Engineer

Lasttest

en test som måler et system under simulert trafikk

Hva det er

En lasttest sender en planlagt mengde simulert trafikk mot et system og måler hvordan systemet oppfører seg mens det bærer den: hvor lang tid svarene tar, hvor mange forespørsler som feiler, hvor mye gjennomstrømning det klarer å holde, og hva serverne bruker kreftene på imens. Trafikken kommer fra et skript som spiller mange brukere samtidig; verktøyet k6 kaller hver av dem en virtuell bruker. Lasttesting er aktiviteten, og en lasttest er én kjøring av den, med en form, en varighet og kriterier for å bestå. På engelsk heter det load test.

Ordet brukes både vidt og snevert. Dokumentasjonen til k6, som Grafana Labs gir ut, sorterer tester etter lasten de legger på. En smoke test kjører en håndfull virtuelle brukere for å bevise at skriptet virker. En average-load test simulerer en vanlig dag i produksjon. En stresstest går over gjennomsnittet, en soak test holder gjennomsnittlig last i timevis, en spike test sender et plutselig og svært kraftig utbrudd i noen få minutter, og en breakpoint test øker lasten til noe gir etter. Fordi den engelske betegnelsen kan dekke enhver test som simulerer trafikk, kaller k6 den snevre varianten en average-load test og nevner at noen team sier day-in-life test eller volume test. Veiledningen er ærlig om ordforrådet: «no consensus even exists about the names of these test types». Kategoriene er også relative. En stresstest for én applikasjon er en average-load test for en annen.

En lasttest er ikke en funksjonell test. Den spør ikke om kassen summerer handlekurven riktig; den spør om kassen fortsatt svarer, og hvor raskt, mens en realistisk mengde mennesker bruker den. De to trenger hverandre: et skript som sender ødelagte forespørsler i høyt tempo, måler bare hvor fort systemet sier nei.

Bruk begrepet i en beslutning

Bruk «Hva er en lasttest?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Ett team som tar en idé fra skisse til noe kundene dine kan logge inn på, i stedet for at den sendes videre mellom tre leverandører.

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. Ledet en GIS-kartapp til 10 ganger mer omsetning — 10x inntektsvekst, ledet fra kartproduktet

Ledet en GIS-kartapp til 10 ganger mer omsetning Ta kontakt →

Hvorfor det er viktig

Ethvert system har et trafikknivå der det ikke lenger henger med, og spørsmålet er bare hvem som finner det først. Locust, et åpen kildekode-verktøy for lasttesting der testene skrives i vanlig Python, viser formen i sin quickstart. Etter hvert som verktøyet legger til brukere, kommer det et punkt, rundt 20 brukere i eksempelet, der svartidene stiger så fort at antallet forespørsler per sekund slutter å vokse, selv om det stadig kommer flere brukere til. Dokumentasjonen kaller målet overbelastet eller mettet. Det knekkpunktet er den reelle kapasiteten til systemet slik det er bygget, uansett hva arkitekturtegningen lover, og det er langt billigere å finne på en generalprøve enn foran kundene.

Selve opptrappingen mot knekkpunktet forteller også noe. k6 sin veiledning for average-load tests sier at det første resultatet viser seg mens lasten fortsatt stiger: om svartiden blir dårligere etter hvert som trafikken vokser, og om systemet feiler før det i det hele tatt når full last. Platået viser deretter om ytelse og ressursbruk holder seg stabile, for noen systemer blir uforutsigbare akkurat på den strekningen.

Lasttester gjør dessuten ytelse til en kontroll i stedet for en mening. k6 lar et skript erklære thresholds, terskler for bestått eller ikke bestått på målingene, for eksempel at under 1 % av forespørslene feiler, eller at 95 % av svarene kommer innen 200 ms. Dokumentasjonen til k6 påpeker at testere ofte bruker terskler til å skrive ned sine tjenestenivåmål. Når alle terskler holder, avslutter k6 med exit-kode 0, og når en av dem feiler, avslutter den med en kode som ikke er 0, slik at en pipeline kan stoppe en release som gjorde tjenesten tregere, uten at noen må lese en graf.

Slik ser det ut i praksis

Ta en nettbutikk som gjør seg klar til salg. Det meste av trafikken under et salg kommer fra folk som ikke får varen, så svaret som må holde, er «utsolgt», og det er den veien som må øves først. Testen under er skrevet i k6. Den starter forespørsler i et fast tempo i stedet for med et fast antall brukere, trapper opp til 2 000 forespørsler i sekundet over tre minutter, holder det i tjue og trapper ned igjen. Formen følger rådet fra k6 for average-load tests: opptrappingen tar 5 til 15 % av testen, platået varer minst fem ganger så lenge som opptrappingen, og nedtrappingen omtrent like lenge som opptrappingen. En spike test, som k6 foreslår for billettsalg, produktlanseringer og sesongsalg, ville komme etterpå med nesten ingen opptrapping; denne beviser først at utsolgt-svaret holder i et jevnt tempo.

```
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');
}
```

Kjør den med `k6 run sold-out.js`. Antallet virtuelle brukere er et anslag som dokumentasjonen til k6 gir en formel for: median iterasjonstid ganget med tempoet, pluss en margin. Ved 2 000 forespørsler i sekundet og rundt en tidel sekund på hver er 200 virtuelle brukere i arbeid samtidig, og 400 gir rom. Blir

svarene tregere, varer hver iterasjon lenger, poolen går tom, og k6 teller iterasjonene den ikke klarte å starte, i `dropped_iterations`. Derfor er den tredje terskelen med: en droppet iterasjon betyr at testen ikke sendte den lasten den hevder, og k6 påpeker at droppede iterasjoner også kan tyde på at ytelsen til systemet blir dårligere. De to andre tersklene underkjenner kjøringen hvis minst 1 % av forespørslene feiler, eller hvis 95-persentilen er 200 ms eller høyere.

