Guide pratique des tests de performance

Livre blanc de Neotys

Sommaire

Chapitre 1 – Introduction aux tests de performance

Les applications sont de plus en plus complexes avec des cycles de développement de plus en plus courts qui nécessitent de nouvelles méthodes efficaces de test et de développement. Les performances d'une application en termes d'expérience globale de l'utilisateur constituent désormais le facteur clé de la qualité d'une application.

Les projets séquentiels traditionnels avec des phases statiques de qualification/implémentation/test qui repoussent les tests de performance à la fin du projet présentent un risque de performance. Ce risque n'est plus acceptable au vu des normes de qualité d'aujourd'hui des applications.
Ce livre blanc fournit des informations pratiques sur la façon d'exécuter des tests de performance efficaces dans ce nouvel environnement plus exigeant.

État de complexité des applications modernes

L'un des principaux moteurs de l'évolution actuelle vers des tests de charge modernes est la complexité croissante du paysage informatique :

- La plupart des utilisateurs utilisent des appareils mobiles, des clients légers, des tablettes ou d'autres appareils pour accéder aux informations.
- Des applications complexes, partagées par plusieurs applications à la fois, sont créées régulièrement.
- Les nouvelles technologies offrent une gamme de solutions (cadre d'application AJAX, RIA, WebSocket, etc.) qui améliorent l'expérience utilisateur dans les applications.

Traditionnellement, vous testiez les applications pour valider leur qualité dans plusieurs domaines : fonctionnel, performance, sécurité, etc. Ces phases de test répondent aux exigences des utilisateurs et aux risques de l'activité.
Toutefois, le discours a changé ; la discussion ne porte plus sur la qualité, mais sur l'expérience utilisateur. L'expérience utilisateur est une combinaison d'apparence et de ressenti, de stabilité, de sécurité et de performance.

Performances : essentielles à la garantie d'une expérience utilisateur réussie

Les performances sont un facteur clé d'une expérience utilisateur réussie. Cela est dû aux progrès technologiques, à la complexité de l'architecture et aux emplacements et réseaux des utilisateurs. Les tests de charge ont été un ajout judicieux au processus de développement. Ils sont d'ailleurs maintenant devenus une étape essentielle des tests.

Les tests de charge et de performance répondent aux questions suivantes :

- L'application est-elle capable de gérer un certain nombre d'utilisateurs simultanés ?
- Le temps de réponse moyen pour le chargement des pages est-il acceptable sous cette charge définie ?
- L'application se comporte-t-elle à nouveau normalement après un pic de charge ?
- Combien d'utilisateurs l'application peut-elle gérer en maintenant un temps de réponse acceptable ?
- Quel est le seuil de charge au-dessus duquel les serveurs commencent à générer des erreurs et/ou à refuser des connexions ?
- Les serveurs restent-ils fonctionnels lorsqu'ils sont soumis à une forte charge ou se bloquent-ils ?

Comme toute activité de test, les tests de performance requièrent des méthodes et une logique appropriées.

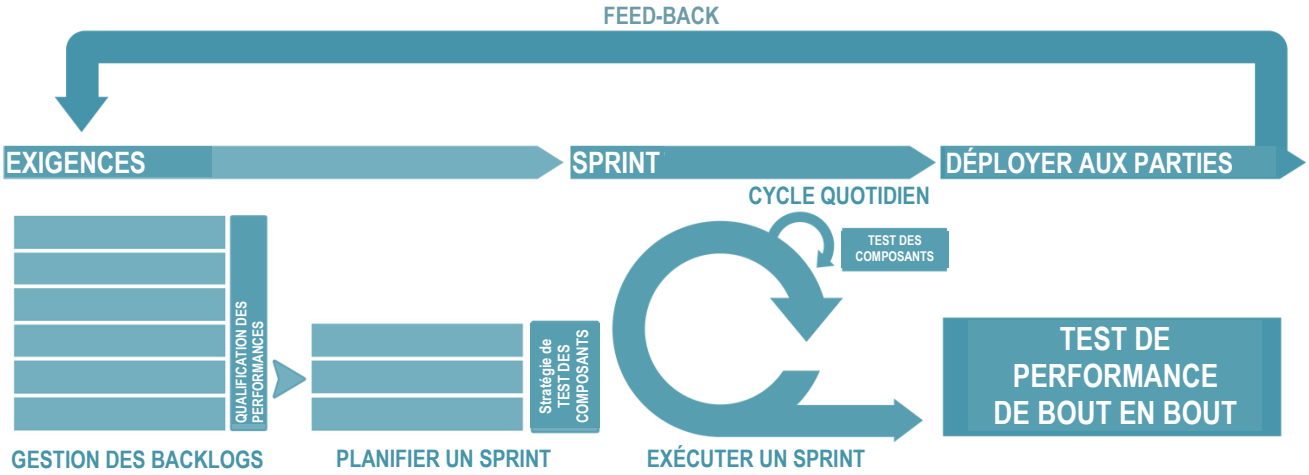
Lorsqu'une application réussit les tests de performance mais échoue ensuite en production, cela est souvent dû à des tests non réalistes. Dans ce cas, il est facile, bien que peu recommandé, de blâmer les tests eux-mêmes ou les outils utilisés pour les exécuter. Le vrai problème repose généralement sur une conception des tests qui ne prend pas en compte la base appropriée. Il faut se demander : « Que fallait-il savoir qui, si nous l'avions su, nous aurait permis de prédire cet échec avant la mise en production ? ». En d'autres termes : « Comment pouvons-nous fournir des tests de performance efficaces? »

Phases d'un projet de test de charge

Les méthodes de développement actuelles telles qu'Agile et DevOps permettent la création d'applications répondant rapidement aux besoins des clients. Ces méthodes impliquent la mise à jour de l'organisation du projet et nécessitent une collaboration étroite entre les équipes.

Dans ces méthodes, le cycle de vie du projet est organisé en plusieurs sprints, chaque sprint fournissant une partie de l'application.

Dans cet environnement, le processus de test de performance devrait suivre le workflow ci-dessous.



Une stratégie de test de performance devrait être mise en œuvre à un stade précoce du cycle de vie du projet. Première étape : qualification des performances. Elle définit l'effort de test de l'ensemble du projet.

Une approche traditionnelle des tests de performance obligerait le projet à attendre que l'application soit assemblée avant de commencer la validation des performances. Dans un cycle de vie de projet moderne, la seule façon d'inclure la validation des performances à un stade précoce consiste à tester les composants individuels après chaque build et à mettre en œuvre des tests de performance de bout en bout une fois l'application assemblée.

