Engineer

Chaos engineering

kontrollerte forsøk som fører feil inn i et system i drift

Hva det er

Chaos engineering er praksisen med å føre feil fra virkeligheten inn i et system i drift, med vilje og under kontroll, for å finne ut om det tåler dem. Principles of Chaos Engineering, det korte dokumentet de fleste i fagmiljøet viser til, definerer det som «disiplinen å eksperimentere med et system for å bygge tillit» til at systemet tåler urolige forhold i produksjon. Ordet som teller, er eksperimentere. Et chaos-forsøk sjekker ikke en funksjon mot en spesifikasjon; det stiller systemet som helhet et spørsmål og godtar det svaret som kommer.

Argumentet for praksisen begynner med distribuerte systemer. Hver tjeneste i et slikt system kan virke som den skal, og samspillet mellom dem kan likevel gi utfall ingen hadde sett for seg, og sjeldne hendelser som en server som går ned eller en nettverksforbindelse som blir brutt, gjør det verre. Principles lister opp hvilke svakheter som gjemmer seg der: reserveinnstillinger som er feil når en tjeneste ikke svarer, stormer av nye forsøk på grunn av dårlig innstilte tidsavbrudd, brudd når en avhengighet lenger ned i kjeden får mer trafikk enn den tåler, og at en feil i én komponent får flere andre til å svikte. Ingen av disse viser seg når hver del testes for seg.

Et forsøk har fire trinn i Principles sin versjon:

- Definer stabil tilstand (steady state) som et målbart resultat fra systemet som viser normal oppførsel, for eksempel gjennomstrømning, feilrate eller en persentil for forsinkelse.
- Sett opp en hypotese om at den stabile tilstanden holder seg i både en kontrollgruppe og en forsøksgruppe.
- Før inn variabler som speiler virkelige hendelser: servere som går ned, disker som svikter, nettverksforbindelser som blir brutt.
- Prøv å motbevise hypotesen ved å lete etter en forskjell i stabil tilstand mellom de to gruppene.

Jo vanskeligere den stabile tilstanden er å forstyrre, desto mer tillit kan teamet ha til systemet. Faller hypotesen, har teamet funnet en svakhet mens den er liten og mens folk følger med, og ikke midt i et nattlig driftsavbrudd.

Det best kjente verktøyet er Netflix sin Chaos Monkey, som tilfeldig avslutter virtuelle maskiner og containere i produksjon. README-en omtaler det som et eksempel på et verktøy som følger Principles, og formålet er å få ingeniørene til å bygge tjenester som tåler at enkeltinstanser forsvinner.

Bruk begrepet i en beslutning

Bruk «Hva er chaos engineering?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Overvåking og varsler som sier fra mens en feil fortsatt er liten, og tallene som viser hvor ofte den holder seg liten.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Bygde selskapets første observerbarhetsdashbord — selskapets første innsyn i egne systemer

Bygde selskapets første observerbarhetsdashbord Ta kontakt →

Hvorfor det er viktig

Redundans, failover og reserveløsninger er alle løfter om hva som skjer når noe går i stykker, og et løfte som aldri er prøvd, er en hypotese. En standby-server som aldri har tatt trafikk, en regel for nye forsøk som aldri har møtt en treg avhengighet, en alarm som aldri har gått: hver av dem virker kanskje, og den første virkelige feilen er en dyr måte å finne det ut på. Chaos engineering flytter oppdagelsen til et tidspunkt teamet selv velger.

Det tester også systemet og ikke delene. Enhetstester og integrasjonstester sjekker oppførsel noen allerede har tenkt på. Principles trekker skillet rett ut: chaos-forsøk bekrefter at systemet faktisk virker, i stedet for å prøve å godtgjøre hvordan det virker. Derfor ser de på det en bruker ville merke, som feilrate og forsinkelse, og ikke på det indre i én enkelt komponent.

Bevisene ligger i produksjon. Systemer oppfører seg ulikt avhengig av miljø og trafikkmønster, og Principles hevder at bare utvalg av ekte trafikk pålitelig fanger veien en forespørsel tar. Googles bok om Site Reliability Engineering kommer til samme konklusjon fra motsatt kant: syntetisk last viser ikke alltid hvor komponentene virkelig ryker, så når en tjeneste først har vern mot overbelastning, foreslår boken feiltester i en liten del av produksjonen, med ekstra kapasitet klar i tilfelle vernet ikke virker og noen må bytte over for hånd.

