Engineer

Chaos engineering

kontrollerade experiment som för in fel i ett system i drift

Vad det är

Chaos engineering är metoden att med flit och under kontroll föra in verkliga fel i ett system i drift, för att ta reda på om det klarar dem. Principles of Chaos Engineering, det korta dokument som de flesta inom området hänvisar till, definierar det som ”disciplinen att experimentera med ett system för att bygga förtroende” för att systemet tål turbulenta förhållanden i produktion. Ordet som räknas är experimentera. Ett chaos-experiment kontrollerar inte en funktion mot en specifikation; det ställer en fråga till systemet som helhet och tar emot det svar som kommer.

Argumentet för metoden börjar i distribuerade system. Varje tjänst i ett sådant system kan fungera som den ska, och samspelet mellan dem kan ändå ge utfall som ingen hade förutsett, och sällsynta händelser som en server som kraschar eller en nätverksförbindelse som bryts gör saken värre. Principles räknar upp vilka svagheter som gömmer sig där: reservinställningar som är fel när en tjänst inte svarar, stormar av omförsök på grund av dåligt inställda tidsgränser, avbrott när ett beroende längre ner i kedjan får mer trafik än det klarar, och att en felande komponent får flera andra att sluta fungera. Inget av detta syns när varje del testas för sig.

Ett experiment har fyra steg i Principles version:

- Definiera stabilt tillstånd (steady state) som ett mätbart utfall från systemet som visar normalt beteende, till exempel genomströmning, felfrekvens eller en percentil för svarstid.
- Ställ upp en hypotes om att det stabila tillståndet håller i sig i både en kontrollgrupp och en försöksgrupp.
- För in variabler som speglar verkliga händelser: servrar som kraschar, diskar som går sönder, nätverksförbindelser som bryts.

- Försök motbevisa hypotesen genom att leta efter en skillnad i stabilt tillstånd mellan de två grupperna.

  Ju svårare det stabila tillståndet är att rubba, desto mer förtroende kan teamet ha för systemet. Faller hypotesen har teamet hittat en svaghet medan den är liten och medan folk håller utkik, och inte mitt i ett avbrott på natten.

Det mest kända verktyget är Netflix Chaos Monkey, som slumpmässigt avslutar virtuella maskiner och containrar i produktion. Dess README beskriver det som ett exempel på ett verktyg som följer Principles, och syftet är att få ingenjörerna att bygga tjänster som klarar att enskilda instanser försvinner.

Använd begreppet i ett beslut

Använd ”Vad är chaos engineering?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Övervakning och larm som säger till medan ett fel fortfarande är litet, och siffrorna som visar hur ofta det förblir så.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Byggde företagets första observerbarhetsdashboard — företagets första överblick över de egna systemen

Byggde företagets första observerbarhetsdashboard Hör av er →

Varför det spelar roll

Redundans, failover och reservlösningar är alla löften om vad som händer när något går sönder, och ett löfte som aldrig har prövats är en hypotes. En standby-server som aldrig har tagit trafik, en regel för omförsök som aldrig har mött ett långsamt beroende, ett larm som aldrig har gått: alla kanske fungerar, och det första verkliga felet är ett dyrt sätt att ta reda på det. Chaos engineering flyttar upptäckten till en tidpunkt som teamet själv väljer.

Det testar också systemet och inte delarna. Enhetstester och integrationstester kontrollerar beteende som någon redan har tänkt på. Principles drar gränsen rakt ut: chaos-experiment bekräftar att systemet faktiskt fungerar, i stället för att försöka belägga hur det fungerar. Därför tittar de på det en användare skulle märka, som felfrekvens och svarstid, och inte på det inre i en enskild komponent.

Beläggen finns i produktion. System beter sig olika beroende på miljö och trafikmönster, och Principles hävdar att bara stickprov på verklig trafik pålitligt fångar den väg en förfrågan tar. Googles bok om Site Reliability Engineering kommer fram till samma slutsats från andra hållet: syntetisk last visar inte alltid

var komponenterna verkligen går sönder, så när en tjänst väl har skydd mot överbelastning föreslår boken feltester i en liten del av produktionen, med extra kapacitet redo ifall skydden inte fungerar och någon måste växla över för hand.