Chapitre 2 – Établir une stratégie de test de performance

Il s'agit de la première et de la plus importante étape des tests de performance. Elle définit :

- La portée des tests de performance
- La stratégie de charge
- Les contrats de niveau de service (SLA, Service Level Agreements) et/ou les objectifs de niveau de service (SLO, Service Level Objectives)

Il n'est jamais possible de tout tester, il faut donc prendre des décisions conscientes sur les points où concentrer l'intensité des tests. Généralement, 10 à 15 % des scénarios d'essai les plus fructueux révèlent entre 75 et 90 % des problèmes.

Tests basés sur le risque

L'évaluation des risques fournit un mécanisme permettant de hiérarchiser les efforts de test. Ce dernier permet de déterminer où diriger les efforts de test les plus poussés, et où il est préférable de tester seulement brièvement afin de préserver suffisamment de ressources pour les tests poussés.

Les tests basés sur les risques peuvent identifier les problèmes importants plus rapidement et plus tôt dans le processus, en portant uniquement sur les aspects les plus risqués d'un système.

La plupart des problèmes de performance et de robustesse d'un système surviennent dans les domaines suivants :

- Fonctions gourmandes en ressources
- Utilisations urgentes
- Goulots d'étranglement potentiels (basés sur l'architecture interne et la mise en œuvre)
- Impact sur les clients ou les utilisateurs, y compris en termes de visibilité
- Historique des défauts antérieurs (observations d'autres systèmes similaires en fonctionnement direct)
- Caractéristiques et fonctionnalités nouvelles et modifiées
- Forte demande : fonctionnalités utilisées de façon intensive
- Fonctionnalités complexes
- Exceptions
- Parties problématiques (mal élaborées ou entretenues) du système
- Maintenance de la plateforme

L'expert de l'industrie Ross Collard a élaboré la liste de questions ci-dessous pour identifier les différents risques de performance :

Vue situationnelle

- Quels domaines de l'exploitation du système, s'ils ont des performances insuffisantes, ont le plus d'impact sur les résultats (revenus et bénéfices) ?
- Quelles utilisations du système sont susceptibles de consommer un niveau élevé de ressources système par événement, quelle que soit la fréquence de l'événement ? Les ressources consommées doivent être significatives pour chaque événement et non pas élevées dans

l'ensemble simplement parce que l'événement se produit fréquemment et que le nombre total d'événements est élevé.

- Quels domaines du système peuvent être testés de façon minimale sans trop augmenter le risque, afin de conserver les ressources requises pour les domaines qui nécessitent des tests poussés ?

Vue des systèmes

- Quelles utilisations du système ont une priorité critique ?
- Quelles sont les utilisations les plus populaires (autrement dit, les plus fréquentes) ?
- Quelles utilisations sont les plus visibles (ont une visibilité élevée) ?
- Quelles sont les circonstances susceptibles de provoquer une forte demande du système de la part des utilisateurs externes (par exemple, des visiteurs distants d'un site web public qui ne sont pas des employés internes) ?
- Existe-t-il des fonctions particulièrement complexes dans le système, par exemple dans le domaine du traitement des exceptions ?
- Existe-t-il des domaines dans lesquels des technologies nouvelles et immatures, ou des méthodes inconnues et non éprouvées, ont été utilisées ?
- Existe-t-il d'autres applications secondaires qui partagent la même infrastructure et sont-elles censées interférer ou concurrencer de manière significative les ressources système (par exemple, les serveurs partagés) ?

Intuition/Expérience

- Que pouvons-nous apprendre du comportement des systèmes existants qui sont actuellement remplacés, comme leurs charges de travail et leurs caractéristiques de performance ? Comment pouvons-nous appliquer ces informations dans les tests du nouveau système ?
- Quelle a été votre expérience antérieure dans des situations similaires ? Quels aspects des fonctionnalités, styles de conception, sous-systèmes, composants et systèmes ont généralement présenté des problèmes de performance ? Si vous n'avez aucune expérience avec des systèmes similaires, veuillez ignorer cette question.
- Parmi les facteurs que vous avez identifiés en répondant aux questions précédentes, *quels* sont-ceux qui méritent d'être testés en priorité ? Quelles activités sont susceptibles de se produire simultanément et peuvent causer une charge et des contraintes importantes sur le système ?
- Selon votre compréhension de l'architecture du système et de l'infrastructure de support, où se trouvent les éventuels goulets d'étranglement ?

Vue des exigences

- Dans quelles circonstances une forte demande interne peut-elle avoir lieu (par exemple, en raison des employés internes d'un site web) ?
- Quelle est la stratégie d'archivage des bases de données ? Quel est le taux de données ajoutées par an ?
- Le système doit-il être disponible pendant 7 heures, 24 heures, etc. ?
- Des tâches de maintenance sont-elles exécutées pendant les heures d'ouverture ?

Les réponses à ces questions aident à identifier :

- Les domaines qu'il convient de tester
- Le type de test requis pour valider les performances de l'application

Test des composants

Une fois que les domaines fonctionnels requis pour les tests de performance ont été identifiés, décomposez les étapes métier en workflows techniques qui mettent en lumière les composants techniques.

Pourquoi les actions commerciales devraient-elles être fractionnées en composants ? Puisque l'objectif est de tester les performances à un stade précoce, le fait de répertorier tous les composants importants aide à définir une stratégie d'automatisation des tests de performance. Une fois qu'un composant a été codé, il est logique de le tester séparément et de mesurer :

- Le temps de réponse
- Le nombre maximal d'appels que le composant est capable de gérer

De plus, les tests des composants prennent en charge JMS, API, Service, Messages, etc., ce qui permet de créer et de gérer facilement des scénarios. Un autre avantage majeur de cette stratégie est que les interfaces des composants ont
moins de chance d'être affectées par les mises à jour techniques. Une fois
qu'un scénario de composant est créé, il peut être inclus dans le processus de génération et il est possible de recevoir un feed-back sur les performances de la build en cours.

Après chaque sprint, il est nécessaire de tester l'application assemblée en exécutant des tests utilisateur réalistes (impliquant plusieurs composants). Même si les composants ont été testés, il est obligatoire de mesurer :

- Le comportement du système lorsque plusieurs processus d'entreprise s'exécutent en parallèle
- Le temps de réponse réel de l'utilisateur
- La disponibilité de l'architecture
- La taille de l'architecture
- Stratégie de mise en cache

L'effort de test devient plus complexe à mesure que le projet avance. Au début, l'accent est mis sur la qualité des applications, puis toute l'attention est portée sur l'environnement cible, l'architecture et le réseau. Cela signifie que les objectifs des tests de performance varieront en fonction du calendrier du projet.

