Engineer

Chaos engineering

kontrollerede forsøg, der indfører fejl i et kørende system

Hvad det er

Chaos engineering er praksissen med bevidst og under kontrol at påføre et kørende system de fejl, der også sker i virkeligheden, for at se, om det klarer dem. Principles of Chaos Engineering, det korte dokument, som de fleste i feltet henviser til, beskriver det som »disciplinen at eksperimentere med et system for at opbygge tillid« til, at systemet kan modstå ustabile forhold i produktion. Det vigtige ord er eksperimentere. Et chaos-forsøg tjekker ikke en funktion op mod en specifikation; det stiller systemet som helhed et spørgsmål og tager imod det svar, der kommer.

Begrundelsen tager udgangspunkt i distribuerede systemer. Hver enkelt tjeneste kan fungere korrekt, og alligevel kan samspillet mellem dem give resultater, som ingen havde forudset, og sjældne hændelser som en server, der går ned, eller en netværksforbindelse, der bliver skåret over, gør det værre. Principperne nævner de svagheder, der gemmer sig her: forkerte reserveløsninger, når en tjeneste ikke svarer, storme af genforsøg på grund af dårligt indstillede timeouts, nedbrud, når en tjeneste længere nede i kæden får mere trafik, end den kan klare, og at en fejl i én komponent får flere andre til at gå ned. Ingen af dem viser sig, når hver del testes for sig.

Et forsøg har fire trin i principperne:

- Definér en stabil tilstand (steady state) som et målbart output fra systemet, der viser normal adfærd, for eksempel gennemløb, fejlrate eller en percentil for svartid.
- Opstil hypotesen, at den stabile tilstand fortsætter både i en kontrolgruppe og i en forsøgsgruppe.
- Indfør variable, der afspejler virkelige hændelser: servere, der går ned, diske, der svigter, netværksforbindelser, der bliver afbrudt.

- Forsøg at modbevise hypotesen ved at lede efter en forskel i den stabile tilstand mellem de to grupper.

  Jo sværere det er at forstyrre den stabile tilstand, jo mere tillid kan holdet have til systemet. Falder hypotesen, har holdet fundet en svaghed, mens den stadig er lille, og mens folk holder øje, i stedet for under et nedbrud midt om natten.

Det mest kendte værktøj er Netflix' Chaos Monkey, der tilfældigt afslutter virtuelle maskiner og containere i produktion. Projektets README kalder det et eksempel på et værktøj, der følger principperne, og formålet er at få udviklere til at bygge tjenester, der kan tåle, at en instans forsvinder.

Brug begrebet i en beslutning

Brug »Hvad er chaos engineering?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Overvågning og alarmer, der siger til, mens fejlen stadig er lille, og tallene, der viser, hvor tit den bliver ved med at være det.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har bygget det første observerbarhedsdashboard — virksomhedens første overblik over sine egne systemer

Har bygget det første observerbarhedsdashboard Kontakt os →

Hvorfor det betyder noget

Redundans, failover og reserveløsninger er alle sammen løfter om, hvad der sker, når noget går i stykker, og et løfte, der aldrig er afprøvet, er en hypotese. En standby-server, der aldrig har taget trafik, en politik for genforsøg, der aldrig har mødt en langsom afhængighed, en alarm, der aldrig har lydt: hver af dem virker måske, og den første rigtige fejl er en dyr måde at finde ud af det på. Chaos engineering flytter opdagelsen til et tidspunkt, holdet selv vælger.

Det tester også systemet frem for delene. Enhedstest og integrationstest kontrollerer adfærd, som nogen allerede har tænkt på. Principperne siger det direkte: et chaos-forsøg efterviser, at systemet virker, og forsøger ikke at validere, hvordan det virker. Derfor ser det på det output, en bruger ville mærke, som fejlrate og svartid, og ikke på det indre af en enkelt komponent.

Beviserne findes i produktion. Systemer opfører sig forskelligt alt efter miljø og trafikmønstre, og principperne fremhæver, at kun rigtig trafik giver et pålideligt billede af, hvilken vej en forespørgsel tager. Googles bog om Site Reliability Engineering når frem til det samme fra den anden side: syntetisk belastning viser ikke nødvendigvis, hvor komponenterne reelt bryder sammen, så når en tjeneste

er beskyttet mod overbelastning, foreslår bogen fejltest i en lille del af produktionen, med ekstra kapacitet klar, hvis beskyttelsen ikke virker, og nogen må skifte over manuelt.

