Engineer

Навантажувальне тестування

перевірка системи під змодельованим потоком запитів

Що це таке

Навантажувальне тестування подає на систему заплановану кількість змодельованого трафіку і вимірює, як система поводиться, поки його несе: скільки часу займають відповіді, скільки запитів завершуються помилкою, яку пропускну здатність вона витримує і на що в цей час витрачають ресурси сервери. Трафік генерує скрипт, який грає роль багатьох користувачів одночасно; інструмент k6 називає кожного з них віртуальним користувачем. Навантажувальне тестування — це діяльність, а навантажувальний тест — один її запуск, із формою, тривалістю і критеріями успіху. Англійською це називають load test.

Термін вживають і широко, і вузько. Документація k6, яку видає Grafana Labs, розрізняє тести за навантаженням, яке вони створюють. Короткий перевірочний тест (smoke test) запускає кількох віртуальних користувачів, щоб перевірити, що скрипт працює. Тест із середнім навантаженням (average-load test) моделює звичайний день у робочому середовищі. Стрес-тест виходить за межі середнього, тест на витривалість (soak test) тримає середнє навантаження годинами, тест на сплеск навантаження (spike test) дає раптовий і дуже сильний сплеск на кілька хвилин, а тест граничного навантаження (breakpoint test) підвищує навантаження, доки щось не зламається. Оскільки англійський термін може означати будь-який тест, що моделює трафік, k6 називає вузький різновид average-load test і зазначає, що деякі команди кажуть day-in-life test або volume test. Посібник відверто визнає плутанину в назвах: «немає навіть згоди щодо назв цих типів тестів». Категорії до того ж відносні. Стрес-тест для одного застосунку може бути average-load test для іншого.

Навантажувальний тест — не функціональний тест. Він не перевіряє, чи правильно сторінка оформлення замовлення підсумовує кошик; він перевіряє, чи вона ще відповідає і як швидко, поки нею користується

реалістична кількість людей. Одне без іншого не працює: скрипт, що на великій швидкості надсилає зламані запити, вимірює лише те, як швидко система каже «ні».

Застосуйте поняття до рішення

Застосуйте «Що таке навантажувальне тестування?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Одна команда веде ідею від ескізу до продукту, у який ваші клієнти вже можуть увійти, а не перекидає роботу між трьома підрядниками.

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. Збільшила дохід GIS-застосунку в 10 разів — 10x зростання виторгу завдяки картографічному продукту

Збільшила дохід GIS-застосунку в 10 разів Зв'язатися з нами →

Чому це важливо

Кожна система має рівень трафіку, на якому вона перестає встигати, і питання лише в тому, хто знайде його першим. Locust, інструмент для навантажувального тестування з відкритим кодом, тести для якого пишуть звичайним Python, показує цю картину у своєму короткому вступному посібнику (quickstart). Коли інструмент додає користувачів, настає момент, у прикладі близько 20 користувачів, коли час відповіді зростає так швидко, що кількість запитів за секунду перестає збільшуватися, хоча користувачі все прибувають. Документація називає ціль перевантаженою, або насиченою. Цей злам і є справжньою пропускною здатністю системи в тому вигляді, в якому її збудовано, хоч би що обіцяла архітектурна схема, і знайти його на генеральній репетиції значно дешевше, ніж перед клієнтами.

Саме наростання навантаження до зламу теж багато про що говорить. Посібник k6 з average-load tests каже, що перший результат з'являється, поки навантаження ще зростає: чи погіршується час відповіді в міру зростання трафіку і чи не відмовляє система ще до того, як вийде на повне навантаження. Плато потім показує, чи залишаються стабільними продуктивність і використання ресурсів, бо деякі системи стають непередбачуваними саме на цьому відрізку.

Навантажувальні тести також перетворюють продуктивність із думки на перевірку. k6 дозволяє скрипту оголосити thresholds, пороги «пройдено» чи «не пройдено» для його метрик, наприклад, що з помилкою завершується менше 1% запитів або що 95% відповідей надходять протягом 200 мс.

Документація k6 зазначає, що тестувальники часто записують порогами свої цілі рівня обслуговування (SLO). Коли всі пороги виконано, k6 завершується з кодом виходу 0, а коли якийсь не виконано, з ненульовим кодом, тож конвеєр може відхилити реліз, який сповільнив сервіс, і нікому не доведеться читати графік.

Як це виглядає на практиці

Візьмімо інтернет-магазин, що готується до розпродажу. Більшість трафіку під час розпродажу — це люди, яким товар не дістанеться, тож відповідь, яка мусить витримати, — «розпродано», і саме цей шлях треба відрепетирувати першим. Тест нижче написано на k6. Він запускає запити із заданою частотою, а не із заданою кількістю користувачів, за три хвилини нарощує частоту до 2 000 запитів за секунду, тримає її двадцять хвилин і знову знижує. Форма відповідає порадам k6 для average-load tests: наростання займає від 5 до 15% тесту, плато триває щонайменше вп'ятеро довше за наростання, а спад приблизно стільки ж, скільки наростання. Spike test, який k6 радить для продажу квитків, запуску продуктів і сезонних розпродажів, ішов би потім, майже без наростання; цей спершу доводить, що відповідь «розпродано» тримається за рівномірної частоти.

```
import http from 'k6/http';

export const options = {
  scenarios: {
    sold_out: {
      executor: 'ramping-arrival-rate',
      startRate: 0,
      timeUnit: '1s',
      preAllocatedVUs: 400,
      stages: [
        { target: 2000, duration: '3m' },
        { target: 2000, duration: '20m' },
        { target: 0, duration: '3m' },
      ],
    },
  },
  thresholds: {
    http_req_failed: ['rate<0.01'],
    http_req_duration: ['p(95)<200'],
    dropped_iterations: ['count<1'],
  },
};
```

```
export default function () {
  http.get('https://shop.example/sale/items/4711');
}
```

