Engineer

Lasttest

ett test som mäter ett system under simulerad trafik

Vad det är

Ett lasttest skickar en planerad mängd simulerad trafik mot ett system och mäter hur systemet beter sig medan det bär den: hur lång tid svaren tar, hur många begäranden som misslyckas, hur stort genomflöde det klarar att hålla och vad servrarna lägger kraften på under tiden. Trafiken kommer från ett skript som spelar många användare på en gång; verktyget k6 kallar var och en av dem en virtuell användare. Lasttestning är aktiviteten, och ett lasttest är en körning av den, med en form, en längd och kriterier för att bli godkänd. På engelska heter det load test.

Ordet används både brett och snävt. Dokumentationen till k6, som Grafana Labs ger ut, sorterar tester efter den last de lägger på. Ett smoke test kör en handfull virtuella användare för att bevisa att skriptet fungerar. Ett average-load test simulerar en vanlig dag i produktion. Ett stresstest går över genomsnittet, ett soak test håller genomsnittlig last i timmar, ett spike test skickar en plötslig och mycket kraftig topp under några få minuter, och ett breakpoint test höjer lasten tills något ger vika. Eftersom den engelska termen kan täcka varje test som simulerar trafik, kallar k6 den snäva sorten ett average-load test och nämner att vissa team säger day-in-life test eller volume test. Guiden är ärlig om ordförrådet: ”no consensus even exists about the names of these test types”. Kategorierna är också relativa. Ett stresstest för en applikation är ett average-load test för en annan.

Ett lasttest är inte ett funktionellt test. Det frågar inte om kassan räknar ihop varukorgen rätt; det frågar om kassan fortfarande svarar, och hur snabbt, medan en realistisk mängd människor använder den. De två behöver varandra: ett skript som skickar trasiga begäranden i högt tempo mäter bara hur fort systemet säger nej.

Använd begreppet i ett beslut

Använd ”Vad är ett lasttest?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Ett team som tar en idé från en skiss till något dina kunder kan logga in i, i stället för att lämna den vidare mellan tre leverantörer.

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. Ledde GIS-kartappen till 10 gånger högre intäkter — 10x intäktstillväxt, med kartprodukten i spetsen

Ledde GIS-kartappen till 10 gånger högre intäkter Hör av er →

Varför det spelar roll

Varje system har en trafiknivå där det inte längre hinner med, och frågan är bara vem som hittar den först. Locust, ett lasttestverktyg med öppen källkod där testerna skrivs i vanlig Python, visar formen i sin quickstart. Allt eftersom verktyget lägger till användare kommer en punkt, runt 20 användare i exemplet, där svarstiderna stiger så fort att antalet begäranden per sekund slutar öka, trots att fler användare hela tiden tillkommer. Dokumentationen kallar målet överbelastat eller mättat. Den knäcken är systemets verkliga kapacitet som det är byggt, oavsett vad arkitekturskissen lovar, och det är mycket billigare att hitta den vid en generalrepetition än framför kunderna.

Själva upprampningen mot knäcken säger också något. k6:s guide till average-load tests säger att det första resultatet syns medan lasten fortfarande ökar: om svarstiden försämras när trafiken växer, och om systemet fallerar innan det ens når full last. Platån visar sedan om prestanda och resursanvändning håller sig stabila, eftersom vissa system blir oberäkneliga just under den sträckan.

Lasttester gör dessutom prestanda till en kontroll i stället för en åsikt. k6 låter ett skript deklarera thresholds, tröskelvärden för godkänt eller underkänt på dess mätvärden, till exempel att färre än 1 % av begärandena misslyckas eller att 95 % av svaren kommer inom 200 ms. k6:s dokumentation påpekar att testare ofta använder tröskelvärden för att skriva ned sina servicenivåmål. När alla tröskelvärden håller avslutar k6 med slutkod 0, och när något av dem fallerar avslutar det med en kod som inte är 0, så att en pipeline kan stoppa en release som gjorde tjänsten långsammare, utan att någon behöver läsa en graf.