Till sist ändrar det drivkrafterna. Chaos Monkeys README säger det rakt ut: när ingenjörer möter fel oftare får de ett skäl att bygga motståndskraftiga tjänster. Ett team vars instanser kan avslutas vilken arbetsdag som helst planerar för det från början, i stället för att upptäcka luckan den dag en maskin försvinner utan förvarning.

Så ser det ut i praktiken

Ett chaos-experiment skrivs ner innan det körs, ungefär som en testplan. Här är ett för en påhittad webbutik, där alla siffror är uppdiktade för exemplets skull:

```
Experiment:      betalleverantören blir långsam
Stabilt läge:    andel lyckade köp och p99 för kassan, per 5 min
Hypotes:         med 300 ms extra på anrop till betalleverantören
                 skiljer sig försöksgruppens andel lyckade köp högst
                 0,2 procentenheter från kontrollgruppens
Grupper:         2 % av trafiken till kassan i vardera, slumpvis
Variabel:        300 ms fördröjning i betalklienten, inget annat
Avbryt om:       skillnad över 1 procentenhet i 2 minuter i följd
Fönster:         30 min under arbetstid, jourhavande förvarnad
Resultat:        hypotesen motbevisad; omförsök tredubblade anropen
                 till leverantören och p99 passerade tidsgränsen på 1 s
Uppföljning:     en budget för omförsök per förfrågan; kör om sedan
```

Den motbevisade hypotesen är det användbara utfallet här. Svagheten hittas på 2 % av trafiken under några minuter, med ett avbrottsvillkor som redan är överenskommet, och uppföljningen är en teknisk ändring i stället för en ursäkt.

