
Visual Studio
Testez simplement
la montée en charge
de vos applications
INFINITE SQUARE
Microsoft Technology Centers

Testez simplement la montée en charge de vos applications

Sommaire

Avertissement

Ce document s'adresse aux testeurs, aux architectes, aux concepteurs et développeurs, ainsi qu'aux chefs de projet qui veulent pouvoir contrôler les composants logiciels qu'ils produisent d'une manière plus systématique et mieux outillée que celle généralement pratiquée dans l'industrie aujourd'hui.

Auteurs

Etienne Margraff (InfiniteSquare)

Simon Ferquel (InfiniteSquare)

Stéphane Goudeau (Microsoft France)

Mai 2012

Introduction

Le cycle de vie du développement d'une application Cloud requiert la mise en œuvre de multiples catégories de tests. Les principes d'application de ces tests restent similaires à ceux ciblant une plateforme à demeure, toutefois, en fonction de leur type (fonctionnel, intégration, performance, évolutivité, déploiement, mise à jour, sécurité, diagnostic, supervision,….), des différences sont à prendre en considération, notamment en ce qui concerne l'hébergement de l'environnement d'exécution.

Par exemple, si l'on considère une application Windows Azure, il est fréquent d'automatiser les tests fonctionnels sur un environnement émulant le fonctionnement des services de cette plateforme ou d'exécuter directement en ligne les tests d'intégration sur une souscription Azure dédiée. Les « tests de charge » supposent également la mise en place d'une infrastructure dédiée à la simulation d'une forte sollicitation du système cible et à la collecte de résultats. L'objectif est de pouvoir simuler le comportement d'un nombre significatif d'utilisateurs et d'anticiper les problèmes qui pourraient survenir. Cette démarche permet alors de mettre en place une stratégie d'optimisation associée à des procédures de reprise testées et éprouvées en cas de problème en production.

Les tests de charge

1. Périmètre des tests de charge

Deux cas d'utilisation des tests de charge sont fréquemment rencontrés :

- Valider une infrastructure cible avant sa mise en production : s'assurer que l'infrastructure et l'application vont supporter la charge utilisateur visée
- Contrôler la qualité de l'application sur la durée : s'assurer quotidiennement qu'il n'y ait pas de régression en termes de performances de l'application afin de pouvoir réagir immédiatement en cas de soucis

Les tests de charge permettent donc de s'assurer que la solution fonctionne correctement avec un grand nombre d'utilisateurs. Le périmètre couvert par ces tests regroupe :

- Les tests caractéristiques de performances : comment la réactivité de l'application est affectée en augmentant la charge
- Les tests de stress : comment la solution gère les niveaux extrêmes de charge
- Les tests d'évolutivité : comment la solution réagit en fonction du modèle de « scaling » (« scale-up » ou « scale out »)
- Les tests de durée : comment se comporte la solution si une charge est appliquée sur une période prolongée de test

Test de la continuité d'activité : comment la solution gère une mise en échec partielle de son infrastructure logicielle ou matérielle.

2. Démarche

La mise en place d'un test de charge nécessite une méthodologie en quatre étapes :

- Définition du périmètre de test : que va-t-on tester ? Comment et avec quels objectifs ?
- Définition de l'architecture matérielle : que va-t-on utiliser comme environnement de test et tester ?
- Création et exécution de la campagne
- Analyse des résultats : quels sont les problèmes et les axes d'amélioration ? Où est le point de rupture (à partir de quand est-ce que l'on considère que l'application/l'infrastructure a atteint ses limites) ? Et quels sont les facteurs limitant qui permettraient de décaler le point de rupture (processeur trop utilisé, mémoire saturée, goulets d'étranglements dans l'application…).

Il faut mettre en place une méthode de test de charge qui sera appliquée tout au long du cycle de développement de l'application. Chaque phase de ce développement peut être associée à une ou plusieurs campagnes de tests de charge associée. Bien évidemment, la nature des tests de charge (qui présuppose un minimum d'intégration), rend complexe leur mise en œuvre en début de cycle de développement. Les tests de charge font partie de la catégorie des tests d'intégration, ce qui signifie qu'il est compliqué d'en effectuer extrêmement tôt au sein de ce cycle, contrairement aux tests unitaires par exemple.

Il faut cependant éviter de tomber dans l'extrême inverse et réaliser une seule campagne de charge en fin de développement. Si l'unique test de charge de votre système distribué met en avant de graves problèmes de performances à 2 semaines de la fin d'un développement de 10 mois, il est certain que le projet finira en retard, voire pire : sans la correction de ces problèmes !

Il s'agit donc de trouver un juste équilibre. Il n'y a pas de règle qui fonctionne dans tous les cas, mais un minimum de deux tests de charges est souvent nécessaire. Le plus généralement, le premier aura lieu en milieu de développement, et le second en fin pour valider que les problèmes potentiels identifiés dans le précédent ont été correctement adressés. Ce minimum de deux tests pourra être enrichi avec de nombreux autres de façon à effectuer de petites phases d'optimisation plutôt qu'une gigantesque, souvent plus difficile à réaliser.

