Engineer

Хаос-інженерія

контрольовані експерименти зі збоями в робочій системі

Що це таке

Хаос-інженерія — це практика навмисно й під контролем вносити в робочу систему ті збої, що трапляються в реальному житті, щоб перевірити, чи вона з ними впорається. Principles of Chaos Engineering, короткий документ, на який посилається більшість практиків, описує її як «дисципліну експериментування з системою, щоб здобути впевненість» у тому, що система витримає неспокійні умови в продакшні. Головне слово тут — експериментування. Хаос-експеримент не звіряє функцію зі специфікацією; він ставить запитання системі в цілому й приймає ту відповідь, яку отримає.

Аргументи на її користь починаються з розподілених систем. Кожен окремий сервіс може працювати правильно, а взаємодія між ними все одно дасть результат, якого ніхто не передбачив, і рідкісні події на кшталт сервера, що впав, чи розірваного мережевого з'єднання лише погіршують справу. Принципи перелічують слабкі місця, які там ховаються: неправильну резервну поведінку, коли сервіс недоступний, шторми повторних спроб через погано налаштовані тайм-аути, відмови, коли залежний сервіс нижче в ланцюжку отримує більше трафіку, ніж може обробити, і каскадні збої, коли відмова одного компонента зупиняє кілька інших. Жодне з цього не проявляється, коли кожну частину перевіряють окремо.

Експеримент у версії принципів має чотири кроки:

- Визначити усталений стан як вимірюваний вихід системи, що свідчить про нормальну поведінку: пропускну здатність, частку помилок або перцентиль затримки.
- Висунути гіпотезу, що усталений стан збережеться і в контрольній, і в експериментальній групі.
- Внести змінні, що відображають реальні події: сервери, які падають, диски, які виходять з ладу, мережеві з'єднання, які обриваються.

- Спробувати спростувати гіпотезу, шукаючи різницю в усталеному стані між двома групами.

  Що важче порушити усталений стан, то більше впевненості команда може мати в системі. Якщо гіпотеза не встояла, команда знайшла слабке місце, поки воно ще мале і поки всі пильнують, а не під час нічної аварії.

Найвідоміший інструмент — Chaos Monkey від Netflix, який випадково завершує екземпляри віртуальних машин і контейнери в продакшні. README проєкту називає його прикладом інструмента, що дотримується цих принципів, а його мета — змусити інженерів будувати сервіси, стійкі до втрати окремого екземпляра.

Застосуйте поняття до рішення

Застосуйте «Що таке хаос-інженерія?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Моніторинг і сповіщення, які попереджають, поки збій ще малий, і цифри, що показують, як часто так і залишається.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Запустила перший дашборд спостережуваності — перший огляд власних систем компанії

Запустила перший дашборд спостережуваності Зв'язатися з нами →

Чому це важливо

Резервування, перемикання на резерв і запасні шляхи — це обіцянки про те, що станеться, коли щось зламається, а обіцянка, яку ніколи не перевіряли, — лише гіпотеза. Резервний сервер, який ніколи не приймав трафіку, політика повторних спроб, яка ніколи не стикалася з повільною залежністю, сповіщення, яке ніколи не спрацьовувало: кожне з них, можливо, працює, а перший справжній збій — дорогий спосіб це з’ясувати. Хаос-інженерія переносить це відкриття на час, який команда обирає сама.

Вона також перевіряє систему, а не її частини. Модульні та інтеграційні тести перевіряють поведінку, про яку хтось уже подумав. Принципи кажуть про це прямо: хаос-експеримент підтверджує, що система працює, а не намагається довести, як саме вона працює. Тому він дивиться на те, що помітить користувач, — частку помилок і затримку, — а не на внутрішню будову окремого компонента.

Докази лежать у продакшні. Системи поводяться по-різному залежно від середовища й характеру трафіку, і принципи наголошують, що лише справжній трафік надійно показує, яким шляхом проходить запит. Книжка Google про Site Reliability Engineering доходить того самого висновку з іншого боку: синтетичне навантаження не завжди показує, де компоненти насправді ламаються, тож коли сервіс уже захищений від перевантаження, книжка радить проводити тести на відмову на невеликій частині продакшну, тримаючи напоготові додаткову потужність на випадок, якщо захист не спрацює і перемикатися доведеться вручну.

