Engineer

Hochverfügbarkeit

Weiterbetrieb eines Dienstes bei Ausfall einer Komponente

Was es ist

Hochverfügbarkeit ist die Eigenschaft eines Dienstes, weiterhin zu antworten, wenn einer der dahinterliegenden Rechner, Datenträger, Netzwerkpfade oder Prozesse ausfällt. Meist wird sie durch einen Cluster erreicht: mehrere Hosts, die jeweils den Dienst betreiben können, Software zu ihrer Überwachung und ein Verfahren, das die Arbeit von einem ausgefallenen auf einen gesunden Host verschiebt. Das ClusterLabs-Projekt, das den Open-Source-Clusterstack aus Pacemaker und Corosync pflegt, beschreibt, wie ein solcher Cluster Dienste auf mehreren Hosts verwaltet: vom Start und der Zustandsprüfung bis zur Wiederherstellung nach einem Ausfall, „um Ausfallzeiten zu minimieren und dabei die Datenintegrität zu bewahren“.

Die zweite Hälfte dieses Satzes ist ebenso wichtig wie die erste. Ein Dienst, der schnell mit falschen Daten zurückkommt, wurde nicht hochverfügbar gemacht. Er wurde gefährlich gemacht.

Die Dokumentation von PostgreSQL trennt zwei Gründe dafür, mehrere Datenbankserver gemeinsam zu betreiben. Der eine ist, bei Ausfall des Primärservers schnell einen zweiten übernehmen zu lassen; das nennt sie Hochverfügbarkeit. Der andere ist, dieselben Daten von mehreren Rechnern aus bereitzustellen; das nennt sie Lastverteilung. Beide nutzen oft dieselben Mechanismen, beantworten aber unterschiedliche Fragen: Hochverfügbarkeit soll einen Ausfall überstehen, Lastverteilung und Skalierbarkeit sollen mehr Arbeit bewältigen.

Verfügbarkeit selbst ist eine Zahl. Das traditionelle Maß bezieht sich auf die Zeit: den Anteil eines Zeitraums, in dem der Dienst genutzt werden konnte. Googles SRE-Buch erklärt, dass es bei weltweit verteilten Diensten, von denen fast immer

irgendwo ein Teil läuft, stattdessen den Anteil erfolgreich beantworteter Anfragen misst. Beide Maße werden als Prozentsatz geschrieben und als Anzahl von Neunen bezeichnet.

Jeder hochverfügbare Entwurf besteht aus denselben wenigen Teilen:

- **Redundanz.** Ersatz für jeden Teil, dessen Ausfall den Dienst stoppen würde: ein zweiter Host, eine Standby-Datenbank, ein zweiter Netzwerkpfad. Pacemaker unterstützt Aktiv/Passiv-Paare, N+1-Anordnungen mit einer gemeinsamen Reserve für mehrere aktive Nodes und weitere Aufbauten.
- **Erkennung.** Etwas muss den Ausfall bemerken. Das Failover-Kapitel von PostgreSQL beschreibt einen Heartbeat zwischen Primärserver und Standby, der laufend die Verbindung und die Funktionsfähigkeit des Primärservers prüft.
- **Failover.** Die Arbeit verlagern: den Standby hochstufen, die IP-Adresse des Dienstes verschieben, den Prozess auf einem anderen Host starten.
- **Fencing.** Sicherstellen, dass der ausgefallene Node tatsächlich angehalten ist, bevor seine Arbeit anderswo beginnt.
- **Die Reserve wiederherstellen.** Nach einem Failover läuft das System ohne Redundanz, bis eine neue Reserve bereitsteht.

Hochverfügbarkeit ist weder Backup noch Notfallwiederherstellung. Sie schützt vor dem Ausfall eines Teils, während der Standort als Ganzes weiterarbeitet. Ein Backup schützt vor Datenverlust oder Datenbeschädigung, einschließlich einer Beschädigung, die ein Cluster gewissenhaft auf jeden Node kopiert. Notfallwiederherstellung ist der Plan für den Verlust des gesamten Standorts.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Hochverfügbarkeit?“ 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

Jeder Dienst hat Teile, die ausfallen werden. Hardware geht kaputt, Kernel-Panics treten auf, eine virtuelle Maschine wird versehentlich gelöscht, eine Netzwerkpartition trennt einen gesunden Node vom übrigen Cluster. Die Kubernetes-Dokumentation führt das als unfreiwillige Unterbrechungen auf. Daneben stehen geplante freiwillige Unterbrechungen, etwa die Arbeitslasten für eine Reparatur oder ein Upgrade von einem Node zu entfernen. Ohne Redundanz bedeutet jeder dieser Fälle einen Ausfall, auch die geplanten: Einen einzelnen Server zu patchen erfordert, ihn herunterzufahren.