Environnement de test

Il est impératif que le système testé soit correctement configuré et que les résultats obtenus puissent être utilisés dans le système de production. Les considérations liées à l'environnement et à l'installation doivent rester prioritaires lors du développement de la stratégie de test. En voici quelques-unes :

- Quelles sont les données utilisées ? S'agit-il de données de production réelles, de données générées artificiellement ou de quelques enregistrements aléatoires ? Le volume des données correspond-il au
  volume prévu pour la production ? Si ce n'est pas le cas, quelle est
  la différence ?

- Comment les utilisateurs sont-ils définis ? Les comptes sont-ils définis avec les droits de sécurité appropriés pour chaque utilisateur virtuel ou un seul ID administrateur sera-t-il réutilisé ?

- Quelles sont les différences entre les environnements de production et de test ? Si le système de test n'est qu'un sous-ensemble du système de production, peut-on simuler la totalité de la charge ou seulement une partie de cette charge ?

Il est important que l'environnement de test reflète le mieux possible l'environnement de production, mais certaines différences peuvent subsister.
Même si les tests portaient sur l'environnement de production avec les données de production réelles, ils ne représenteraient qu'un seul point dans le temps. D'autres conditions et facteurs devraient également être pris en compte.

Concevoir un plan de test

Le plan de test est un document décrivant la stratégie de performance. Il doit inclure :

- Des évaluations du risque de performance mettant en évidence les exigences de performance

- Une modélisation des performances expliquant la logique de calcul d'une autre stratégie de charge

- La transposition du parcours utilisateur principal en composants

- La description des différents parcours utilisateur avec le temps de réflexion spécifique par transaction métier

- Le ou les jeux de données

- Le contrat de niveau de service (SLA)

- La description de chaque test à exécuter pour valider les performances

- Les environnements de test

Le plan de test constitue le fondement d'une stratégie de test de performance bien conçue et exécutée. Il est la preuve qu'une équipe a pris en compte comme il se doit le rôle essentiel des performances dans l'expérience finale des utilisateurs.

Dans de nombreux cas, les équipes de projet garantissent la livraison des plans de test de performance au cours des cycles de développement et de planification, en les exigeant pour considérer le projet comme prêt à démarrer. Bien que chaque histoire ou chaque cas d'utilisation ne nécessite pas le même niveau de tests de performance, il est primordial de placer le processus de réflexion au cœur du projet pour obtenir de meilleurs systèmes et une meilleure appréciation globale de la qualité de bout en bout livrée par l'équipe.

Chapitre 3 - Modélisation des tests de performance

L'objectif des tests de charge est de simuler une activité utilisateur réaliste sur l'application. Si un parcours utilisateur non représentatif est sélectionné
ou si la bonne stratégie de charge n'est pas définie, le comportement de l'application sous charge ne pourra pas être correctement validé.

La modélisation des tests de performance ne nécessite aucune compétence technique, mais seulement du temps pour comprendre pleinement l'application :

- Comment les utilisateurs utilisent-ils le système ?
- Quelles sont leurs habitudes ?
- Quand et à quelle fréquence utilisent-ils l'application ? Et d'où ?
- Existe-t-il une relation entre les événements externes et les pics d'activité ?
- Le plan d'affaires de l'entreprise est-il lié à l'activité de cette application ?
- La communauté des utilisateurs va-t-elle s'étendre dans d'autres zones géographiques ?
- Existe-t-il un plan marketing pour commercialiser/promouvoir l'application ? Si oui, quel en est le public ?
- Existe-t-il des couches d'architecture partagées avec d'autres systèmes ?

Pour comprendre pleinement l'application lors de la modélisation des performances, les rôles suivants doivent être impliqués :

- Architecte fonctionnel
- Analystes d'entreprise
- Chef de projet

Bien sûr, les différents rôles fourniront un feed-back différent, mais l'idée est de comprendre l'application, les habitudes des utilisateurs et la relation de l'application avec l'organisation actuelle ou future.

Les journaux système ou les extractions de bases de données sont également très utiles. Ceux-ci indiquent souvent les principaux composants et domaines fonctionnels utilisés en production. Ils récupèrent également le nombre de transactions/heure par processus d'entreprise.

- Lorsque l'on considère la charge de l'utilisateur, il faut être conscient de la différence entre le nombre d'utilisateurs **simultanés** et le nombre d'utilisateurs au sein du public visé. Une application fournie à 5 000 utilisateurs peut n'être utilisée simultanément que par 500 utilisateurs.
- Sans estimation du nombre maximal d'utilisateurs simultanés, le nombre d'utilisateurs peut être calculé en fonction du nombre de transactions/heure relatives aux actions d'entreprise.

Établir les contrats de niveau de service (SLA, Service Level Agreements) et/ou les objectifs de niveau de service (SLO, Service Level Objectives)

Les contrats SLA et les objectifs SLO sont la clé de l'automatisation et de la validation des performances. Le chef de projet et l'architecte fonctionnel doivent définir des temps de réponse optimaux pour l'application. Il n'y a pas de temps de réponse standard.

Un contrat SLA ou un objectif SLO permettra à l'ingénieur Performance de fournir un statut en fonction des résultats des tests de performance. Avec le contrat SLA ou l'objectif SLO, les tests de performance peuvent être facilement automatisés pour identifier une régression des performances entre plusieurs versions de l'application.

Si l'importance des tests des composants et de l'insertion des tests de performance à un stade précoce du projet est bien comprise, le contrat SLA ou l'objectif SLO doit être défini au niveau des composants pour permettre une automatisation correcte des tests.

Sélection des tests

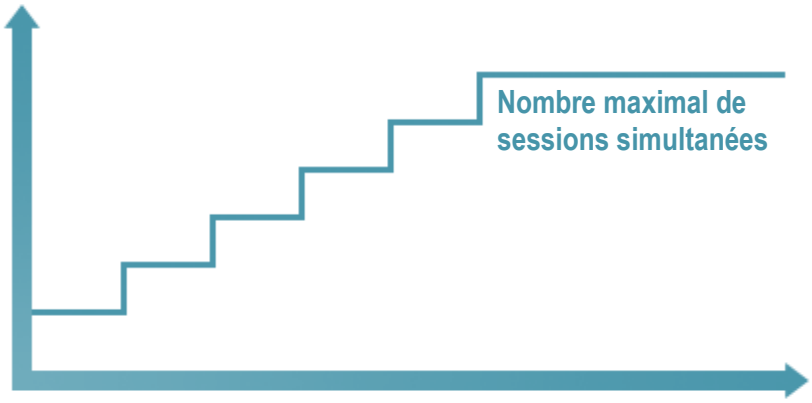
Chaque test exécuté traite un risque individuel. Chaque test fournit un statut relatif aux exigences de performance.

