**Engineer**

# Chaos Engineering

kontrollierte Experimente mit Fehlern im laufenden System

## Was es ist

Chaos Engineering bedeutet, reale Fehler absichtlich und kontrolliert in ein laufendes System einzubringen, um herauszufinden, ob es damit zurechtkommt. Die Principles of Chaos Engineering, das kurze Dokument, auf das die meisten Praktiker verweisen, definieren es als „die Disziplin, an einem System zu experimentieren, um Vertrauen aufzubauen“, dass es turbulenten Bedingungen in Produktion standhält. Entscheidend ist das Experimentieren. Ein Chaos-Experiment prüft keine Funktion gegen eine Spezifikation. Es stellt eine Frage über das gesamte System und akzeptiert die Antwort, die zurückkommt.

Die Begründung beginnt bei verteilten Systemen. Jeder Dienst darin kann korrekt arbeiten, und die Wechselwirkungen können dennoch Ergebnisse hervorbringen, die niemand vorhergesehen hat. Seltene reale Ereignisse wie ein abgestürzter Server oder eine unterbrochene Netzwerkverbindung verschärfen das. Die Principles nennen die dort verborgenen Schwächen: falsche Fallback-Einstellungen bei nicht verfügbarem Dienst, schlechte Timeout-Einstellungen, die Wiederholungsversuche verstärken, von ihren Aufrufern überlastete Abhängigkeiten und eine ausgefallene Komponente, die mehrere andere mitreißt. Nichts davon zeigt sich, wenn jeder Teil einzeln getestet wird.

Ein Experiment besteht nach den Principles aus vier Schritten:

- Definieren Sie den Normalzustand als messbare Ausgabe des Systems, die normales Verhalten anzeigt, etwa Durchsatz, Fehlerrate oder ein Latenzperzentil.
- Stellen Sie die Hypothese auf, dass der Normalzustand sowohl in einer Kontrollgruppe als auch in einer Versuchsgruppe bestehen bleibt.
- Führen Sie Variablen ein, die reale Ereignisse abbilden: abstürzende Server, ausfallende Platten, unterbrochene Netzwerkverbindungen.

- Vergleichen Sie die gemessenen Ausgaben der Gruppen und suchen Sie nach Belegen, die der Vorhersage widersprechen.

  Je schwerer sich der Normalzustand stören lässt, desto mehr Vertrauen kann das Team in das System haben. Wird die Hypothese widerlegt, hat das Team eine Schwäche gefunden, solange sie noch klein ist und Menschen zusehen, statt bei einem nächtlichen Ausfall.

Das bekannteste Werkzeug ist Netflix' Chaos Monkey. Es beendet zufällig ausgewählte Instanzen virtueller Maschinen und Container in Produktion. Seine README nennt es als Beispiel eines Werkzeugs, das den Principles folgt. Sein erklärtes Ziel ist, Entwickler zum Bau von Diensten zu bewegen, die gegen Instanzausfälle widerstandsfähig sind.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Chaos Engineering?" 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

Redundanz, Failover und Fallbacks sind Versprechen darüber, was bei einem Defekt geschieht. Ein nie geprüftes Versprechen ist eine Hypothese. Ein Standby, der noch keinen Verkehr übernommen hat, eine Wiederholungsstrategie, die noch keine langsame Abhängigkeit erlebt hat, ein noch nie ausgelöster Alarm: Alles kann funktionieren. Der erste echte Ausfall ist eine teure Art, das herauszufinden. Chaos Engineering verlegt diese Entdeckung auf einen vom Team gewählten Zeitpunkt.

Außerdem prüft es das System statt der Einzelteile. Unit- und Integrationstests prüfen Verhalten, an das bereits jemand gedacht hat. Die Principles benennen den Unterschied direkt: Chaos-Experimente prüfen, dass das System funktioniert, statt zu validieren, wie es funktioniert. Deshalb betrachten sie die von Nutzern bemerkbare Ausgabe, etwa Fehlerrate und Latenz, und nicht die Interna einer einzelnen Komponente.

Die Belege liegen in Produktion. Systeme verhalten sich je nach Umgebung und Verkehrsmuster unterschiedlich. Laut den Principles lässt sich der Anfragepfad nur durch Stichproben aus echtem Verkehr zuverlässig erfassen. Googles Buch Site Reliability Engineering kommt aus der anderen Richtung zum selben Schluss: Synthetische Last zeigt womöglich nicht, wo Komponenten wirklich versagen. Sobald ein Dienst gegen Überlastung geschützt ist, empfiehlt das Buch deshalb Fehlertests an einem begrenzten Teil des laufenden Dienstes. Reservekapazität soll bereitstehen, falls die Schutzmaßnahmen nicht funktionieren und jemand von Hand umschalten muss.