Til slutt endrer det insentivene. Chaos Monkey sin README sier det rett ut: når ingeniører møter feil oftere, får de en grunn til å bygge robuste tjenester. Et team der instanser kan bli avsluttet hvilken som helst arbeidsdag, planlegger for det fra starten av, i stedet for å oppdage hullet den dagen en maskin forsvinner uten forvarsel.

Slik ser det ut i praksis

Et chaos-forsøk skrives ned før det kjøres, omtrent som en testplan. Her er ett for en oppdiktet nettbutikk, der alle tall er laget for eksemplets skyld:

```
Forsøk:          betalingsleverandøren blir treg
Stabil tilstand: andel vellykkede kjøp og p99 for kassen, per 5 min
Hypotese:        med 300 ms ekstra på kall til betalingsleverandøren
                 avviker forsøksgruppens andel vellykkede kjøp
                 høyst 0,2 prosentpoeng fra kontrollgruppens
Grupper:         2 % av trafikken til kassen i hver, tilfeldig fordelt
Variabel:        300 ms forsinkelse i betalingsklienten, ikke noe annet
Avbryt hvis:     avvik over 1 prosentpoeng i 2 minutter på rad
Vindu:           30 min i arbeidstiden, vakthavende varslet på forhånd
Resultat:        hypotesen motbevist; nye forsøk tredoblet kallene til
                 leverandøren, og p99 gikk over tidsavbruddet på 1 s
Oppfølging:      et budsjett for nye forsøk per forespørsel; kjør igjen
```

Den motbeviste hypotesen er det nyttige utfallet her. Svakheten blir funnet på 2 % av trafikken i noen få minutter, med en avbruddsbetingelse som er avtalt på forhånd, og oppfølgingen er en teknisk endring i stedet for en unnskyldning.