Endelig ændrer det incitamenterne. Chaos Monkeys README siger det ligeud: når udviklere oftere møder fejl, får de en grund til at bygge robuste tjenester. Et hold, hvis instanser kan blive afsluttet på en hvilken som helst arbejdsdag, designer efter det fra begyndelsen i stedet for at opdage hullet den dag, en maskine dør uden varsel.

Sådan ser det ud i praksis

Et chaos-forsøg skrives ned, før det køres, ligesom en testplan. Her er et for en opdigtet webbutik, hvor alle tal er opfundet til eksemplet:

```
Forsøg:          betalingsudbyderen bliver langsom
Stabil tilstand: andel gennemførte betalinger og p99-svartid, pr. 5 min
Hypotese:        med 300 ms ekstra på kald til betalingsudbyderen holder
                 forsøgsgruppens gennemførelsesrate sig inden for 0,2
                 procentpoint af kontrolgruppens
Grupper:         2 % af betalingstrafikken i hver, tilfældigt fordelt
Variabel:        300 ms forsinkelse i betalingsklienten, intet andet
Afbryd hvis:     forskellen er over 1 procentpoint i 2 minutter i træk
Tidsrum:         30 min i arbejdstiden, vagthavende orienteret på forhånd
Resultat:        hypotesen modbevist; genforsøg tredoblede kaldene til
                 udbyderen, og p99-svartiden passerede timeouten på 1 s
Opfølgning:      et budget for genforsøg pr. forespørgsel; kør forsøget igen
```

En modbevist hypotese er det nyttige udfald her. Svagheden bliver fundet på 2 % af trafikken i nogle få minutter, med en afbrydelsesbetingelse, der allerede er aftalt, og opfølgningen er en teknisk ændring og ikke en undskyldning.