Scenarioet kommer fra notatet her på nettstedet om å selge den siste billetten nøyaktig én gang. Med en million kjøpere og ti tusen enheter må 990 000 forespørsler mislykkes, påpeker det, så avvisningen må være det billigste svaret systemet gir. Delen om å bevise designet ber om lasttesting av avvisningene: utsolgt-stien må holde seg rask når den er 99 % av trafikken, og kanten av nettverket må ta imot den. En test som bare øver på et vellykket kjøp, måler den ene prosenten som betyr minst for kapasiteten.

Hvor det går galt

Testen venter på systemet den måler. I det k6 kaller den lukkede modellen, starter hver virtuelle bruker neste iterasjon først når den forrige er ferdig. Når serveren blir tregere, blir brukerne det også, ankomsttempoet faller, og systemet får en mildere test enn den som var planlagt, en effekt som ifølge k6 kalles coordinated omission i deler av litteraturen. Ekte besøkende venter ikke på hverandre. Når spørsmålet er om et system tåler et gitt tempo av forespørsler, bruk en executor fra den åpne modellen, som `constant-arrival-rate` eller `ramping-arrival-rate`, som starter iterasjoner etter planen uansett svartid.

Lastgeneratoren er flaskehalsen. Går maskinen som kjører testen, tom for virtuelle brukere, CPU eller minne, beskriver resultatene generatoren og ikke målet. k6 fraråder `maxVUs` i de fleste tilfeller, fordi det koster CPU og minne å legge til virtuelle brukere mens testen kjører, og det kan overbelaste generatoren og forvrenge resultatene. Dimensjoner `preAllocatedVUs` på forhånd med noen lokale prøvekjøringer, og følg med på `dropped_iterations`. Locust fordeler lasten over flere maskiner med én master-prosess og mange workers når én maskin ikke er nok.

Den første kjøringen er den største. Dokumentasjonen til k6 forteller at tusenvis av brukerne har kjørt lasttester som raskt fikk applikasjonene eller staging-miljøene deres til å gå ned. Start med en smoke test med noen få

virtuelle brukere for å bevise skript og miljø, kjør så en average-load test, og først deretter en stresstest; k6 kaller den rekkefølgen avgjørende. Problemer som viser seg ved lavt volum, er mye lettere å finne før lasten er høy.

Lasten går opp og ned som en berg-og-dal-bane. Et mønster som stiger og faller flere ganger, sløser med ressurser og gjør det vanskelig å se hvilken endring som ga hvilket symptom. k6 anbefaler én retning om gangen: trapp opp, hold et platå, trapp ned. Hold også scenario, data og miljø likt fra kjøring til kjøring, slik at to resultater i det hele tatt kan sammenlignes.

Ingen har bestemt hva bestått betyr. En kjøring som ender med en side full av grafer og ingen kriterier, blir lest ulikt av alle som ser på den, som regel velvillig. Skriv tersklene før kjøringen ut fra målet tjenesten holdes til, og la exit-koden avgjøre. Ved lange kjøringer kan k6 avbryte så snart en terskel feiler, med `abortOnFail`, og `delayAbortEval` hindrer at den stopper før det har kommet nok målinger til at de betyr noe.

Én testtype blir behandlet som hele svaret. En average-load test sier lite om et salg som kommer på få sekunder, og en spike test sier lite om en feil som først viser seg etter timer med jevn bruk. k6 er tydelig på at ingen enkelt testtype fjerner all risiko, og at risikoprofilen avgjør blandingen: et system som er utsatt ved langvarig bruk, trenger soak tests først, og et som er utsatt ved intensiv bruk, trenger stresstester. En spike kan også sette i gang prosesser som aldri kjører på en vanlig dag, så skriptet må ta dem med.

Les mer

- Load test types — oversikten i k6 over smoke, average-load, stress, soak, spike og breakpoint tests, med lasten og varigheten som er typisk for hver.
- Average-load testing — hvordan opptrapping, platå og nedtrapping formes, og hva du kan lese ut av hver del.
- Thresholds — kriteriene i k6 for bestått og ikke bestått, syntaksen deres, og hvordan en terskel som feiler, setter exit-koden.
- Open and closed models — hvorfor en test der brukerne venter på svar, får et tregt system til å se bedre ut.
- Arrival-rate VU allocation — hvor mange virtuelle brukere en test med åpen modell trenger, og hva droppede iterasjoner betyr.
- What is Locust? — dokumentasjonen til Locust sin egen beskrivelse av et verktøy for lasttesting i Python og den distribuerte modusen.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en lasttest?» 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/load-test/