Schließlich ändert das die Anreize. Die README von Chaos Monkey sagt es klar: Werden Entwickler häufiger Fehlern ausgesetzt, haben sie einen Grund, widerstandsfähige Dienste zu bauen. Ein Team, dessen Instanzen an jedem Arbeitstag beendet werden können, plant das von Anfang an ein, statt die Lücke an dem Tag zu entdecken, an dem eine Maschine ohne Vorwarnung stirbt.

## Wie es in der Praxis aussieht

Ein Chaos-Experiment wird vor der Ausführung aufgeschrieben, ähnlich einem Testplan. Hier ist eines für einen erfundenen Onlineshop. Jede Zahl ist für das Beispiel erfunden:

```
Experiment:   the payment provider slows down
Steady state: checkout success rate and p99 checkout latency, per 5 min
Hypothesis:   with 300 ms added to calls to the payment provider, the
              experimental group's success rate stays within 0.2 points
              of the control group's
Groups:       2% of checkout traffic in each, assigned at random
Variable:     300 ms of latency on the payment client, nothing else
Abort if:     success rate gap above 1 point for 2 minutes in a row
Window:       30 min, working hours, on-call engineer told in advance
Result:       hypothesis disproved; retries tripled the calls to the
              provider and p99 latency passed the 1 s timeout
Follow-up:    a retry budget per request; rerun the experiment after
```

Eine widerlegte Hypothese ist hier das nützliche Ergebnis. Die Schwäche wird auf einem Anteil von 2 % des Verkehrs entdeckt, der nur für wenige Minuten dem Experiment ausgesetzt ist; die Abbruchbedingung steht bereits fest. Darauf folgt eine technische Änderung statt einer Entschuldigung.