Själva variabeln förs ofta in med helt vanliga verktyg. På Linux efterliknar nätverksemulatorn netem, som ställs in med tc, egenskaperna hos verkliga nätverk, bland annat fördröjning, förlust, duplicering och korruption.
Till exempel:

```
# experiment 1: lägg 100 ms fördröjning på varje paket som lämnar eth0
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root
```

```
# experiment 2: släng slumpvis ett av tusen paket
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

Raden med `del` är den viktiga, och den hör hemma i ett städsteg som körs oavsett om experimentet gick igenom, misslyckades eller kraschade halvvägs. Netem tar också ett `seed`, så att slumpmässig förlust kan upprepas, och manualsidan varnar för att netem måste ligga på ingången hos den mottagande värden om mätresultat för TCP-prestanda ska bli realistiska.

På nivån hela maskiner visar Chaos Monkey hur försiktiga standardvärden ser ut. Direkt efter installationen är `enabled` false och `leashed` true, vilket betyder att avslutningar bara simuleras och loggas. När verktyget slås på arbetar det med grupper av instanser (per app, stack eller cluster), med en genomsnittlig tid mellan avslutningar på två dagar per grupp och ett minimum på en dag, bara på arbetsdagar och bara mellan kl. 9 och 15 i den inställda tidszonen. Konton, regioner och stackar kan undantas, och konfigurationen har en plats för en outage checker som talar om för verktyget om ett avbrott pågår; de implementationer som följer med gör ingenting, så den kontrollen får driften själv stå för. SRE-bokens lista över feltester i produktion är lika konkret: minska antalet uppgifter snabbt eller långsamt, förlora ett helt klusters kapacitet på en gång och låt backends svälja förfrågningar utan att svara, även de icke-kritiska, vars tystnad ändå kan få tjänsten som anropar dem att stanna upp.

Här på webbplatsen är fallstudien om att bevisa att intrångsspärren fungerar vid varje körning, från september 2026, inget chaos-experiment i full mening, men den har samma form. Varje härdningskörning spärrar en adress ur det block som standarderna reserverar för dokumentation, läser tillbaka den resulterande regeln från kärnan i stället för att lita på tjänstens egen status och häver spärren i ett städblock som körs oavsett om kontrollen gick igenom eller misslyckades. Eftersom testadressen inte tillhör någon kan en kvarglömd spärr som överlever en misslyckad körning inte stänga ute ett verkligt nätverk, och sidan är öppen med priset: att skriva en regel i produktion vid varje körning är ett verkligt ingrepp i ett system i drift, godtaget eftersom alternativet redan hade visat sig vara sämre. Implementeringen av säkerhetskopiering och återställningstest tillämpar samma tanke på återställning i sin design: ett månatligt arbetsflöde återställer en känd fil till ett tillfälligt arbetsutrymme och jämför dess byte. Implementeringsbeskrivningen daterad 2026-10-27 anger att timerenheterna var inaktiverade utan ett konfigurerat repo och redovisar ingen genomförd återställning i drift. Arbetsflödet finns i kod; en lyckad återställningsövning återstår som en separat leverans.

Var det går fel

Att ha sönder saker utan hypotes. Att stänga av en slumpvis vald server och se vad som händer är en demonstration, inte ett experiment. Skriv ner stabilt tillstånd, hypotes och avbrottsvillkor först. Kan teamet inte se sitt stabila tillstånd på en instrumentpanel i realtid är det det första som ska åtgärdas, och experimentet får vänta.

Att titta på det inre i stället för utfallet. CPU på en värd eller längden på en kö säger lite om huruvida användarna märkte något. Principles ber om mätbart utfall från hela systemet, som genomströmning, felfrekvens och percentiler för svarstid, eftersom det är det hypotesen handlar om.

En obegränsad sprängradie. Principles godtar en viss kortvarig negativ påverkan men lägger ansvaret för att hålla följderna små och avgränsade på ingenjören. Börja med en instans eller en liten andel av trafiken, ha ett nödstopp som fungerar utan verktyget som orsakade skadan och kör aldrig ett experiment ovanpå en incident som redan pågår.

Att aldrig lämna stagingmiljön. En testmiljö är ett rimligt första steg och ett dåligt sista, eftersom trafiken, data och grannarna där inte är produktionens. Gå över till en liten del av verklig trafik när experimentet har gått rent före produktion, i stället för att se resultaten från testmiljön som svaret.

Att köra det en gång. Ett system som klarade ett experiment i mars har ändrats många gånger sedan dess. Principles kallar manuella experiment arbetskrävande och i längden ohållbara, och ber om experiment som är automatiserade för att köras löpande, så att förlorad motståndskraft syns samma vecka som den införs.

Att bara döda maskiner. Rena krascher är det enkla fallet, eftersom något märker dem. Principles nämner också fel i mjukvaran, som felformade svar, och händelser som inte är fel, som en trafiktopp eller en skalning, och SRE-boken lägger till en backend som tar emot förfrågningar och aldrig svarar, vilket ändå kan skada tjänsten som anropar den när förfrågningarna har långa tidsfrister. Prioritera variablerna efter trolig påverkan eller efter hur ofta de inträffar.

Fynd som inte leder någonstans. En motbevisad hypotes är en svaghet med ett namn. Ge den en ägare och en prioritet i samma ärendesystem som allt annat, och kör om experimentet när rättningen är ute för att bekräfta att det stabila

tillståndet nu håller.

Läs mer

- Principles of Chaos Engineering — definitionen, de fyra stegen i ett experiment och de fem avancerade principerna, senast uppdaterad i mars 2019.
- Chaos Monkey — Netflix dokumentation för verktyget som avslutar instanser och som kräver Spinnaker.
- Chaos Monkey: configuring behavior via Spinnaker — hur ofta instanser avslutas, gruppering och undantag, med standardvärdena.
- Chaos Monkey: configuration file format — alla inställningar och deras standardvärden, bland annat `leashed` och fönstret för arbetstid.
- Addressing Cascading Failures — kapitel 22 i Googles bok om Site Reliability Engineering, bland annat om att testa tills något går sönder och om feltester i produktion.
- tc-netem(8) — Linux-manualsidan för nätverksemulatorn som används för att föra in fördröjning, förlust och korruption.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är chaos engineering?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/chaos-engineering/