І нарешті, вона змінює стимули. README Chaos Monkey формулює це просто: коли інженери частіше стикаються зі збоями, у них з'являється причина будувати стійкі сервіси. Команда, чиї екземпляри можуть бути завершені будь-якого робочого дня, від початку проєктує з урахуванням цього, а не виявляє прогалину того дня, коли машина без попередження перестає існувати.

Як це виглядає на практиці

Хаос-експеримент записують до запуску, як план тестування. Ось приклад для вигаданого інтернет-магазину, усі числа в ньому придумані:

```
Експеримент:   платіжний провайдер сповільнюється
Усталений:     частка успішних оплат і p99 затримки оплати, за 5 хв
Гіпотеза:      з додатковими 300 мс на виклики платіжного провайдера
               частка успішних оплат в експериментальній групі
               відрізняється від контрольної не більш ніж на 0,2 п. п.
Групи:         по 2% трафіку оплат у кожній, розподіл випадковий
Змінна:        300 мс затримки в платіжному клієнті, більше нічого
Зупинити, якщо: різниця понад 1 п. п. протягом 2 хвилин поспіль
Вікно:         30 хв у робочий час, черговий інженер попереджений
Результат:     гіпотезу спростовано; повторні спроби потроїли виклики
               провайдера, і p99 затримки перевищила тайм-аут в 1 с
Наслідки:      бюджет повторних спроб на запит; потім повторити тест
```

Спростована гіпотеза тут і є корисним результатом. Слабке місце знайдено на 2% трафіку за кілька хвилин, з наперед погодженою умовою зупинки, а наслідком стає інженерна зміна, а не вибачення.