Bien sûr, le but de la plupart des projets est d'atteindre la limite de l'application. Atteindre la limite de l'application est important pour valider le dimensionnement et la configuration de l'architecture, mais cela ne répondra probablement pas à toutes les exigences de performance.

D'autres tests doivent être exécutés pour qualifier correctement les performances de l'application.

Les tests suivants aideront à valider les exigences de performance :

- Test unitaire : De nombreux projets négligent la puissance des tests d'une unité individuelle. Ne commencez pas à exécuter des tests de charge si les performances ne sont pas acceptables avec un seul utilisateur. Un test unitaire permet à l'application d'être instrumentée avec des outils d'analyse approfondie et fournit des informations utiles aux développeurs. Cette étape devrait être une composante obligatoire de toute stratégie de test de charge.

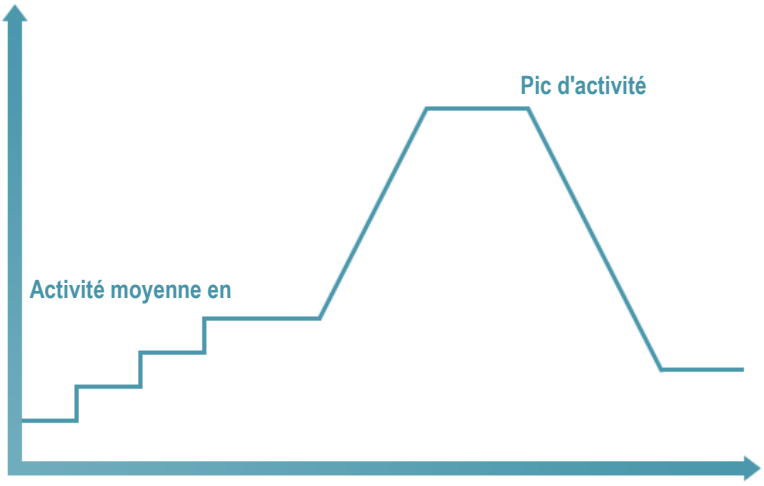


- Test de connexion : Supposons que chaque matin, tous les utilisateurs de l'application se connectent au système à peu près à la même heure, puis vont prendre un café ou discutent avec leurs collègues. Même s'il n'y a pas d'activité majeure sur les composants d'entreprise de l'application, l'architecture doit gérer ce pic très important de sessions utilisateur. Il est donc évident qu'il convient de garantir que le système ne se bloque pas tous les matins. La validation de ce point facilitera grandement le travail de l'équipe ops (opérations). Ce test incorporera tous les utilisateurs attendus.

- Test d'activité en production : Chaque application inclut des actions d'entreprise majeures. Ces actions entraînent souvent la mise à jour ou l'insertion de données dans la base de données. Par conséquent, il convient d'avoir une idée juste du nombre d'enregistrements créés par jour ou par heure. Le test d'activité en production garantit que le système est capable de gérer la charge liée au nombre de transactions/heure attendues en production. Ce test charge les composants d'entreprise et valide leur comportement au sein de la base

de données. Les deux exigences principales pour ce type de test sont le nombre de transactions/heure par parcours utilisateur et les temps de réflexion appropriés.

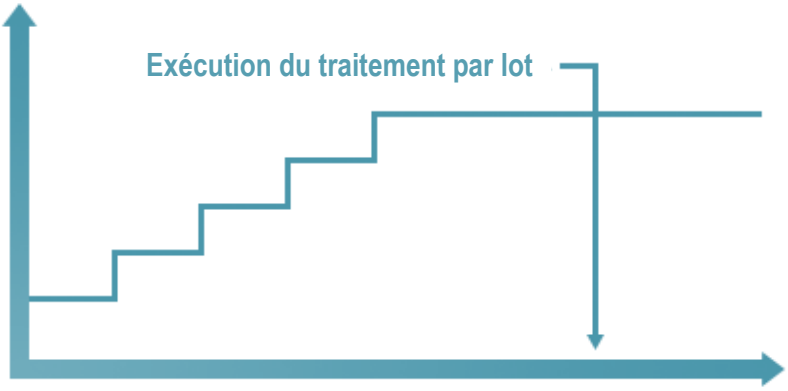
- **Test de pic d'activité :** Chaque application peut être confrontée à des pics de volumes. Même si ces pics apparaissent seulement une ou deux fois par an, il est obligatoire de s'assurer que l'application peut les gérer. Le but d'un test de pic d'activité est de simuler une augmentation importante du nombre d'utilisateurs sur une courte période de temps (par exemple, un site de



vente au détail peut avoir besoin de simuler une activité qui résulterait d'une grande vente portant sur des produits extrêmement populaires).

- **Test d'endurance :** L'application/architecture sera probablement disponible pendant plusieurs jours ou même plusieurs semaines. Si aucune période donnée n'est dédiée aux tâches de maintenance, il est important que l'architecture s'exécute sans défaillance pendant une longue période. Un test d'endurance fera intervenir un nombre constant d'utilisateurs sur une période de temps prolongée. Ce test permet d'identifier aisément toute fuite de mémoire ou conjonction de réseau, et de superviser la stabilité de la configuration actuelle de l'environnement.

- **Test par lot :** Certaines applications sont conçues pours des tâches ou des lots asynchrones.

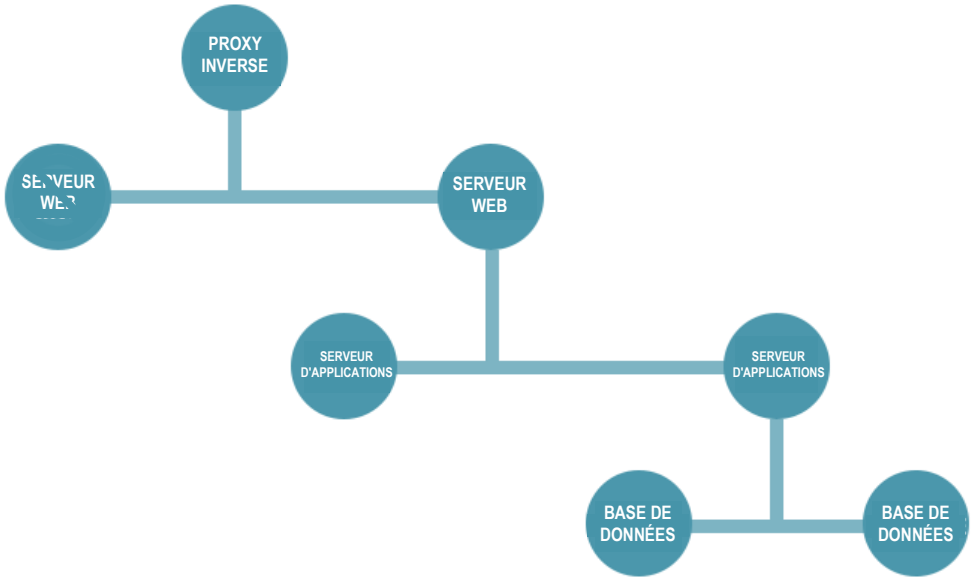


Quel est l'impact d'un lot sur les utilisateurs réels ? Le test par lot exécute l'activité de production et déclenche l'exécution d'un lot de tâches à un moment spécifique du test. Le but de ce test est d'identifier si le lot de tâches a des répercussions sur l'expérience utilisateur.