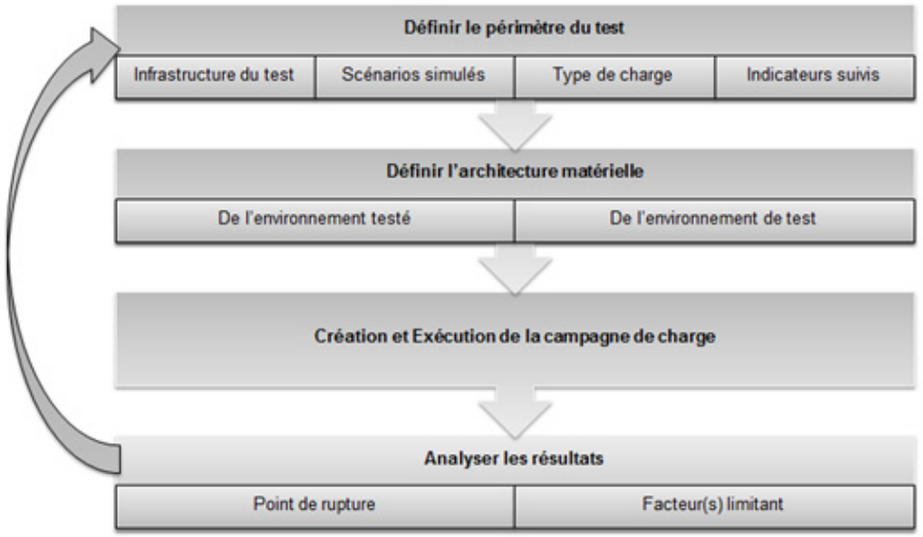
Un test de charge aide à mettre en évidence un ensemble de comportements et de tendances de l'application dans certains cas d'utilisation. Il faut bien garder à l'esprit qu'un test de charge ne donnera pas la liste des lignes de code à modifier pour améliorer les problèmes rencontrés. De même, il n'y a pas de règle absolue permettant de trouver instantanément la raison d'un problème de performance, c'est un travail de recherche de longue durée, demandant parfois de nombreuses expérimentations. Cependant les outils disponibles au sein de Visual Studio 2012 permettent de simplifier la mise en place de ces tests et surtout de comparer très simplement deux exécutions d'un test.

La prise en compte les tests de charge au sein d'un développement d'application requiert le respect d'une démarche articulée autour de quatre grandes étapes :

- La définition du périmètre du test, qui permet de définir la liste des scénarios à simuler, le type de charge que l'on appliquera et les indicateurs que l'on souhaite suivre pour analyser le résultat.
- La définition de l'architecture matérielle à la fois de l'environnement qui exécutera le test et de l'environnement qui sera validé, c'est-à-dire celui sur lequel la solution à tester sera déployée.
- La création et l'exécution de la campagne. Cette étape permet de créer ou mettre à jour les scénarios nécessaires à la simulation et d'exécuter le scénario de charge complet autant de fois qu'il sera utile pour obtenir un ou plusieurs résultats probant.

L'analyse des résultats, qui permet de comprendre les indicateurs. Au cours de cette étape, on identifie le point de rupture, c'est-à-dire le moment ou l'application ne répond plus aux exigences que l'on avait définies. On s'attachera également à mettre en évidence le ou les facteurs limitant qui sont à l'origine de cette rupture. Cette analyse viendra alors enrichir la batterie de résultats existante et permettra de suivre l'évolution des performances à travers le processus de création de l'application.

Ces différentes étapes peuvent être représentées par le schéma suivant :



Il est important de ne pas perdre de vue le fait que ce processus est cyclique, comme l'indique la flèche remontant à gauche du schéma.

3. Les outils de tests de la Plateforme Azure

A. Les solutions du marché

Au-delà des services proposés nativement sur la plateforme Azure, de multiples solutions de tests sont aujourd'hui disponibles sur le marché.

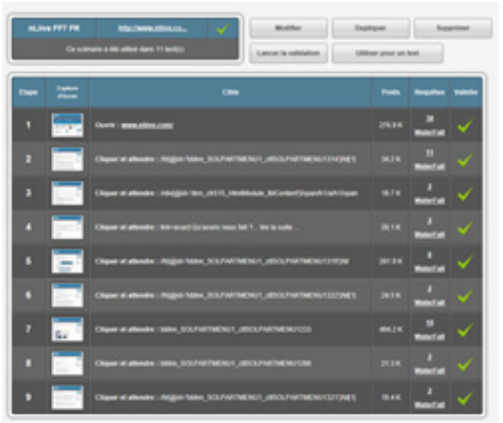
En voici une liste non exhaustive :