Запустіть його командою `k6 run sold-out.js`. Кількість віртуальних користувачів — це оцінка, для якої документація k6 дає формулу: медіанний час ітерації, помножений на частоту, плюс запас. Якщо частота становить 2 000 запитів за секунду, а кожен запит триває близько десятої частки секунди, одночасно зайнято 200 віртуальних користувачів, а 400 залишають резерв. Якщо відповіді сповільнюються, кожна ітерація триває довше, пул вичерпується, і k6 рахує ітерації, які не зміг запустити, у `dropped_iterations`. Тому й потрібен третій поріг: пропущена ітерація означає, що тест не надіслав того навантаження, яке заявляє, а k6 зазначає, що пропущені ітерації можуть також свідчити про погіршення продуктивності системи. За двома іншими порогами запуск вважається невдалим, якщо помилкою завершується щонайменше 1% запитів або 95-й процентиль становить щонайменше 200 мс.

Сценарій узято з нотатки на цьому сайті про те, як продати останній квиток рівно один раз. З мільйоном покупців і десятьма тисячами одиниць 990 000 запитів мусять отримати відмову, наголошує вона, тож відмова має бути найдешевшою відповіддю системи. Розділ про те, як довести, що задум працює, вимагає перевірити відмови під навантаженням: шлях «розпродано» мусить лишатися швидким, коли це 99% трафіку, а периферія мережі мусить його витримати. Тест, який репетирує лише вдалу покупку, вимірює той один відсоток, що найменше важить для пропускної здатності.

Де виникають проблеми

Тест чекає на систему, яку вимірює. У тому, що k6 називає закритою моделлю, кожен віртуальний користувач починає наступну ітерацію лише тоді, коли завершилася попередня. Коли сервер сповільнюється, сповільнюються й користувачі, частота надходження падає, і система отримує м’якший тест, ніж планувалося. За словами k6, у частині літератури цей ефект називають coordinated omission. Справжні відвідувачі не чекають одне на одного. Коли питання в тому, чи витримає система задану частоту запитів, використовуйте виконавця сценарію відкритої моделі, як-от `constant-arrival-rate` або `ramping-arrival-rate`, що запускає ітерації за розкладом незалежно від часу відповіді.

Вузьким місцем є генератор навантаження. Якщо машині, що запускає тест, бракує віртуальних користувачів, процесора чи пам'яті, результати описують генератор, а не ціль. k6 у більшості випадків не радить `maxVUs`, бо додавання віртуальних користувачів під час тесту коштує процесорного часу й пам'яті і може перевантажити генератор та спотворити результати. Визначте `preAllocatedVUs` заздалегідь кількома локальними пробними запусками і стежте за `dropped_iterations`. Locust розподіляє навантаження на кілька машин з одним головним процесом і багатьма робочими процесами, коли однієї машини замало.

Перший запуск — найбільший. У документації k6 сказано, що тисячі користувачів інструмента запускали навантажувальні тести, які швидко призводили до збою їхніх застосунків або staging-середовищ. Почніть зі smoke test на кілька віртуальних користувачів, щоб перевірити скрипт і середовище, потім запустіть average-load test і лише після нього стрес-тест; k6 називає таку послідовність вкрай важливою. Проблеми, що проявляються за малого обсягу, значно легше знайти, поки навантаження ще невелике.

Навантаження скаче вгору-вниз, як на американських гірках. Шаблон, що кілька разів зростає і падає, марнує ресурси і заважає зрозуміти, яка зміна спричинила який симптом. k6 радить один напрямок за раз: наростання, плато, спад. Також тримайте сценарій, дані й середовище однаковими між запусками, щоб два результати взагалі можна було порівняти.

Ніхто не вирішив, що означає «пройдено». Запуск, що закінчується сторінкою графіків без жодних критеріїв, кожен, хто на неї дивиться, читає по-своєму, зазвичай поблажливо. Запишіть пороги до запуску, виходячи з цілі, якої має дотримуватися сервіс, і нехай рішення ухвалює код виходу. Для довгих запусків k6 може перервати тест, щойно поріг не виконано, за допомогою `abortOnFail`, а `delayAbortEval` не дає йому зупинитися, доки не набереться достатньо вимірів, щоб вони щось означали.

Один тип тесту вважають повною відповіддю. Average-load test мало що каже про розпродаж, який налітає за секунди, а spike test мало що каже про ваду, яка проявляється лише після годин рівномірного використання. k6 прямо пише, що жоден окремий тип тесту не усуває всього ризику і що суміш визначає профіль ризику: системі, якій загрожує тривале використання, спершу потрібні soak tests, а системі, якій загрожує інтенсивне використання, потрібні стрес-тести. Сплеск може також запустити процеси, які ніколи не працюють у звичайний день, тож скрипт мусить їх охопити.

Що почитати далі

- Load test types — огляд k6 тестів smoke, average-load, stress, soak, spike і breakpoint із типовим для кожного навантаженням і тривалістю.
- Average-load testing — як формувати наростання, плато і спад і що можна прочитати з кожної частини.
- Thresholds — критерії k6 «пройдено» і «не пройдено», їхній синтаксис і те, як невиконаний поріг задає код виходу.
- Open and closed models — чому тест, користувачі якого чекають на відповіді, показує повільну систему кращою, ніж вона є.
- Arrival-rate VU allocation — скільки віртуальних користувачів потрібно тесту з відкритою моделлю і що означають пропущені ітерації.
- What is Locust? — власний опис Locust у його документації: інструмент для навантажувального тестування на Python і його розподілений режим.



Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке навантажувальне тестування?» 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/load-test/