Engineer

Belastningstest

en test, der måler et system under simuleret trafik

Hvad det er

En belastningstest sender en planlagt mængde simuleret trafik mod et system og måler, hvordan systemet opfører sig, mens det bærer den: hvor lang tid svarene tager, hvor mange forespørgsler der fejler, hvor meget gennemløb det kan holde, og hvad serverne bruger kræfterne på imens. Trafikken kommer fra et script, der spiller mange brugere på én gang; værktøjet k6 kalder hver af dem en virtuel bruger. Belastningstestning er aktiviteten, og en belastningstest er én kørsel af den, med en form, en varighed og kriterier for at bestå. På engelsk hedder det load test.

Ordet bruges både bredt og snævert. Dokumentationen til k6, som Grafana Labs udgiver, sorterer test efter den belastning, de lægger på. En smoke test kører en håndfuld virtuelle brugere for at bevise, at scriptet virker. En average-load test simulerer en typisk dag i produktion. En stresstest går over gennemsnittet, en soak test holder gennemsnitlig belastning i timevis, en spike test sender et pludseligt og meget kraftigt udbrud i nogle få minutter, og en breakpoint test øger belastningen, indtil noget giver efter. Fordi den engelske betegnelse kan dække enhver test, der simulerer trafik, kalder k6 den snævre slags en average-load test og bemærker, at nogle hold siger day-in-life test eller volume test. Guiden er ærlig om ordforrådet: »no consensus even exists about the names of these test types«. Kategorierne er også relative. En stresstest for én applikation er en average-load test for en anden.

En belastningstest er ikke en funktionel test. Den spørger ikke, om kassen lægger indkøbskurven rigtigt sammen; den spørger, om kassen stadig svarer, og hvor hurtigt, mens en realistisk mængde mennesker bruger den. De to har brug for hinanden: et script, der sender ødelagte forespørgsler i højt tempo, måler kun, hvor hurtigt systemet siger nej.

Brug begrebet i en beslutning

Brug »Hvad er en belastningstest?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Ét team, der tager en idé fra skitse til noget, jeres kunder kan logge ind på, i stedet for at kaste den mellem tre leverandører.

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 ført en GIS-kortapp til 10x omsætningsvækst — 10x omsætningsvækst, drevet af kortproduktet

Har ført en GIS-kortapp til 10x omsætningsvækst Kontakt os →

Hvorfor det betyder noget

Ethvert system har et trafikniveau, hvor det ikke længere kan følge med, og spørgsmålet er kun, hvem der finder det først. Locust, et open source-værktøj til belastningstest, hvor testene skrives i almindelig Python, viser formen i sin quickstart. Efterhånden som værktøjet tilføjer brugere, kommer der et punkt, omkring 20 brugere i eksemplet, hvor svartiderne stiger så hurtigt, at antallet af forespørgsler i sekundet holder op med at vokse, selv om der bliver ved med at komme flere brugere til. Dokumentationen kalder målet overbelastet eller mættet. Det knæk er systemets reelle kapacitet, som det er bygget, uanset hvad arkitekturdiagrammet lover, og det er langt billigere at finde ved en generalprøve end foran kunderne.

Selve optrapningen mod knækket fortæller også noget. k6's guide til average-load tests siger, at det første resultat viser sig, mens belastningen stadig stiger: om svartiden forringes, efterhånden som trafikken vokser, og om systemet fejler, før det overhovedet når fuld belastning. Plateauet viser derefter, om ydelse og ressourceforbrug forbliver stabile, for nogle systemer bliver uberegnelige netop på den strækning.

Belastningstests gør desuden ydelse til en kontrol i stedet for en holdning. k6 lader et script erklære thresholds, tærskler for bestået eller dumpet på dets målinger, for eksempel at under 1 % af forespørgslerne fejler, eller at 95 % af svarene kommer inden for 200 ms. k6's dokumentation bemærker, at testere ofte bruger tærskler til at skrive deres service level objectives ned. Når alle tærskler holder, slutter k6 med exit-kode 0, og når en af dem fejler, slutter den med en kode, der ikke er 0, så en pipeline kan afvise en release, der gjorde tjenesten langsommere, uden at nogen skal læse en graf.

Sådan ser det ud i praksis

Tag en webshop, der gør klar til udsalg. Det meste af trafikken under et udsalg kommer fra folk, der ikke får varen, så det svar, der skal holde, er »udsolgt«, og det er den vej, der skal øves først. Testen herunder er skrevet i k6. Den starter forespørgsler i en fast takt i stedet for med et fast antal brugere, trapper op til 2.000 forespørgsler i sekundet over tre minutter, holder det i tyve og trapper ned igen. Formen følger k6's råd for average-load tests: optrapningen tager 5 til 15 % af testen, plateauet varer mindst fem gange så længe som optrapningen, og nedtrapningen omtrent lige så længe som optrapningen. En spike test, som k6 foreslår til billetsalg, produktlanceringer og sæsonudsalg, ville komme bagefter med næsten ingen optrapning; denne beviser først, at udsolgt-svaret holder ved en jævn takt.

```
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');
}
```

Kør den med `k6 run sold-out.js`. Antallet af virtuelle brugere er et skøn, som k6's dokumentation giver en formel for: den mediane iterationstid ganget med takten plus en margen. Ved 2.000 forespørgsler i sekundet og omkring en tiendedel sekund til hver er 200 virtuelle brugere i gang på samme tid, og 400

giver luft. Bliver svarene langsommere, varer hver iteration længere, puljen løber tør, og k6 tæller de iterationer, den ikke kunne starte, i `dropped_iterations`. Derfor er den tredje tærskel med: en droppet iteration betyder, at testen ikke sendte den belastning, den påstår, og k6 bemærker, at droppede iterationer også kan være tegn på, at systemets ydelse forringes. De to andre tærskler får kørslen til at fejle, hvis mindst 1 % af forespørgslerne fejler, eller hvis 95-percentilen er 200 ms eller højere.

