Engineer

Test de charge

Test mesurant un système sous un trafic simulé

Ce que c’est

Un test de charge envoie à un système un volume prévu de trafic simulé et mesure son comportement sous cette charge : délai des réponses, nombre de requêtes échouées, débit soutenu et ressources dépensées par les serveurs. Un script simule de nombreux utilisateurs à la fois ; k6 appelle chacun utilisateur virtuel. On distingue l’activité de test de charge d’une exécution donnée, caractérisée par un profil, une durée et des critères de réussite.

Le terme s’emploie au sens large et au sens étroit. La documentation k6 de Grafana Labs classe les tests selon la charge appliquée. Un test de bon fonctionnement fait tourner quelques utilisateurs virtuels pour vérifier le script. Un test de charge moyenne simule une journée typique en production. Un test de stress dépasse cette moyenne, un test d’endurance maintient la charge moyenne pendant des heures, un test de pic applique une poussée soudaine et très élevée pendant quelques minutes, et un test de rupture augmente la charge jusqu’à une défaillance. Comme « test de charge » peut désigner tout test de trafic simulé, k6 appelle la forme étroite test de charge moyenne et note que certaines équipes parlent de test d’une journée type ou de test de volume. Son guide reconnaît l’état du vocabulaire : « il n’existe même pas de consensus sur les noms de ces types de tests ». Les catégories sont aussi relatives : la charge qui stresse une application est une journée ordinaire pour une autre.

Un test de charge se distingue d’un test fonctionnel. Il ne demande pas si le processus de commande calcule correctement le total du panier, mais s’il répond encore et à quelle vitesse pendant qu’un nombre réaliste d’utilisateurs le sollicite. Les deux sont nécessaires : un script envoyant très vite des requêtes incorrectes ne mesure que la vitesse à laquelle le système refuse.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un test de charge ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une seule équipe transforme une idée en un produit auquel vos clients peuvent se connecter, sans le faire passer entre trois prestataires.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Application GIS : revenus multipliés par 10 — 10x croissance du chiffre d'affaires portée par le produit cartographique

Application GIS : revenus multipliés par 10 Contactez-nous →

Pourquoi c’est important

Tout système a un niveau de trafic au-delà duquel il ne suit plus ; la seule question est qui le trouve d’abord. Locust, outil open source dont les tests sont écrits directement en Python, montre ce profil dans son guide de démarrage. À mesure que l’outil ajoute des utilisateurs, on atteint un point, vers 20 utilisateurs dans son exemple, où le temps de réponse augmente si vite que le nombre de requêtes par seconde cesse de progresser malgré de nouvelles arrivées. La documentation appelle la cible surchargée ou saturée. Ce coude indique la capacité réelle du système tel qu’il a été construit, quelles que soient les promesses du schéma d’architecture ; le découvrir lors d’un essai coûte bien moins que devant les clients.

La montée vers ce coude renseigne déjà. Le guide k6 des tests de charge moyenne indique qu’un premier résultat apparaît avant la fin de la montée : le temps de réponse se dégrade-t-il avec le trafic, et le système échoue-t-il avant même d’atteindre la pleine charge ? Le plateau montre ensuite si performances et consommation de ressources restent stables, car certains systèmes deviennent erratiques précisément pendant cette phase.

Les tests de charge transforment aussi la performance en contrôle qualité plutôt qu’en opinion. k6 permet au script de déclarer des seuils, critères de réussite ou d’échec sur ses métriques, comme moins de 1 % de requêtes échouées ou 95 % de réponses en 200 ms au maximum. Sa documentation précise que les testeurs s’en servent souvent pour formaliser leurs objectifs de niveau de service. Si tous les seuils sont respectés, k6 se termine avec 0 ; si l’un échoue, il renvoie un code non nul. Un pipeline de traitement peut ainsi refuser une version qui a ralenti le service sans que personne ne lise un graphique.

À quoi cela ressemble en pratique

Prenez une boutique préparant une vente. La majorité du trafic provient de personnes qui n'obtiendront pas l'article : la réponse qui doit supporter la charge est donc « épuisé », et c'est le chemin à tester en premier. Le test ci-dessous est écrit en k6. Il lance les requêtes à un rythme fixé plutôt qu'avec un nombre fixe d'utilisateurs, monte à 2 000 requêtes par seconde en trois minutes, maintient ce niveau vingt minutes puis redescend. La montée suit les recommandations k6 pour la charge moyenne : elle occupe 5 à 15 % du test, le plateau dure au moins cinq fois plus longtemps, et la descente environ autant que la montée. Un test de pic, recommandé par k6 pour les ventes de billets, lancements de produits et ventes saisonnières, viendrait ensuite avec presque aucune montée ; celui-ci vérifie d'abord que la réponse « épuisé » tient à rythme stable.

```
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');
}
```

Exécutez-le avec `k6 run sold-out.js` . Le nombre d'utilisateurs virtuels est une estimation dont la documentation k6 donne la formule : temps médian d'itération multiplié par le rythme, plus une marge. À 2 000 requêtes par seconde et environ un dixième de seconde par requête, 200 utilisateurs virtuels travaillent

simultanément, et en prévoir 400 laisse de la réserve. Si les réponses ralentissent, les itérations durent davantage, le pool s'épuise et k6 enregistre dans `dropped_iterations` chaque itération sans utilisateur virtuel libre. C'est la raison du troisième seuil : une itération abandonnée signifie que le test n'a pas envoyé la charge annoncée, et k6 précise que le système lui-même peut ralentir lorsque le générateur abandonne des itérations. Les deux autres seuils font échouer l'exécution si 1 % ou plus des requêtes échouent ou si le 95e percentile atteint ou dépasse 200 ms.