Dieser zweite Punkt ist oft der erste Vorteil, den ein Team bemerkt. Das Failover-Kapitel von PostgreSQL nennt den regelmäßigen Wechsel zwischen Primärserver und Standby nützlich: Jede Maschine kann so nacheinander zur Wartung abgeschaltet werden, und jeder Wechsel prüft, ob der Failover-Mechanismus im Bedarfsfall funktioniert.

Die Neunen lassen sich in Zeit übersetzen. Für ein Jahr mit 365 Tagen (525.600 Minuten) und einen Monat mit 30 Tagen (43.200 Minuten) lautet die Rechnung:

```
Zulässige Ausfallzeit:
Verfügbarkeit   pro Jahr              pro 30-Tage-Monat
99%             3.65 Tage             7.2 Stunden
99.9%           8.76 Stunden          43.2 Minuten
99.95%          4.38 Stunden          21.6 Minuten
99.99%          52.56 Minuten         4.32 Minuten
99.999%         etwa 5.26 Minuten     etwa 26 Sekunden
```

Jede zusätzliche Neun verkleinert das erlaubte Zeitfenster um den Faktor zehn; die Kosten steigen schneller. Das SRE-Buch warnt, dass ein weiterer Zuverlässigkeitsschritt hundertmal so viel kosten kann wie der vorherige. Ab einem Punkt können Nutzer den Unterschied zudem nicht mehr erkennen: Wer ein selbst nur zu 99 % zuverlässiges Telefon nutzt, bemerkt nicht, ob der dahinterliegende Dienst mit 99,99 % oder 99,999 % läuft. Deshalb behandelt das Buch ein Verfügbarkeitsziel zugleich als Unter- und Obergrenze. Ein Dienst, der viel verfügbarer ist als nötig, hat Geld und Entwicklungszeit verbraucht, die anderswo hätten eingesetzt werden können.

Darum ist das Ziel eine Geschäftsentscheidung, die für jeden Dienst gesondert getroffen wird. Ein Service Level Objective hält die Zahl fest; Hochverfügbarkeit ist die technische Arbeit, die sie erreichbar macht.

Wie es in der Praxis aussieht

Nehmen wir einen erfundenen Buchungsdienst mit drei Schichten: Webschicht, Anwendungsschicht und PostgreSQL-Datenbank, jede auf einem eigenen Host. Alle folgenden Zahlen sind für das Beispiel erfunden. Ist jede Schicht zu 99,9 % verfügbar und benötigt der Dienst alle drei, werden ihre Verfügbarkeiten miteinander multipliziert. Eine Schicht zu verdoppeln wirkt umgekehrt, weil dann beide Kopien gleichzeitig ausfallen müssen:

```
Drei Schichten in Reihe, jede zu 99.9% verfügbar:
  0.999 × 0.999 × 0.999 ≈ 0.997      etwa 26 Stunden Ausfall pro Jahr

Eine Schicht doppelt, wenn ihre beiden Kopien unabhängig ausfallen:
  1 − 0.001 × 0.001 = 0.999999       etwa eine halbe Minute pro Jahr
```

Die zweite Rechnung steht und fällt mit einem Wort: unabhängig. Zwei Kopien auf demselben Host, im selben Rack, hinter demselben Switch oder in derselben Cloudzone teilen ein Schicksal; die Rechnung gilt dann nicht mehr. Die Formel setzt außerdem voraus, dass der Wechsel zwischen den Kopien sofort geschieht und immer funktioniert. Das können nur Tests zeigen.