Un certain nombre d'autres tests de charge pourraient être utilisés en fonction des contraintes de l'entreprise et de l'emplacement des utilisateurs. Il y a toujours des événements liés à la société ou à l'organisation qui affecteront la charge de l'application.

Par exemple, la charge conçue pour une application de courtage dépendra des heures d'ouverture des différents marchés. L'application aura trois phases typiques : les utilisateurs asiatiques, puis les utilisateurs européens associés à une partie des utilisateurs asiatiques, puis les utilisateurs américains associés aux utilisateurs européens. Chaque fois que les marchés ouvrent ou ferment, il y a des pics d'activité de courtage. Par conséquent, les tests incluraient différents types de charges associant différentes combinaisons de cas d'affaires.

D'un autre côté, il existe des tests conçus pour valider la disponibilité de la plateforme au cours des



tâches de maintenance ou d'un incident en production :

- **Test de basculement :** Le but de ce test est de simuler une défaillance en production sur l'environnement. Ce type de test est obligatoire pour valider la disponibilité de l'environnement et s'assurer que le mécanisme de cluster de basculement réagit correctement. Lorsque l'architecture a N nœuds, il est important de valider que les nœuds N-1 ou N-2 peuvent gérer la charge attendue, afin que les événements ne s'accumulent pas au cours d'un problème en production. L'équipe ops souhaite également savoir si elle peut ou non effectuer ses tâches de maintenance sans avoir à placer l'application en état de maintenance. La plupart des applications à haute disponibilité ont de nombreux nœuds sur différentes couches de l'architecture : serveurs web, serveurs d'applications, base de données, etc. Il devrait y avoir un test par couche.

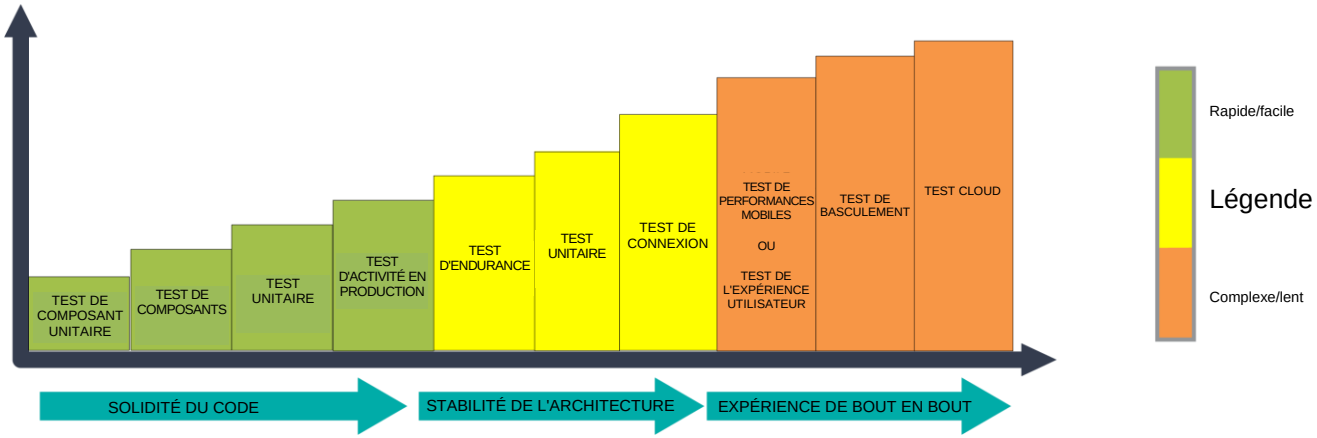
- **Test de récupération :** Ce test est en fait le contraire du test de dégradation. Le test est initialisé en arrêtant l'un des nœuds, puis en redémarrant le nœud pendant le chargement de l'application. Le but de ce test est de voir comment le nœud réagit à une lourde charge juste après avoir redémarré.

De plus, des situations opérationnelles doivent éventuellement être incluses pendant le test de charge pour mesurer le comportement de l'application lors du nettoyage du cache.

Il y a encore un autre point qui pourrait affecter les performances de l'application : les données. Souvent, lors des tests de charge, une base de données de test dotée de moins de données est utilisée, ou du moins une base de données rendue anonyme à des fins de sécurité. Le processus anonyme crée généralement des enregistrements commençant par les mêmes lettres ou chiffres. Ainsi, tous les index utilisés lors du test de charge seront incomparables à ceux de la base de données de production normale.

Les données augmentent assez rapidement en production. Les bases de données réagissent assez différemment en fonction de leur taille. Si la durée de vie de la base de données est longue, il peut être judicieux de valider les performances avec des bases de données de tailles différentes pour voir si la limite est différente avec des bases de données légère et lourde. Pour réaliser ce type de test, un

grand jeu de données se référant à différentes zones de la base de données doit être utilisé ; n'utilisez jamais une liste de noms où tous les comptes commencent par AA … … AAA2 … mais plutôt un jeu de données représentatif allant de A à Z.



Le type et la complexité du test changeront au cours du cycle de vie du projet :

Inclure les temps de réflexion

- Un élément important de la conception des performances est le « temps de réflexion ».
- Le temps de réflexion est le temps requis par un utilisateur réel entre deux actions d'entreprise :
- Temps requis pour lire les informations affichées à l'écran
- Temps requis pour remplir un formulaire
- Autres actions d'utilisateur réel qui ne provoquent pas d'interaction avec les serveurs d'applications

Comme chaque utilisateur réel se comporte différemment, les temps de réflexion sont toujours différents. Par conséquent, il est important de :

- Compiler le temps de réflexion de chaque étape d'entreprise dans chaque scénario. Évitez d'utiliser le même temps de réflexion pour chaque étape d'entreprise. Il y a toujours des écrans affichant des informations que les utilisateurs doivent lire ou des formulaires qu'ils doivent remplir avant de passer à l'étape suivante du processus d'entreprise. Il devrait y avoir un temps de réflexion spécifique pour chaque action.
- Calculer le temps de réflexion minimum moyen et le temps de réflexion maximum moyen par action.
- Laisser l'outil de test de charge choisir au hasard un temps de réflexion dans une plage spécifiée (min, max).