Selve variablen påføres tit med helt almindelige værktøjer. På Linux efterligner kødisciplinen netem egenskaberne ved virkelige netværk, herunder forsinkelse, tab, duplikering og beskadigede pakker. For eksempel:

```
# forsøg 1: læg 100 ms forsinkelse på hver pakke, der sendes ud på eth0
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root
```

```
# forsøg 2: smid én pakke ud af tusind væk, tilfældigt
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

Linjen med `del` er den vigtige, og den hører hjemme i et oprydningstrin, der kører, uanset om forsøget bestod, fejlede eller gik ned halvvejs. Netem tager også et `seed`, så tilfældigt tab kan gentages, og manualen advarer om, at netem skal sidde på den modtagende værts indgående trafik, hvis resultater for TCP-ydelse skal være realistiske.

På niveauet med hele maskiner viser Chaos Monkey, hvordan forsigtige standardværdier ser ud. Fra start er `enabled` falsk og `leashed` sand, hvilket betyder, at afslutninger kun simuleres og logges. Når værktøjet slås til, arbejder det med grupper af instanser (efter app, stack eller cluster) med en gennemsnitlig tid mellem afslutninger på to dage pr. gruppe og et minimum på én dag, kun på arbejdsdage og kun mellem kl. 9 og 15 i den konfigurerede tidszone. Konti, regioner og stacks kan undtages, og konfigurationen har en plads til en outage checker, der fortæller værktøjet, om der er et nedbrud i gang; de medfølgende implementeringer af den gør ingenting, så det tjek må driften selv levere. SRE-bogens liste over fejltest i produktion er lige så konkret: skær antallet af opgaver ned hurtigt eller langsomt, mist en hel klynges kapacitet på én gang, og lad backends sluge forespørgsler uden at svare, også de ikke-kritiske, hvis tavshed stadig kan få tjenesten, der kalder dem, til at gå i stå.

Her på sitet har casen om at bevise, at blokeringen virker ved hver kørsel, fra september 2026, ikke karakter af et chaos-forsøg i fuld forstand, men formen er den samme. Hver hærdningskørsel blokerer en adresse fra den blok, standarderne reserverer til dokumentation, læser den resulterende regel tilbage fra kernen i stedet for at stole på tjenestens egen status og ophæver blokeringen i en oprydningsblok, der kører, uanset om tjekket bestod eller fejlede. Fordi testadressen ikke tilhører nogen, kan en efterladt blokering, der overlever en fejlet kørsel, ikke lukke et rigtigt netværk ude, og siden lægger ikke skjul på prisen: at skrive en regel i produktion ved hver kørsel er et reelt indgreb i et system i drift, accepteret, fordi alternativet allerede havde vist sig at være værre. Implementeringen af sikkerhedskopiering og gendannelsestest bruger samme idé på gendannelse i sit design: et månedligt workflow gendanner en kendt fil til et midlertidigt arbejdsområde og sammenligner dens bytes. Implementeringsbeskrivelsen dateret 2026-10-27 angiver, at timerne var deaktiveret uden et konfigureret backuplager, og dokumenterer ingen gennemført gendannelse i drift. Workflowet findes i kode; en vellykket gendannelsesøvelse er fortsat en særskilt leverance.

Hvor det går galt

At slå ting i stykker uden en hypotese. At lukke en tilfældig server ned og se, hvad der sker, er en demonstration, ikke et forsøg. Skriv den stabile tilstand, hypotesen og afbrydelsesbetingelsen ned først. Kan holdet ikke se sin stabile tilstand på et dashboard i realtid, er det det første, der skal rettes, og forsøget kan vente.

At kigge på det indre i stedet for output. CPU på én vært eller længden af én kø siger ikke meget om, hvorvidt brugerne mærkede noget. Principperne beder om målbart output fra hele systemet, som gennemløb, fejlrater og percentiler for svartid, fordi det er det, hypotesen handler om.

En ubegrænset skadesradius. Principperne accepterer en vis kortvarig negativ effekt, men gør det til chaos-ingeniørens ansvar at holde følgerne små og afgrænsede. Begynd med én instans eller en lille procentdel af trafikken, hav en nødstop-knap, der virker uden det værktøj, der forårsagede skaden, og kør aldrig et forsøg oven i en igangværende hændelse.

At blive i testmiljøet. Et testmiljø er et fornuftigt første skridt og et dårligt sidste, fordi trafikken, data og naboerne ikke er produktionens. Gå videre til en lille del af den rigtige trafik, når forsøget har kørt rent før produktion, i stedet for at tage testmiljøets resultater som svaret.

At køre det én gang. Et system, der klarede et forsøg i marts, er blevet ændret mange gange siden. Principperne kalder manuelle forsøg arbejdskrævende og i sidste ende uholdbare og beder om forsøg, der er automatiseret til at køre løbende, så et tab af robusthed viser sig i den uge, det bliver indført.

Kun at slå maskiner ihjel. Rene nedbrud er det lette tilfælde, fordi noget opdager dem. Principperne nævner også softwarefejl som forkert formede svar og hændelser, der ikke er fejl, som en trafikspids eller en skalering, og SRE-bogen tilføjer en backend, der tager imod forespørgsler og aldrig svarer, hvilket stadig kan skade tjenesten, der kalder den, når forespørgslerne har lange tidsfrister. Prioritér variablerne efter den sandsynlige effekt eller efter, hvor tit de sker.

Fund, der ikke fører nogen vegne. En modbevist hypotese er en svaghed med et navn. Giv den en ejer og en prioritet i samme sagssystem som alt andet, og kør forsøget igen, når rettelsen er ude, for at bekræfte, at den stabile tilstand

nu holder.

Læs mere

- Principles of Chaos Engineering — definitionen, de fire trin i et forsøg og de fem avancerede principper, senest opdateret i marts 2019.
- Chaos Monkey — Netflix' dokumentation for værktøjet, der afslutter instanser, og som kræver Spinnaker.
- Chaos Monkey: configuring behavior via Spinnaker — hyppighed, gruppering og undtagelser med standardværdierne.
- Chaos Monkey: configuration file format — alle indstillinger og deres standardværdier, herunder `leashed` og tidsrummet i arbejdstiden.
- Addressing Cascading Failures — kapitel 22 i Googles bog om Site Reliability Engineering, blandt andet om at teste til grænsen og om fejltest i produktion.
- tc-netem(8) — Linux-manualsiden for netværksemulatoren, der bruges til at påføre forsinkelse, tab og beskadigede pakker.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er chaos engineering?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/chaos-engineering/