Le scénario vient de la note de ce site sur la vente du dernier billet une seule fois. Avec un million d'acheteurs et dix mille unités, indique-t-elle, 990 000 requêtes doivent échouer : le refus doit donc être la réponse la moins coûteuse du système. Sa section consacrée à la preuve de conception inclut un test de charge des refus : le chemin qui répond « épuisé » doit rester rapide lorsqu'il représente 99 % du trafic, et la périphérie du réseau doit l'absorber. Un test ne répétant que l'achat réussi mesure le 1 % qui compte le moins pour la capacité.

Où les problèmes surviennent

Le test attend le système qu'il mesure. Dans le modèle fermé de k6, chaque utilisateur virtuel ne commence son itération suivante qu'après la fin de la précédente. Si le serveur ralentit, les utilisateurs ralentissent avec lui, le taux d'arrivée baisse et le test devient plus doux que prévu, effet appelé omission coordonnée dans certains travaux selon k6. Les vrais visiteurs ne s'attendent pas entre eux. Pour savoir si un système tient un rythme donné de requêtes, utilisez un exécuteur en modèle ouvert comme `constant-arrival-rate` ou `ramping-arrival-rate`, qui lance les itérations au rythme prévu quel que soit le temps de réponse.

Le générateur de charge est le goulet d'étranglement. Si la machine de test manque d'utilisateurs virtuels, de puissance de calcul ou de mémoire, les résultats décrivent le générateur plutôt que la cible. k6 déconseille généralement `maxVUs`, car ajouter des utilisateurs pendant le test consomme des ressources processeur et de la mémoire et peut surcharger le générateur, faussant les résultats. Dimensionnez `preAllocatedVUs` à l'avance avec quelques essais locaux et surveillez `dropped_iterations`. Quand une machine ne suffit pas, Locust répartit la charge sur plusieurs, avec un processus maître et de nombreux processus exécutants.

Le premier test est le plus gros. La documentation k6 rapporte que des milliers de ses utilisateurs ont lancé des tests de charge ayant rapidement fait tomber leurs applications ou environnements de préproduction. Commencez par un test de bon fonctionnement avec quelques utilisateurs virtuels pour vérifier script et environnement, poursuivez par un test de charge moyenne, puis seulement un test de stress ; k6 juge cet ordre essentiel. Les problèmes apparaissant à faible volume sont bien plus faciles à trouver avant la forte charge.

La charge monte et descend comme des montagnes russes. Un profil montant et descendant plusieurs fois gaspille des ressources et empêche d'identifier facilement quel changement a causé quel symptôme. k6 recommande une direction à la fois : monter, tenir un plateau, descendre. Conservez aussi le même scénario, les mêmes données et le même environnement entre les exécutions pour que deux résultats soient comparables.

Personne n'a défini les critères de réussite. Une exécution se terminant sur des graphiques sans critères est interprétée différemment par chaque personne, généralement avec indulgence. Écrivez les seuils avant le test, à partir de l'objectif du service, puis laissez le code de sortie trancher. Pour les longues exécutions, k6 peut arrêter dès qu'un seuil échoue avec `abortOnFail`, et `delayAbortEval` évite l'arrêt avant l'arrivée d'assez d'échantillons pour donner un sens au résultat.

Un seul type de test tient lieu de réponse complète. Un test de charge moyenne renseigne peu sur une vente dont le trafic afflue en quelques secondes, et un test de pic peu sur un défaut n'apparaissant qu'après des heures d'usage stable. k6 précise qu'aucun type ne supprime tous les risques : le profil de risque détermine la combinaison de tests. Un système exposé à l'usage prolongé exige d'abord des tests d'endurance ; un système exposé à l'usage intensif, des tests de stress. Un pic peut aussi solliciter des processus absents d'une journée ordinaire : le script doit donc les inclure.

Pour aller plus loin

- Types de tests de charge — la présentation k6 des tests de bon fonctionnement, charge moyenne, stress, endurance, pic et rupture, avec leurs charges et durées typiques.
- Tests de charge moyenne — le profil de montée, plateau et descente, et les enseignements de chaque phase.

- Seuils — les critères de réussite et d'échec de k6, leur syntaxe et le code de sortie en cas de seuil non respecté.
- Modèles ouverts et fermés — pourquoi un test dont les utilisateurs attendent les réponses sous-estime les ralentissements du système.
- Allocation des utilisateurs virtuels selon le taux d'arrivée — le nombre d'utilisateurs virtuels nécessaires à un test ouvert et le sens des itérations abandonnées.
- Qu'est-ce que Locust ? — la présentation par Locust de son outil de test de charge en Python et de son mode distribué.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un test de charge ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/load-test/