Imaginez que l'application est un château avec de grands murs et de grandes portes. Les composants qui seront validés (via un test de charge) sont à l'intérieur de ce château. Le mur représente les serveurs proxy, les équilibreurs de charge, les couches de mise en cache, les pare-feu, etc.

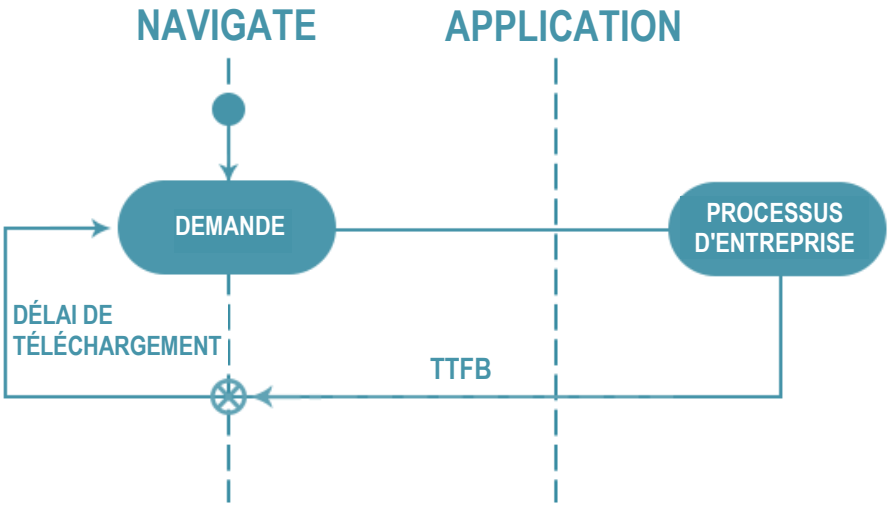
Si un test est exécuté sans inclure de temps de réflexion, seule la porte principale est touchée. Elle peut céder, mais il est fort probable qu'elle verrouille tout après quelques minutes. Soyez subtil et

cachez-vous de la défense. La porte principale autorise l'entrée et les composants situés à l'intérieur du château peuvent être testés en charge correctement.

Valider l'expérience utilisateur

Comme mentionné précédemment, une fois l'application assemblée, les objectifs de test vont changer. À un moment donné, la qualité de l'expérience utilisateur doit être validée.

Les applications mobiles, les applications Internet enrichies (RIA) et les cadres d'application AJAX complexes constituent un défi, dans la mesure où la plupart sont utilisés pour mesurer les temps de réponse. Par le passé, les mesures étaient limitées au temps de téléchargement et au temps jusqu'au premier octet (TTFB).



Cette approche n'est pas judicieuse car une grande partie du temps de rendu qui contribue à l'expérience utilisateur dépend de la logique ActiveX/JavaScript locale ou de l'application native. Ainsi, les mesures de test ne peuvent pas être limitées à TTFB et au temps de téléchargement, car l'objectif principal ici consiste à valider l'expérience utilisateur globale sur l'application.

Il est possible de mesurer l'expérience utilisateur en combinant deux solutions : un logiciel de test de charge (NeoLoad) et un outil de test basé sur navigateur ou mobile.

L'outil de test de charge générera 98 % de la charge sur l'application. L'outil de test basé sur navigateur ou mobile générera les 2 % restants de la charge pour récupérer l'expérience utilisateur réelle (y compris le temps de rendu) pendant le chargement de l'application.

Cela signifie qu'il faut soigneusement identifier les transactions et les processus d'entreprise que les solutions basées sur navigateur/mobile doivent superviser.

Supervision

Exécuter des tests sans supervision, c'est un peu comme regarder un film d'horreur à la radio. Vous entendez des gens crier sans savoir pourquoi. La supervision est le seul moyen d'obtenir des métriques liées au comportement de l'architecture.

Toutefois, de nombreux projets ont tendance à ne pas prendre en charge une supervision suffisante des performances en raison :

- d'un manque d'outils,
- de la peur des exigences requises pour effectuer la supervision.

La supervision ne se limite pas au système d'exploitation des différents serveurs, mais son but est de valider que chaque couche de l'architecture est disponible et stable. Les architectes ont pris le temps d'élaborer une architecture des plus intelligentes. Il est donc nécessaire de mesurer le comportement des différentes couches.

La supervision permet de mieux comprendre l'architecture et d'étudier le comportement des différentes parties de l'environnement :

- Système d'exploitation : UC, mémoire, disque et utilisation du réseau ...
- Serveur d'applications : utilisation de la mémoire, récupérateur de mémoire, utilisation des threads, sessions ...
- Serveur web : collaborateur, nombre de sessions ...
- Base de données : pool de mémoires tampons, utilisation du cache, nombre de transactions, nombre de validations, pourcentage de requêtes indexées ...
- Serveur de mise en cache : taux de réussite

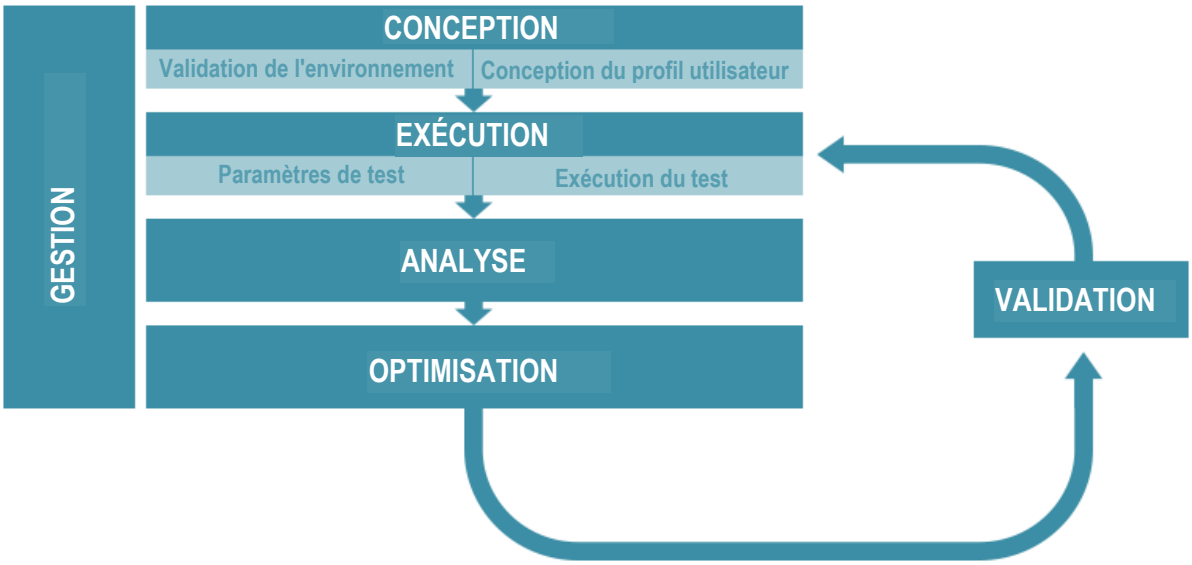
De nombreux projets utilisent des outils de supervision en production pour extraire des métriques de l'architecture. Cette approche n'est pas recommandée car la supervision de la production présente une haute granularité entre les différents points de données (toutes les 2 à 5 minutes). Lors des tests de charge, il est important de faire collecter les données de supervision au moins toutes les cinq secondes. Il convient de fournir à l'ingénieur Performance toutes les opportunités d'identifier les goulots d'étranglement. Lorsqu'un pic n'apparaît que quelques secondes au cours d'un test, il est essentiel d'avoir une granularité suffisante pour signaler le goulot d'étranglement.