Scenariet kommer fra sidens note om at sælge den sidste billet præcis én gang. Med en million købere og ti tusind enheder skal 990.000 forespørgsler fejle, påpeger den, så afvisningen skal være det billigste svar, systemet giver. Afsnittet om at bevise designet omfatter en belastningstest af afvisningerne: udsolgt-stien skal forblive hurtig, når den udgør 99 procent af trafikken, og kanten af netværket skal kunne tage imod den. En test, der kun øver et vellykket køb, måler den ene procent, der betyder mindst for kapaciteten.

Hvor det går galt

Testen venter på det system, den måler. I det, k6 kalder den lukkede model, starter hver virtuel bruger først sin næste iteration, når den forrige er færdig. Når serveren bliver langsommere, bliver brugerne det også, ankomsttakten falder, og systemet får en mildere test end den planlagte, en effekt, som ifølge k6 kaldes coordinated omission i noget af litteraturen. Rigtige besøgende venter ikke på hinanden. Når spørgsmålet er, om et system kan klare en given takt af forespørgsler, så brug en executor fra den åbne model, som `constant-arrival-rate` eller `ramping-arrival-rate`, der starter iterationer efter planen uanset svartiden.

Belastningsgeneratoren er flaskehalsen. Løber maskinen, der kører testen, tør for virtuelle brugere, CPU eller hukommelse, beskriver resultaterne generatoren og ikke målet. k6 fraråder `maxVUs` i de fleste tilfælde, fordi det koster CPU og hukommelse at tilføje virtuelle brugere, mens testen kører, og kan overbelaste generatoren og forvride resultaterne. Dimensionér `preAllocatedVUs` på forhånd med et par lokale prøvekørsler, og hold øje med `dropped_iterations`. Locust fordeler belastningen over flere maskiner med én master-proces og mange workers, når én maskine ikke er nok.

Den første kørsel er den største. k6's dokumentation fortæller, at tusindvis af dens brugere har kørt belastningstests, der hurtigt fik deres applikationer eller staging-miljøer til at gå ned. Start med en smoke test med nogle få virtuelle

brugere for at bevise script og miljø, kør så en average-load test, og først derefter en stresstest; k6 kalder den rækkefølge afgørende. Problemer, der viser sig ved lav volumen, er langt lettere at finde, før belastningen er høj.

Belastningen går op og ned som en rutsjebane. Et mønster, der stiger og falder flere gange, spilder ressourcer og gør det svært at se, hvilken ændring der gav hvilket symptom. k6 anbefaler én retning ad gangen: trap op, hold et plateau, trap ned. Hold også scenarie, data og miljø ens fra kørsel til kørsel, så to resultater overhovedet kan sammenlignes.

Ingen har besluttet, hvad bestået betyder. En kørsel, der ender med en side fuld af grafer og ingen kriterier, bliver læst forskelligt af alle, der ser på den, som regel velvilligt. Skriv tærsklerne før kørslen ud fra det mål, tjenesten holdes op på, og lad exit-koden afgøre sagen. Ved lange kørsler kan k6 afbryde, så snart en tærskel fejler, med `abortOnFail`, og `delayAbortEval` forhindrer, at den stopper, før der er kommet nok målinger til, at de betyder noget.

Én testtype bliver behandlet som hele svaret. En average-load test siger ikke meget om et udsalg, der rammer på få sekunder, og en spike test siger ikke meget om en fejl, der først viser sig efter timers jævn brug. k6 siger klart, at ingen enkelt testtype fjerner al risiko, og at risikoprofilen afgør blandingen: et system, der er i fare ved langvarig brug, har først brug for soak tests, og et, der er i fare ved intensiv brug, har brug for stresstests. En spike kan også sætte processer i gang, som aldrig kører på en almindelig dag, så scriptet skal have dem med.

Læs mere

- Load test types — k6's overblik over smoke, average-load, stress, soak, spike og breakpoint tests med den belastning og varighed, der er typisk for hver.
- Average-load testing — hvordan optrapning, plateau og nedtrapning formes, og hvad man kan aflæse af hver del.
- Thresholds — k6's kriterier for bestået og dumpet, deres syntaks, og hvordan en dumpet tærskel sætter exit-koden.
- Open and closed models — hvorfor en test, hvis brugere venter på svar, får et langsomt system til at se bedre ud.
- Arrival-rate VU allocation — hvor mange virtuelle brugere en test med åben model har brug for, og hvad droppede iterationer betyder.

- What is Locust? — Locust-dokumentationens egen beskrivelse af et værktøj til belastningstest i Python og dets distribuerede tilstand.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en belastningstest?« 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/load-test/