Саму змінну часто вносять звичайними інструментами. У Linux мережевий емулятор netem, який налаштовують через tc, відтворює властивості реальних мереж, зокрема затримку, втрату, дублювання й пошкодження пакетів. Наприклад:

```
# експеримент 1: додати 100 мс затримки кожному пакету, що виходить з eth0
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root

# експеримент 2: випадково відкидати один пакет із тисячі
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

Найважливіший тут рядок із `del`, і його місце — у кроці прибирання, який виконується незалежно від того, чи експеримент пройшов, провалився, чи впав на півдорозі. Netem також приймає `seed`, щоб випадкову втрату можна було відтворити, а його довідка попереджає: щоб результати вимірювання швидкодії TCP були реалістичними, netem треба розміщувати на вхідному трафіку хоста, що отримує пакети.

На рівні цілих машин Chaos Monkey показує, як виглядають обережні типові налаштування. Одразу після встановлення `enabled` має значення false, а `leashed` — true, тобто завершення лише імітуються й записуються в журнал. Коли його вмикають, він працює з групами екземплярів (за застосунком, стеком або кластером), із середнім інтервалом між завершеннями у два дні на групу й мінімальним в один день, лише в робочі дні й лише з 9:00 до 15:00 у налаштованому часовому поясі. Облікові записи, регіони й стеки можна виключити, а в конфігурації є місце для outage checker, який повідомляє інструменту, чи триває зараз аварія; реалізації, що постачаються з ним, нічого не роблять, тож цю перевірку має забезпечити сама команда експлуатації. Перелік тестів на відмову в продакшні з книжки SRE не менш конкретний: швидко чи повільно зменшувати кількість завдань, одразу втратити потужність цілого кластера й змусити бекенди мовчки поглинати запити, зокрема некритичні, чия тиша однаково може загальмувати сервіс, що їх викликає.

На цьому сайті кейс про те, як доведено, що блокування вторгнень працює за кожного запуску, від вересня 2026 року, не є хаос-експериментом у повному сенсі, але побудований за тією самою схемою. Кожен запуск посилення захисту блокує адресу з діапазону, який стандарти резервують для документації, зчитує створене правило з ядра, а не покладається на

власний стан служби, і знімає блокування в блоці очищення, що виконується незалежно від результату перевірки. Оскільки тестова адреса нікому не належить, випадкове блокування, яке залишиться після невдалого запуску, не відріже справжню мережу. Сторінка відверто називає ціну: записувати правило у виробниче середовище за кожного запуску — це справжнє втручання в робочу систему, на яке пішли, бо вже було показано, що альтернатива гірша. Реалізація резервного копіювання та тестового відновлення застосовує цю ідею до відновлення на рівні схеми: щомісячний процес відновлює відомий файл у тимчасове місце й порівнює його байти. У записі про реалізацію від 2026-10-27 зазначено, що таймери були вимкнені за відсутності налаштованого репозиторію; жодного завершеного реального відновлення не зафіксовано. Процес існує в коді; успішне практичне випробування відновлення залишається окремим результатом, який ще потрібно забезпечити.

Де виникають проблеми

Ламати без гіпотези. Вимкнути випадковий сервер і подивитися, що буде, — це демонстрація, а не експеримент. Спершу запишіть усталений стан, гіпотезу й умову зупинки. Якщо команда не бачить свого усталеного стану на дашборді в реальному часі, виправити треба саме це, а експеримент зачекає.

Дивитися на нутрощі, а не на вихід. Навантаження процесора на одному хості чи довжина однієї черги мало що кажуть про те, чи помітили щось користувачі. Принципи вимагають вимірюваного виходу всієї системи: пропускної здатності, частки помилок, перцентилів затримки, бо саме про це гіпотеза.

Необмежений радіус ураження. Принципи допускають певний короткочасний негативний ефект, але покладають на інженера обов’язок тримати наслідки малими й локалізованими. Почніть з одного екземпляра чи невеликого відсотка трафіку, майте аварійний вимикач, що працює без інструмента, який спричинив шкоду, і ніколи не запускайте експеримент поверх інциденту, що вже триває.

Не виходити зі стейджингу. Тестове середовище — розумний перший крок і поганий останній, бо його трафік, дані й сусіди не такі, як у продакшні. Переходьте на невелику частку справжнього трафіку, щойно експеримент чисто пройшов до продакшну, а не вважайте результати стейджингу остаточною відповіддю.

Запустити один раз. Систему, яка витримала експеримент у березні, відтоді змінювали багато разів. Принципи називають ручні експерименти трудомісткими й зрештою нежиттєздатними та закликають автоматизувати їх для безперервного запуску, щоб втрата стійкості виявлялася того самого тижня, коли її внесли.

Лише вбивати машини. Чисті падіння — найпростіший випадок, бо їх хтось помічає. Принципи також називають програмні збої на кшталт неправильно сформованих відповідей і події, що не є збоями, як-от сплеск трафіку чи масштабування, а книжка SRE додає бекенд, який приймає запити й ніколи не відповідає, що однаково шкодить сервісу, який його викликає, коли запити мають довгі дедлайни. Обирайте змінні за ймовірним впливом або за тим, як часто вони трапляються.

Висновки, що нікуди не ведуть. Спростована гіпотеза — це слабке місце з ім'ям. Призначте йому відповідального й пріоритет у тому самому трекері, що й решту роботи, а після виправлення повторіть експеримент і переконайтеся, що усталений стан тепер тримається.

Що почитати далі

- Principles of Chaos Engineering — визначення, чотири кроки експерименту й п'ять розширених принципів, востаннє оновлено в березні 2019 року.
- Chaos Monkey — документація Netflix до інструмента, що завершує екземпляри; потребує Spinnaker.
- Chaos Monkey: configuring behavior via Spinnaker — частота завершень, групування й винятки разом зі значеннями за замовчуванням.
- Chaos Monkey: configuration file format — усі параметри та їхні типові значення, зокрема `leashed` і вікно робочих годин.
- Addressing Cascading Failures — розділ 22 книжки Google про Site Reliability Engineering, зокрема про тестування до відмови й тести на відмову в продакшні.
- tc-netem(8) — сторінка довідки Linux про емулятор мережі, яким вносять затримку, втрату й пошкодження пакетів.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке хаос-інженерія?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/chaos-engineering/