La supervision nécessite des exigences techniques telles qu'un compte système, la liste des services à démarrer, les ports de pare-feu à ouvrir, etc. Même s'il semble difficile de répondre à ces exigences, la supervision est possible si une communication adéquate accompagne les opérations ; l'envoi préalable de ces exigences facilite cette communication.

Point clé à retenir pour la supervision : anticipez les exigences à un stade précoce.

Chapitre 4 – Exécuter un test de performance

Chaque activité de test de performance est mappée dans le flux de travail suivant :



Les objectifs des tests de performance doivent être adaptés en fonction des résultats. L'ingénieur Performance analyse ces résultats entre le test et l'optimisation. En fin de compte, les principaux objectifs d'un test de performance sont d'identifier les goulets d'étranglement et de proposer des solutions.

Voici les trois étapes clés de l'exécution d'un test de performance efficace.

Conception

La phase de conception est souvent appelée « phase d'écriture de script », mais la nouvelle technologie a rendu la phase de conception plus complexe.

La création de scripts de test de performance est, par nature, un projet de développement logiciel. Parfois, la génération automatique d'un script à partir d'un enregistrement est interprétée à tort comme le processus complet de création de script, alors qu'il ne s'agit que du début. Ce n'est que dans des cas très simples que la génération automatique fournit des scripts complets ; dans la plupart des cas non triviaux, ce n'est qu'une première étape. Les scripts doivent être corrélés (obtenir des variables dynamiques du serveur) et paramétrés (utiliser des données différentes pour des utilisateurs différents). Ces opérations sont sujettes à erreurs car les modifications sont apportées directement au flux de communication. Toute erreur à ce stade peut être très dangereuse car ces types d'erreurs ne peuvent généralement pas se produire dans le monde réel où les utilisateurs interagissent avec le système via une interface utilisateur ou des appels d'API.

Les applications AJAX, le protocole WebSocket et les technologies d'interrogation génèrent des requêtes régulières. Ces requêtes ne sont pas générées par l'interaction de l'utilisateur dans l'interface utilisateur graphique. Au lieu de cela, de nombreux appels sont générés par un ou plusieurs composants de page individuels. Par exemple, les pages d'un site web de commerce électronique présentent toujours un lien direct vers le panier de l'utilisateur. Ce lien doit montrer à l'utilisateur le nombre exact de produits figurant actuellement dans son panier. La plupart du temps, le nombre de produits est affiché via une requête d'interrogation (appel technique effectué toutes les cinq secondes).

Ce type d'appel doit être compris par l'ingénieur Performance. Tous les mécanismes « d'interrogation » doivent être supprimés de l'étape d'entreprise du parcours utilisateur et placés dans un thread dédié.

La technologie de streaming adaptatif constitue un autre exemple pertinent. De nombreux testeurs s'appuient souvent sur l'approche d'enregistrement/lecture. Malheureusement, ce type de logique ne bénéficie à aucune stratégie de test.

L'objectif est de qualifier le comportement de l'application sous une charge réaliste, et non de valider la mise en cache.

La relecture de la demande d'enregistrement sans corrélation n'appellera que le cache de l'application :

- Cache web
- Cache de l'application

Le jeu de données est un autre élément important de l'étape de conception. L'utilisation d'un jeu de données réduit génère exactement la même requête sur la base de données. Comme mentionné précédemment, le but d'un test de charge n'est pas de qualifier l'efficacité du cache de base de données ni de simplement générer des blocages.

Une fois le script créé, il doit être évalué pour un utilisateur unique, plusieurs utilisateurs et avec des données différentes. Ne supposez pas que le système fonctionne correctement lorsqu'un script s'exécute sans erreurs. Au lieu de cela, assurez-vous que la charge de travail appliquée fait ce qu'elle est censée faire et que toutes les erreurs sont interceptées et consignées. Cela peut être fait directement en analysant les réponses du serveur ou, dans les cas où cela est impossible, indirectement. Cela peut être fait, par exemple, en analysant le journal d'application ou la base de données pour détecter certaines entrées particulières.

De nombreux outils permettent de vérifier la charge de travail et de vérifier les erreurs, mais il est nécessaire de comprendre pleinement ce qu'il se passe exactement. NeoLoad permet à l'utilisateur de créer des affirmations pour s'assurer que la réponse du serveur est « normale » (autrement dit, que le contenu est comme prévu). Par exemple, la réception d'erreurs « Mémoire insuffisante » dans une page de navigateur à la place des rapports demandés constitue un contenu inattendu. Il est important de repérer de telles erreurs de contenu à l'aide d'affirmations.

NeoLoad et de nombreuses solutions de test de charge capturent automatiquement les erreurs HTTP pour les scripts web (par exemple, 500 « Internal Server Error »). Si l'on se fie uniquement aux diagnostics par défaut, il n'est pas possible de vérifier que les transactions d'entreprise sont conformes aux attentes.

La nouvelle conception web a tendance à éviter les erreurs HTTP et affiche des erreurs d'exception dans les applications. Ainsi, si la vérification est limitée au seul examen des codes HTTP, il est impossible de déterminer si le scénario performe les actions attendues sur l'application.

Exécution

Lors de l'exécution du test réel, prenez en compte les défis suivants :

- Architecture de test de charge
- Supervision efficace des tests

Architecture de test de charge

L'objectif principal est d'effectuer un test de charge de l'application, et non pas du logiciel de test de charge.

La plupart des solutions de test de charge ont plusieurs composants :

- Contrôleur : orchestre le test et stocke les données des résultats : temps de réponse, nombre d'accès/s, débit, supervision des métriques, erreurs ...
- Générateur de charge : exécute le script de test de charge sur l'application. Ce composant doit gérer plusieurs centaines d'utilisateurs virtuels simultanés.