Selve variabelen føres ofte inn med helt vanlige verktøy. På Linux etterligner nettverksemulatoren netem, som settes opp med tc, egenskapene til virkelige nettverk, blant annet forsinkelse, tap, duplisering og korrupsjon. For eksempel:

```
# forsøk 1: legg 100 ms forsinkelse på hver pakke som går ut av eth0
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root

# forsøk 2: kast tilfeldig én av tusen pakker
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

Linjen med `del` er den viktigste, og den hører hjemme i et oppryddingstrinn som kjører enten forsøket besto, feilet eller krasjet halvveis. Netem tar også en `seed`, slik at tilfeldig tap kan gjentas, og manualsiden advarer om at netem må ligge på inngangen til mottakerverten hvis måleresultater for TCP-ytelse skal bli realistiske.

På nivået med hele maskiner viser Chaos Monkey hvordan forsiktige standardverdier ser ut. Rett etter installasjon er `enabled` false og `leashed` true, som betyr at avslutninger bare blir simulert og logget. Når verktøyet slås på, arbeider det med grupper av instanser (etter app, stack eller cluster), med en gjennomsnittlig tid mellom avslutninger på to dager per gruppe og et minimum på én dag, bare på arbeidsdager og bare mellom kl. 9 og 15 i den konfigurerte tidssonen. Kontoer, regioner og stacks kan unntas, og konfigurasjonen har en plass til en outage checker som forteller verktøyet om et driftsavbrudd pågår; implementeringene som følger med, gjør ingenting, så den sjekken må driften selv levere. SRE-bokens liste over feiltester i produksjon er like konkret: skru ned antallet oppgaver raskt eller sakte, mist en hel klynges kapasitet på én gang, og la backends svelge forespørsler uten å svare, også de ikke-kritiske, der tausheten likevel kan få tjenesten som kaller dem, til å stoppe opp.

På dette nettstedet er prosjektgjennomgangen om hvordan det ble bevist at utestengingen virker ved hver kjøring, fra september 2026, ikke et kaoseksperiment i full forstand, men den har samme form. Hver herdingskjøring blokkerer en adresse fra blokken standardene reserverer for dokumentasjon, leser den resulterende regelen tilbake fra kjernen i stedet for å stole på tjenestens egen status og opphever blokkeringen i en oppryddingsblokk som kjøres enten sjekken besto eller feilet. Fordi testadressen ikke tilhører noen, kan en utilsiktet blokkering som overlever en mislykket kjøring, ikke stenge ute et virkelig nettverk. Siden er åpen om kostnaden: å skrive en regel i produksjon ved hver kjøring er et reelt inngrep i et system i drift, akseptert fordi alternativet allerede hadde vist seg å være verre. Implementeringen av sikkerhetskopiering og gjenopprettingstester bruker denne tanken på gjenoppretting i løsningsutformingen: en månedlig arbeidsflyt gjenoppretter en kjent fil til et midlertidig arbeidsområde og sammenligner bytene. Implementeringsdokumentasjonen datert 2026-10-27 sier at timerne var deaktivert uten et konfigurert sikkerhetskopirepo, og dokumenterer ingen fullført gjenoppretting i drift. Arbeidsflyten finnes i kode; en vellykket gjenopprettingsøvelse er fortsatt en separat leveranse.

Hvor det går galt

Å ødelegge uten en hypotese. Å slå av en tilfeldig server og se hva som skjer, er en demonstrasjon og ikke et forsøk. Skriv ned stabil tilstand, hypotese og avbruddsbetingelse først. Kan ikke teamet se sin stabile tilstand på et dashbord i sanntid, er det det første som må rettes, og forsøket får vente.

Å se på det indre i stedet for resultatet. CPU-bruk på én vert eller lengden på én kø sier lite om brukerne merket noe. Principles ber om målbare resultater fra hele systemet, som gjennomstrømning, feilrater og persentiler for forsinkelse, fordi det er det hypotesen handler om.

En ubegrenset skaderadius. Principles godtar en viss kortvarig negativ virkning, men gjør det til ingeniørens ansvar å holde følgene små og avgrensede. Begynn med én instans eller en liten prosentdel av trafikken, ha en nødstopp som virker uten verktøyet som gjorde skaden, og kjør aldri et forsøk oppå en hendelse som allerede pågår.

Å aldri komme ut av staging. Et testmiljø er et fornuftig første steg og et dårlig siste, fordi trafikken, dataene og naboene der ikke er produksjonens. Gå over til en liten del av ekte trafikk når forsøket har kjørt rent før produksjon, i stedet for å behandle resultatene fra staging som svaret.

Å kjøre det én gang. Et system som klarte seg gjennom et forsøk i mars, er endret mange ganger siden. Principles kaller manuelle forsøk arbeidskrevende og til syvende og sist uholdbare, og ber om forsøk som er automatisert til å kjøre løpende, slik at tap av robusthet viser seg den uka det blir innført.

Bare å drepe maskiner. Rene krasj er det enkle tilfellet, fordi noe legger merke til dem. Principles nevner også svikt i programvaren, som feilformede svar, og hendelser som ikke er feil, som en trafikktopp eller en skalering, og SRE-boken legger til en backend som tar imot forespørsler og aldri svarer, noe som fortsatt kan skade tjenesten som kaller den, når forespørslene har lange tidsfrister. Prioriter variablene etter sannsynlig virkning eller etter hvor ofte de skjer.

Funn som ikke fører noe sted. En motbevist hypotese er en svakhet med navn. Gi den en eier og en prioritet i det samme saksverktøyet som alt annet, og kjør forsøket på nytt når rettelsen er ute, for å bekrefte at den stabile tilstanden nå holder.

Les mer

- Principles of Chaos Engineering — definisjonen, de fire trinnene i et forsøk og de fem avanserte prinsippene, sist oppdatert i mars 2019.
- Chaos Monkey — Netflix sin dokumentasjon for verktøyet som avslutter instanser, og som krever Spinnaker.
- Chaos Monkey: configuring behavior via Spinnaker — hvor ofte instanser avsluttes, gruppering og unntak, med standardverdiene.
- Chaos Monkey: configuration file format — alle innstillinger og standardverdiene deres, blant annet `leashed` og vinduet for arbeidstid.
- Addressing Cascading Failures — kapittel 22 i Googles bok om Site Reliability Engineering, blant annet om testing til noe ryker og om feiltester i produksjon.
- tc-netem(8) — Linux-manualsiden for nettverksemulatoren som brukes til å føre inn forsinkelse, tap og korrupsjon.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er chaos engineering?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/chaos-engineering/