Die Variable selbst wird häufig mit gewöhnlichen Werkzeugen eingebracht. Unter Linux emuliert die Warteschlangendisziplin netem Eigenschaften realer Netze, darunter Verzögerung, Verlust, Duplizierung und Beschädigung. Zum Beispiel:

```
# experiment 1: add 100 ms of delay to every packet leaving eth0
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root

# experiment 2: drop one packet in a thousand, at random
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

Die Zeile mit `del` ist die wichtige. Sie gehört in einen Aufräumschritt, der unabhängig davon läuft, ob das Experiment erfolgreich war, fehlgeschlagen oder mittendrin abgestürzt ist. Netem nimmt außerdem einen `seed` entgegen, damit sich zufälliger Verlust reproduzieren lässt. Sein Handbuch warnt, dass netem für realistische TCP-Leistungsergebnisse am Eingang des empfangenden Hosts sitzen muss.

Auf Ebene ganzer Maschinen zeigt Chaos Monkey, wie vorsichtige Voreinstellungen aussehen. Ab Werk ist `enabled` false und `leashed` true. Beendigungen werden also nur simuliert und protokolliert. Eingeschaltet arbeitet es mit Instanzgruppen nach App, Stack oder Cluster. Der mittlere Abstand zwischen Beendigungen beträgt zwei Tage pro Gruppe, der Mindestabstand einen Tag. Es läuft nur an Arbeitstagen zwischen 09:00 und 15:00 in seiner konfigurierten Zeitzone. Konten, Regionen und Stacks lassen sich ausschließen. Die Konfiguration hat einen Platz für eine Ausfallprüfung, die meldet, ob gerade ein Ausfall läuft. Die mitgelieferten Implementierungen dafür tun nichts; diese Prüfung muss der Betreiber bereitstellen. Die Liste von Produktionsfehlertests im SRE-Buch ist ebenso konkret: die Zahl der Tasks schnell oder langsam verringern, auf einmal die Kapazität eines Clusters verlieren und den Verkehr zu Backends ins Leere leiten, auch zu nicht kritischen, deren Schweigen ein Frontend trotzdem zum Stillstand bringen kann.

Die Fallstudie auf dieser Website, die zeigt, wie das Funktionieren der Sperre gegen Eindringversuche bei jedem Durchlauf nachgewiesen wurde, stammt vom September 2026. Sie ist kein Chaos-Experiment im vollen Sinn, hat aber dieselbe Form. Jeder Härtungsdurchlauf sperrt eine Adresse aus dem von Standards für Dokumentation reservierten Block, liest die entstandene Regel aus dem Kernel zurück, statt dem Status des Dienstes zu vertrauen, und hebt die Sperre in einem

Aufräumblock auf, der bei bestandener wie fehlgeschlagener Prüfung läuft. Weil die Testadresse niemandem gehört, kann eine nach einem fehlgeschlagenen Durchlauf zurückbleibende Sperre kein reales Netz aussperren. Die Seite benennt die Kosten offen: Bei jedem Durchlauf eine Regel in Produktion zu schreiben ist ein echter Eingriff in ein laufendes System. Er wird akzeptiert, weil sich die Alternative bereits als schlechter erwiesen hatte. Die Implementierung von Backups und Wiederherstellungstests überträgt diese Idee in ihrem Entwurf auf die Wiederherstellung: Ein monatlicher Workflow stellt eine bekannte Datei an einem temporären Ort wieder her und vergleicht ihre Bytes. Die Implementierungsdokumentation vom 2026-10-27 hält fest, dass die Timer ohne konfiguriertes Repository deaktiviert waren, und dokumentiert keine abgeschlossene reale Wiederherstellung. Der Workflow existiert als Code. Eine erfolgreiche Wiederherstellungsübung bleibt eine gesonderte Leistung.

## Wo es schiefgeht

**Dinge ohne Hypothese kaputtmachen.** Einen zufälligen Server zu beenden und zuzusehen ist eine Vorführung, kein Experiment. Schreiben Sie zuerst Normalzustand, Hypothese und Abbruchbedingung auf. Kann das Team seinen Normalzustand nicht in Echtzeit auf einem Dashboard sehen, muss zuerst das behoben werden. Das Experiment kann warten.

**Interna statt Ausgabe beobachten.** Die CPU-Auslastung auf einem Host oder die Länge einer Warteschlange sagt wenig darüber aus, ob Nutzer etwas bemerkten. Die Principles verlangen messbare Ausgaben des gesamten Systems wie Durchsatz, Fehlerraten und Latenzperzentile. Darum geht es in der Hypothese.

**Ein unbegrenzter Wirkungsradius.** Die Principles akzeptieren gewisse kurzfristige negative Auswirkungen, machen aber den Entwickler für ihre Begrenzung verantwortlich. Beginnen Sie mit einer Instanz oder einem kleinen Verkehrsanteil. Halten Sie einen Abbruchschalter bereit, der ohne das schadensverursachende Werkzeug funktioniert. Starten Sie nie ein Experiment während eines laufenden Vorfalls.

**Nie über Staging hinauskommen.** Eine Staging-Umgebung ist ein vernünftiger erster und ein schlechter letzter Schritt, weil ihr Verkehr, ihre Daten und ihre Nachbarn nicht die der Produktion sind. Wechseln Sie zu einem kleinen Teil echten Verkehrs, sobald das Experiment vor Produktion fehlerfrei gelaufen ist, statt die Staging-Ergebnisse als abschließende Antwort zu behandeln.

**Es einmal ausführen.** Ein System, das im März ein Experiment überstanden hat, wurde seither oft geändert. Die Principles nennen manuelle Experimente arbeitsaufwendig und auf Dauer nicht tragfähig. Sie verlangen automatisierte, kontinuierlich laufende Experimente, damit eine Regression der Widerstandsfähigkeit in der Woche auffällt, in der sie eingeführt wird.

**Immer nur Maschinen beenden.** Klar erkennbare Abstürze sind der einfache Fall, weil etwas sie bemerkt. Die Principles nennen auch Softwarefehler wie fehlerhaft formatierte Antworten und Ereignisse ohne Ausfall wie eine Verkehrsspitze oder Skalierung. Das SRE-Buch ergänzt ein Backend, das Anfragen annimmt und nie antwortet. Das kann bei langen Anfragefristen trotzdem ein Frontend beeinträchtigen. Priorisieren Sie Variablen nach wahrscheinlicher Auswirkung oder Häufigkeit.

**Erkenntnisse ohne Folgen.** Eine widerlegte Hypothese ist eine benannte Schwäche. Geben Sie ihr im selben Tracker wie allen anderen Aufgaben einen Verantwortlichen und eine Priorität. Wiederholen Sie das Experiment nach Auslieferung der Korrektur, um zu bestätigen, dass der Normalzustand nun bestehen bleibt.

## Weiterführende Lektüre

- Principles of Chaos Engineering — die Definition, die vier Schritte eines Experiments und die fünf fortgeschrittenen Prinzipien, zuletzt im März 2019 aktualisiert.
- Chaos Monkey — Netflix' Dokumentation seines Werkzeugs zum Beenden von Instanzen, das Spinnaker benötigt.
- Chaos Monkey: Verhalten über Spinnaker konfigurieren — Häufigkeit der Beendigungen, Gruppierung und Ausnahmen mit Standardwerten.
- Chaos Monkey: Format der Konfigurationsdatei — jede Einstellung und ihr Standardwert, einschließlich Leashed-Modus und Arbeitszeitfenster.
- Kaskadierende Ausfälle bewältigen — Kapitel 22 von Googles Buch Site Reliability Engineering, einschließlich Tests bis zum Versagen und Fehlertests in Produktion.
- tc-netem(8) — die Linux-Handbuchseite des Netzwerkemulators zum Einbringen von Verzögerung, Verlust und Beschädigung.



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 Chaos Engineering?“ 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/chaos-engineering/**