Pour éviter d'être limité par l'architecture de test de charge, il doit y avoir suffisamment :

- de générateurs de charge pour atteindre la charge attendue ;
- de bande passante réseau entre le générateur de charge et l'application.

Le dimensionnement des générateurs de charge est une étape cruciale dans la définition du nombre d'utilisateurs virtuels qu'un générateur de charge peut gérer. Ce nombre d'utilisateurs virtuels dépend de nombreux aspects :

- Technologies utilisées dans l'application testée
- Complexité de l'application testée
- Capacité de connexion réseau

Commencer un test de montée en puissance (ou « d'évolutivité ») de l'application avec un seul générateur de charge aide à déterminer la capacité maximale d'une machine. Après avoir déterminé le nombre d'utilisateurs virtuels qu'un générateur de charge peut gérer et le nombre d'utilisateurs virtuels pour le test, il est facile de calculer le nombre de générateurs nécessaires à l'exécution du test. Pour identifier les limites de charge d'un générateur de charge, il est nécessaire de vérifier comment il réagit face à une charge de test croissante, et quels effets cela peut avoir sur :

- l'UC,
- la mémoire,
- le débit,
- le taux de réussite,
- la charge d'utilisateurs virtuels.

Le premier point de rupture significatif dans les métriques d'UC, de mémoire, de débit ou de taux de réussite représente la limite de performance du générateur de charge. Ce point doit être corrélé avec le nombre d'utilisateurs virtuels générés. Des problèmes sérieux peuvent se produire avec les générateurs de charge s'ils sont poussés au-delà de ce nombre d'utilisateurs virtuels.

Une marge de sécurité de 20 à 30 % des limites de taille est recommandée pour les générateurs de charge dans le cadre d'un test de charge poussé.

Il est également important d'éviter d'utiliser l'ordinateur contrôleur comme générateur de charge. Le contrôleur est le cœur du test. Il est toujours préférable de perdre un générateur de charge qu'un contrôleur (le résultat du test complet).

Supervision efficace des tests

Comme mentionné précédemment, il existe un délai limité pour exécuter différents tests et procéder à la supervision. Il ne faut pas perdre de temps à examiner les données de test pendant le test lui-même.

Si le temps de réponse commence à augmenter en raison de la saturation du serveur web, arrêtez le test et réglez le serveur web. Presque chaque fois que des tests sont lancés sur une nouvelle application et/ou un nouvel environnement, la plupart des couches (serveur d'applications, serveur web, etc.) ne sont pas configurées ni réglées pour la charge de l'application. Ainsi, chaque test effectué sur un environnement représentatif nécessite l'attention de l'ingénieur Performance qui réglera correctement l'environnement.

Chaque nouveau test utilisant un nouveau jeu de données doit également faire l'objet d'une attention particulière. Par exemple, arrêtez le test si chaque utilisateur de test génère des erreurs lorsqu'il se connecte à l'application.

D'un autre côté, si vous comprenez bien l'application et l'environnement, vous pouvez activer l'automatisation des tests sans prêter attention au comportement de l'application. Une fois le test terminé, analysez les résultats.

Analyse

L'analyse des résultats des tests de charge est un travail en soi. Une connaissance approfondie des éléments suivants est requise :

- Conception des tests de charge
- Couches techniques impliquées dans l'application
- Architecture moderne

Ce livre blanc n'explique pas comment analyser les résultats, mais fournit des recommandations pour rédiger des rapports sur les résultats des tests de performance du projet.

Presque toutes les solutions de test de charge permettent la création de graphiques complexes qui mettent les données en corrélation. Le premier réflexe de l'ingénieur Performance sera de montrer tous les graphiques dans le rapport.

Toutefois, avant de créer ce rapport, il est important de comprendre le rôle et les compétences techniques de la personne qui valide ou qui lit simplement le rapport.

En fonction de ces qualités, différents types de rapports peuvent être créés. Par exemple :

- Un rapport technique pour les développeurs et les opérations, affichant uniquement les graphes importants
- Un rapport pour les décideurs, fournissant un statut très simple des performances de l'application

L'objectif principal du rapport de test de performance est de fournir un statut clair sur les performances de l'application. Les rapports de résultats doivent être simplifiés et mettre l'accent sur les trois principaux thèmes d'application suivants :

- Temps de réponse
- Disponibilité
- Évolutivité

Une présentation graphique (avec trois camemberts) rend les résultats plus faciles à comprendre pour les décideurs.

En fin de compte, le rapport de performance doit mettre en évidence la validation ou non des exigences de performance (identifiées pendant la phase de stratégie de performance).

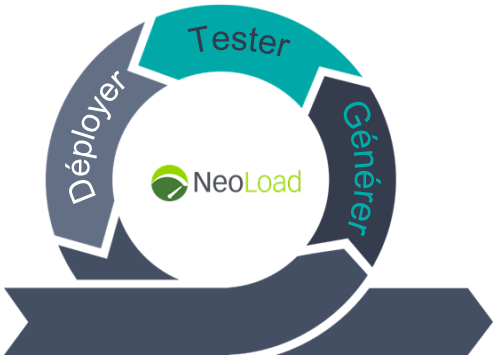
Références

DÉTERMINER LA PRIORITÉ DU TEST PAR L'ÉVALUATION DES RISQUES, Ross Collard

http://www.performance-workshop.org/documents/Determining_Test_Focus_Thru_Risk_assessment_Collard.pdf

À propos de Neotys

Le succès de votre stratégie numérique repose sur votre faculté à livrer régulièrement des logiciels rapides et fiables. La création rapide de bons logiciels à l'aide d'un processus optimisé de test de performance constitue votre avantage concurrentiel, et Agile et DevOps font partie de cette solution.



Neotys a investi plus de 12 ans de développement dans NeoLoad, la plateforme de test de performance conçue pour accélérer les processus Agile et DevOps. Elle a été élaborée par des ingénieurs qui ont reconnu que, pour atteindre leur propre objectif d'adoption d'Agile, ils devaient créer un produit favorisant des tests continus de charge et de performance plus efficaces.

Le résultat final est une création et une maintenance de tests jusqu'à 10 fois plus rapides avec NeoLoad.

Nous croyons vraiment que l'ingénieur Performance peut devenir le partenaire clé de la performance des applications qui assurera la meilleure couverture de test tout en respectant la cadence du processus de livraison continue. Les performances devenant la responsabilité de l'ensemble de l'équipe, nous travaillons quotidiennement à fournir une plateforme de tests de performance optimisée.

Pour plus d'informations sur Neotys et NeoLoad, visitez : www.neotys.com ou contactez sales@neotys.com