In Kubernetes dienen dazu Topology Spread Constraints. Das eigene Beispiel der Dokumentation ist eine Arbeitslast, die auf zwei Pods schrumpfen kann. Beide sollen nicht auf demselben Node liegen, dessen Ausfall die ganze Arbeitslast unerreichbar machen würde. Eine Vorgabe in der Pod-Vorlage verteilt die Replikate über Zonen:

```
topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
          matchLabels:
              app: booking
```

Mit `DoNotSchedule` lässt der Scheduler einen Pod unverplant, statt ihn so zu platzieren, dass sich die Anzahl der Pods zwischen den Zonen um mehr als eins unterscheidet. Geplante Wartung braucht eine zweite Absicherung. Ein PodDisruptionBudget begrenzt, wie viele Replikate einer Anwendung durch freiwillige Unterbrechungen zugleich ausfallen dürfen. `kubectl drain` beachtet das Budget, weil es die Eviction API um das Entfernen jedes Pods bittet. Pods direkt zu löschen umgeht es:

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
    name: booking-pdb
spec:
    maxUnavailable: 1
    selector:
        matchLabels:
            app: booking
```

Die Dokumentation empfiehlt `maxUnavailable` statt `minAvailable`, weil es Änderungen an der Anzahl der Replikate automatisch folgt.

Die Datenbank ist die schwierigste Schicht, weil sie den Zustand speichert. PostgreSQL kann einen Hot Standby betreiben, der nur lesende Abfragen annimmt, oder einen Warm Standby, der erst nach seiner Hochstufung Verbindungen akzeptiert. Zwischen synchroner und asynchroner Replikation zu wählen heißt zu entscheiden, was ein Failover verlieren darf. Bei synchroner Replikation wartet die Commit-Bestätigung auf die konfigurierte Anzahl synchroner Standbys auf der konfigurierten Bestätigungsstufe. Die Einstellungen `synchronous_standby_names` und `synchronous_commit` legen diese Bedingungen fest. Für ein Failover muss ein Standby gewählt werden, der die bestätigten Daten besitzt; ein asynchroner Standby kann noch zurückliegen. Der Preis der ersten Variante ist Geschwindigkeit: Die Dokumentation merkt an, dass langsame Netzwerkverbindungen bei einem vollständig synchronen Aufbau mehr als die Hälfte der Leistung kosten können.

PostgreSQL entscheidet nicht, wann ein Failover erfolgt. Laut Dokumentation liefert es keine Systemsoftware, die den Ausfall des Primärservers erkennt und den Standby benachrichtigt. Diese Aufgabe gehört wie das Verschieben der IP-Adresse zu externer Clustersoftware. Die Hochstufung selbst ist ein einzelner Schritt: `pg_ctl promote` oder die Funktion `pg_promote()`. Danach kommt der Teil, den Pläne vergessen. Ohne den alten Primärserver läuft der Cluster auf einem

Server weiter. Die Dokumentation bezeichnet dies als degenerierten Zustand. Er hält an, bis ein neuer Standby aufgebaut ist. Das Werkzeug `pg_rewind` kann das bei großen Clustern beschleunigen.

Auf dieser Website zeigt der Eintrag über E-Mail mit Failover und Zustellprotokollierung dasselbe Muster für eine externe Abhängigkeit. Drei E-Mail-Anbieter stehen hinter einer Schnittstelle. Der Wechsel zwischen ihnen erfolgt automatisch statt durch eine Konfigurationsänderung unter Druck. Eine Startprüfung sendet eine echte Nachricht, sodass Staging und Produktion ohne funktionierenden Anbieter den Start verweigern. Der Eintrag berichtet, dass ein schlechter Nachmittag eines Anbieters nun einen eingeschränkten Pfad statt eines Ausfalls bedeutet. Der Preis sind drei funktionsfähig zu haltende Integrationen. Wie verfügbar ein System sein muss, ist eine eigene Entscheidung: Die Seite zu Service Levels erklärt, dass die Verfügbarkeitszahl für jedes System gesondert vereinbart wird. Ein über Nacht erstellter Bericht und ein Zahlungsdienst brauchen unterschiedliche Werte.

Wo es schiefgeht

