrigiert, damit niemand einen funktionierenden Alarm zu einem dauerhaft ausgelösten „korrigiert“. Ein empfangenes Signal bestätigt, dass das Skript lief und seine Prüfungen der Grenzwerte für verfügbaren Arbeitsspeicher und freien Plattenplatz bestanden wurden. Es bescheinigt weder den allgemeinen einwandfreien Zustand des Hosts noch, dass die Website ausgeliefert wird. Die Überwachung kann Warnzeichen sichtbar machen; wann diese auftreten, hängt jedoch von der Arbeitslast und den beobachteten Bedingungen ab.

Eine Prüfung beim Start beantwortet noch eine andere Frage. Die Umsetzung von E-Mail mit Failover und Zustellprotokollierung stammt aus der seit 2025 laufenden Tätigkeit. Der Versand läuft über drei Anbieter mit automatischem Failover. Davor steht eine durch ein Flag gesteuerte Startprüfung, die eine wirkliche Nachricht durch den gesamten Stack sendet. In der Entwicklung protokolliert ein Fehlschlag eine Warnung, und die Anwendung läuft weiter. In Staging und Produktion ist er fatal: Der Prozess beendet sich, statt einen Build

auszurollen, der keine E-Mails senden kann. Die Seite fasst das Ergebnis so zusammen: Eine ganze Klasse stiller Fehler wurde von „Ein Nutzer bemerkt es Tage später“ nach „Das Deployment stoppt“ verlagert.

Wo es schiefgeht

Die Prüfung belegt nur, dass der Prozess antwortet. Ein Handler, der 200 zurückgibt, ohne etwas anzusprechen, bleibt grün, während die Datenbank unerreichbar, das Laufwerk voll oder jede echte Anfrage fehlerhaft ist. Lassen Sie die Readiness-Prüfung die für eine Anfrage nötigen Teile mit wenig Aufwand tatsächlich überprüfen. Ergänzen Sie eine Prüfung von außen, die den Nutzerweg durch DNS, TLS und den Load Balancer verfolgt. Der kubelet sendet seine Probes direkt an die IP-Adresse des Pods und sieht davon nichts.

Liveness hängt von etwas außerhalb des Containers ab. Fragt die Liveness-Probe die Datenbank ab, lässt eine langsame Datenbank sie in allen Replikaten zugleich scheitern. Kubernetes startet alle Replikate gerade dann neu, wenn der Dienst ohnehin kämpft. Lassen Sie Liveness wie von Kubernetes empfohlen nur für den Prozess antworten. Prüfen Sie Abhängigkeiten in Readiness, wo ein Fehlschlag Verkehr umleitet, statt Container zu beenden.

Die Zeitwerte passen nicht zur Anwendung. Kubernetes wartet standardmäßig eine Sekunde auf eine Antwort. Wird ein Endpunkt unter Last langsamer, schlägt deshalb die Probe fehl, obwohl die Anwendung nur beschäftigt ist. Ein Container, der langsamer startet, als seine Liveness-Probe zulässt, wird immer wieder beendet, bevor er bereit ist. Messen Sie die Antwortzeit des Endpunkts unter Last und setzen Sie das Timeout darüber. Geben Sie langsam startenden Anwendungen eine Startup-Probe, die auf den langsamsten tatsächlichen Start ausgelegt ist.

Eine Weiterleitung besteht die Prüfung. Eine HTTP-Probe in Kubernetes zählt jeden Status von 200 bis 399 als Erfolg. Ein Pfad, der zur Anmeldung umleitet, wirkt deshalb gesund. Zeigt die Weiterleitung auf einen anderen Host, folgt kubelet ihr nicht und vermerkt die Probe mit einer Warnung als erfolgreich. Beim HTTPS-Schema überspringt kubelet außerdem die Zertifikatsprüfung, sodass selbst ein abgelaufenes Zertifikat eine Probe nie scheitern lässt. Richten Sie Probes auf einen Pfad, der direkt 200 und bei einem Fehler 5xx antwortet. Prüfen Sie Zertifikate von außen.

Der Zustandsendpunkt verrät zu vielen zu viel. Eine detaillierte Antwort mit sämtlichen Abhängigkeiten, Versionen und Hostnamen informiert Angreifer ebenso wie den Load Balancer. Die Sicherheitserwägungen des Internet-Drafts merken an, dass Zustandsendpunkte Authentifizierung und rollenbasierte Zugriffskontrolle benötigen können. Beschränken Sie die öffentliche Antwort auf einen Status und liefern Sie Details nur über einen internen Port oder an authentifizierte Aufrufer.

Die Prüfung kostet mehr als nötig. Kubernetes warnt, dass `exec` -Probes bei jedem Lauf Prozesse abspalten. Bei dicht belegten Nodes und kurzen Intervallen kann das zusätzliche CPU-Last auf dem Node verursachen. Eine Probe, die alle paar Sekunden auf jedem Replikat eine aufwendige Abfrage ausführt, erzeugt selbst Last. Bevorzugen Sie HTTP-, TCP- oder gRPC-Probes, halten Sie den Aufwand des Endpunkts gering und cachen Sie das Ergebnis einer aufwendigen Abhängigkeitsprüfung einige Sekunden, statt sie für jede Probe zu wiederholen.

Weiterführende Lektüre

- Dockerfile-Referenz: HEALTHCHECK — die Anweisung, ihre Optionen und Vorgaben sowie die Bedeutung jedes Exitstatus.
- Liveness-, Readiness- und Startup-Probes — die Aufgaben der Probetypen, die vier Prüfmechanismen und jedes Konfigurationsfeld mit seinem Standardwert.
- Liveness-, Readiness- und Startup-Probes konfigurieren — ausgearbeitete Beispiele, darunter eine Startup-Probe, die einer langsamen Anwendung fünf Minuten gibt.
- Antwortformat für Zustandsprüfungen von HTTP-APIs — der abgelaufene Internet-Draft hinter `application/health+json` mit seinen Zuständen pass, fail und warn.
- nginx als HTTP-Load-Balancer verwenden — passive Prüfungen im laufenden Verkehr bei Open-Source-nginx und die Einstellungen `max_fails` und `fail_timeout` .

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 eine Zustandsprüfung?“ 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/health-check/