Så ser det ut i praktiken

Ta en webbutik som förbereder en rea. Det mesta av trafiken under en rea kommer från folk som inte får varan, så svaret som måste hålla är ”slutsålt”, och det är den vägen som ska övas först. Testet nedan är skrivet i k6. Det startar begäranden i en fast takt i stället för med ett fast antal användare, rampar upp till 2 000 begäranden per sekund under tre minuter, håller det i tjugo och rampar ned igen. Formen följer k6:s råd för average-load tests: upprampningen tar 5 till 15 % av testet, platån varar minst fem gånger så länge som upprampningen och nedrampningen ungefär lika länge som upprampningen. Ett spike test, som k6 föreslår för biljettsläpp, produktlanseringar och säsongsreor, skulle komma efteråt med nästan ingen upprampning alls; det här bevisar först att svaret slutsålt håller i en jämn 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');
}
```

Kör det med `k6 run sold-out.js`. Antalet virtuella användare är en uppskattning som k6:s dokumentation ger en formel för: medianen av iterationstiden gånger takten, plus en marginal. Vid 2 000 begäranden per sekund och ungefär en tiondels sekund för varje är 200 virtuella användare upptagna samtidigt, och 400

ger utrymme. Blir svaren långsammare varar varje iteration längre, poolen tar slut och k6 räknar de iterationer det inte kunde starta i `dropped_iterations`. Därför finns det tredje tröskelvärdet med: en tappad iteration betyder att testet inte skickade den last det påstår, och k6 påpekar att tappade iterationer också kan tyda på att systemets prestanda försämras. De andra två tröskelvärdena underkänner körningen om minst 1 % av begärandena misslyckas eller om 95:e percentilen är 200 ms eller högre.

Scenariot kommer från anteckningen här på webbplatsen om att sälja den sista biljetten exakt en gång. Med en miljon köpare och tiotusen enheter måste 990 000 begäranden misslyckas, påpekar den, så avvisningen måste vara det billigaste svar systemet ger. Avsnittet om att bevisa designen tar upp lasttestning av avvisningarna: vägen för slutsålt måste förbli snabb när den är 99 % av trafiken, och edge-nätverket måste ta upp den. Ett test som bara övar ett lyckat köp mäter den enda procent som betyder minst för kapaciteten.

Var det går fel

Testet väntar på systemet det mäter. I det k6 kallar den stängda modellen startar varje virtuell användare sin nästa iteration först när den förra är klar. När servern blir långsammare blir användarna det också, ankomsttakten sjunker och systemet får ett mildare test än det planerade, en effekt som enligt k6 kallas coordinated omission i delar av litteraturen. Riktiga besökare väntar inte på varandra. När frågan är om ett system klarar en given takt av begäranden, använd en executor från den öppna modellen, som `constant-arrival-rate` eller `ramping-arrival-rate`, som startar iterationer enligt schemat oavsett svarstid.

Lastgeneratorn är flaskhalsen. Får maskinen som kör testet slut på virtuella användare, CPU eller minne beskriver resultaten generatorn och inte målet. k6 avråder från `maxVUs` i de flesta fall, eftersom det kostar CPU och minne att lägga till virtuella användare medan testet kör, och det kan överbelasta generatorn och snedvrida resultaten. Dimensionera `preAllocatedVUs` i förväg med några lokala provkörningar och håll ett öga på `dropped_iterations`. Locust fördelar lasten över flera maskiner med en master-process och många workers när en maskin inte räcker.

Den första körningen är den största. k6:s dokumentation berättar att tusentals av dess användare har kört lasttester som snabbt fick deras applikationer eller staging-miljöer att gå ned. Börja med ett smoke test med några få virtuella

användare för att bevisa skript och miljö, kör sedan ett average-load test, och först därefter ett stresstest; k6 kallar den ordningen avgörande. Problem som syns vid låg volym är mycket lättare att hitta innan lasten är hög.

Lasten går upp och ned som en berg-och-dalbana. Ett mönster som stiger och sjunker flera gånger slösar resurser och gör det svårt att se vilken ändring som gav vilket symtom. k6 rekommenderar en riktning i taget: rampa upp, håll en platå, rampa ned. Håll också scenario, data och miljö lika mellan körningarna, så att två resultat över huvud taget går att jämföra.

Ingen har bestämt vad godkänt betyder. En körning som slutar med en sida full av diagram och inga kriterier läses olika av alla som tittar på den, oftast välvilligt. Skriv tröskelvärdena före körningen utifrån det mål tjänsten hålls till, och låt slutkoden avgöra. Vid långa körningar kan k6 avbryta så fort ett tröskelvärde fallerar med `abortOnFail`, och `delayAbortEval` hindrar att det stoppar innan tillräckligt många mätvärden har kommit in för att betyda något.

En testtyp behandlas som hela svaret. Ett average-load test säger lite om en rea som kommer på några sekunder, och ett spike test säger lite om ett fel som först visar sig efter timmar av jämn användning. k6 är tydligt med att ingen enskild testtyp tar bort all risk, och att riskprofilen avgör blandningen: ett system som är utsatt vid långvarig användning behöver soak tests först, och ett som är utsatt vid intensiv användning behöver stresstester. En topp kan också sätta i gång processer som aldrig körs en vanlig dag, så skriptet måste ta med dem.

Läs mer

- Load test types — k6:s översikt över smoke, average-load, stress, soak, spike och breakpoint tests, med den last och längd som är typisk för var och en.
- Average-load testing — hur upprampning, platå och nedrampning formas och vad du kan läsa ut av varje del.
- Thresholds — k6:s kriterier för godkänt och underkänt, deras syntax och hur ett underkänt tröskelvärde sätter slutkoden.
- Open and closed models — varför ett test där användarna väntar på svar får ett långsamt system att se bättre ut.
- Arrival-rate VU allocation — hur många virtuella användare ett test med öppen modell behöver och vad tappade iterationer betyder.
- What is Locust? — Locust-dokumentationens egen beskrivning av ett lasttestverktyg i Python och dess distribuerade läge.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ett lasttest?” 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/load-test/