Redundanz mit gemeinsamem Schicksal. Zwei Kopien auf einem Host, an einem Netzteil, hinter einem Switch oder in einer Cloudzone fallen gemeinsam aus. Verfolgen Sie jede Reserve zu ihren Abhängigkeiten zurück. Verteilen Sie Replikate bewusst über Ausfalldomänen, statt es dem Zufall zu überlassen.

Split Brain. Verlieren Clusternodes den Kontakt, laufen aber weiter, kann jede Seite die andere für tot erklären und den Dienst selbst starten. Pacemaker beschreibt die Folge ausdrücklich: Dieselbe IP-Adresse auf beiden Hosts zu aktivieren macht das Ziel eingehender Pakete unvorhersehbar. Bei einer Datenbank drohen beschädigte oder auseinanderlaufende Daten. Fencing, auch STONITH genannt (Shoot The Other Node In The Head), ist die Absicherung: Ein Fence-Gerät, etwa ein vom Cluster gesteuerter Stromschalter, trennt den ausgefallenen Node ab, bevor dessen Arbeit anderswo beginnt. Laut Dokumentation kann ein Cluster den Betrieb bei einem nicht antwortenden Node ohne Fencing nicht sicher wiederherstellen. Das Fence-Gerät darf nicht vom eigenen Netzwerk des Clusters abhängen; sonst nimmt der Verlust dieses Netzwerks auch die Möglichkeit zum Fencing.

Ein Failover, das niemand ausprobiert hat. Ein nie hochgestufter Standby ist Hoffnung, kein Plan. Wechseln Sie die Rollen nach einem Zeitplan, wie PostgreSQL empfiehlt. Schreiben Sie das Verfahren auf und üben Sie es, so wie

Chaos Engineering andere Ausfälle übt.

Der degenerierte Zustand wird vergessen. Nach einem Failover gibt es keine Reserve. Dauert ihr Wiederaufbau lange oder erfolgt er manuell, wird der nächste noch so kleine Fehler zum Ausfall. Betrachten Sie die Wiederherstellung der Redundanz als Teil des Vorfalls, nicht als spätere Aufgabe.

Asynchrone Replikation wird für verlustfrei gehalten. Sie kann die letzten vor dem Ausfall bestätigten Transaktionen verlieren. Entscheiden Sie je Datenbank, ob das hinnehmbar ist. Machen Sie die Abwägung für die Datenverantwortlichen sichtbar.

Replikation wird mit einem Backup verwechselt. Ein Standby kopiert ein versehentliches `DELETE` genauso gewissenhaft wie einen richtigen Schreibvorgang. Hochverfügbarkeit hält den Dienst am Laufen; nur ein Backup lässt ein Team zum früheren Datenstand zurückkehren.

Zwei Nodes und kein Stichentscheid. Laut Pacemaker sieht ein durch Stromausfall verlorener Node genauso aus wie eine unterbrochene Netzwerkverbindung zu ihm. Bei zwei Mitgliedern sieht jede Seite genau das. Kubernetes verlangt im gestapelten Aufbau mindestens drei Control-Plane-Nodes. Die PostgreSQL-Dokumentation nennt einen dritten Witness-Server, der manche falschen Failover verhindern kann. Sie warnt zugleich, dass sich die zusätzliche Komplexität nur bei sorgfältiger Einrichtung und strengen Tests lohnt.

Für Neunen zahlen, die niemand bemerkt. Jede zusätzliche Neun kostet mehr als die vorherige. Nutzer erreichen den Dienst über Netzwerke und Geräte, die weniger zuverlässig sind als das Ziel. Wählen Sie die Zahl nach dem Bedarf der Nutzer und des Unternehmens und richten Sie den technischen Aufbau anschließend an diesem Ziel aus.

Weiterführende Lektüre

- ClusterLabs — der Open-Source-Clusterstack für Hochverfügbarkeit: was ein Cluster koordiniert und warum er Fencing braucht.
- Hochverfügbarkeit, Lastverteilung und Replikation — PostgreSQLs Kapitel zu Primärservern und Standbys sowie synchronen und asynchronen Entwürfen und ihren Kosten.

- Failover — PostgreSQL zu Heartbeats, Witness-Servern, STONITH, Hochstufung und degeneriertem Zustand.
- Fencing — Pacemaker Explained zu Split Brain, Fence-Geräten sowie Fencing und Quorum.
- Risiken akzeptieren — das SRE-Buch von Google zur Messung der Verfügbarkeit, den Kosten jeder Neun und der Wahl eines Ziels.
- Unterbrechungen — freiwillige und unfreiwillige Unterbrechungen in Kubernetes und ihre Begrenzung durch ein PodDisruptionBudget.

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 Hochverfügbarkeit?“ 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/high-availability/