- Application Availability Management
  - ❖ Azure Watch - Paraleap Technologies
  - ❖ Azure Check - Apica Systems
  - ❖ Azureops - Opstera
  - ❖ Spotlight - Quest Software

- Application Performance Management (APM)
  - ❖ Gomez - Compuware
  - ❖ Dynatrace APM for Windows Azure - Dynatrace
- Application Load testing
  - ❖ Gomez Web Load Testing - Compuware
  - ❖ Loadstorm - Loadstorm.com
  - ❖ CloudTest - Soasta
  - ❖ CloudNetCare - CloudNetCare
  - ❖ Visual Studio Ultimate Stress Testing - Microsoft
  - ❖ HP LoadRunner - Hewlett Packard

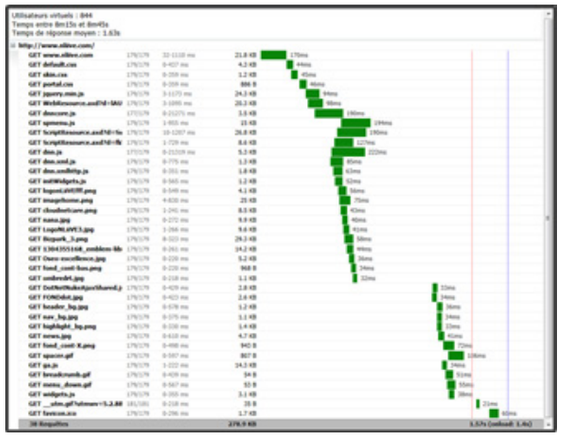
B. Exemple de solution de test de charge en mode SaaS : CloudNetCare

CloudNetCare est un outil de tests de charge en mode SaaS implémenté et déployé sur la plateforme Windows Azure. A ce titre, cet outil ne requiert donc aucune acquisition ou configuration de matériel ou de logiciel. En outre, la simplicité de son interface et le mode d'accès proposé en « self-service » le prédisposent à un usage par des utilisateurs sans expertise particulière.



Sur le plan technique la solution proposée par CloudNetCare se distingue par l'instanciation et pilotage de navigateurs sur des machines virtuelles instanciées depuis les datacenters Microsoft Azure. Ce choix permet de proposer une modèle de charge basé sur le comportement du browser plutôt que sur une simulation par envoi de requêtes HTTP pré-

enregistrées, et par conséquent, il permet de mesurer les temps de réponse tels qu'ils seraient réellement perçus par l'utilisateur.



L'interface de cet outil permet de créer les scénarios de tests par saisie d'URL ou par import de script généré à partir d'autres outils de « recording », d'exécuter ces tests en spécifiant le nombre d'utilisateurs souhaité, et d'afficher les résultats des tests avec une vue détaillée par page (temps de réponse, capture d'écran des erreurs, …) ainsi qu'un « waterfall chart » des requêtes HTTP pour chaque page web testée.

4. Outil et environnement de tests de charge pour le MTC

Au Microsoft Technology Center de Paris, avec notre partenaire InfiniteSquare, nous avons réalisé notre premier test de charge sur Azure, le 17 novembre 2010... Depuis de nombreux autres ont suivi et nous ont permis d'industrialiser complètement notre démarche.

A l'époque, différents outils permettant d'industrialiser la mise en place des tests de charge sur une plateforme Cloud ont été envisagés :

- Outil développé spécifiquement pour simuler la charge de l'application.
- Solution proposée en mode SaaS
- Produits traditionnels de « Load testing » comme ceux proposés dans Visual Studio Ultimate

Capitaliser sur une solution mature et bénéficier d'une expertise acquise de longue date sur les outils de simulation de charge Visual Studio sont les raisons qui, au Microsoft Technology Center de Paris, nous ont orientés sur cette dernière option.

Une fois ce premier choix effectué, d'autres questions se sont posées : fallait-il envisager d'utiliser ces outils de simulation à demeure (et bénéficier ainsi d'une latence cohérente avec la cible mais avec la nécessité de disposer de l'infrastructure suffisante pour les tests) ? Etait-il au contraire préférable de déployer son infrastructure de tests dans le Cloud (et éviter ainsi de payer la bande passante car l'application cible s'exécutait aussi dans Azure) ?

Nous avons fait le choix d'utiliser le Cloud Microsoft Azure pour héberger l'infrastructure de simulation. En effet, pour assurer une qualité en continue de l'application et réagir immédiatement en cas de soucis, il faut pouvoir s'assurer périodiquement qu'il n'y ait pas de régression en termes de performance. Au MTC, nous devons également être en mesure de monter rapidement une plateforme de tests de charge pour les clients qui en font la demande. D'où l'intérêt de pouvoir livrer une infrastructure de « load testing » à la demande avec un minimum de délai et d'investissement. En effet, une telle mise en place peut s'avérer coûteuse en ressources humaines et matérielles, et peut-être requise plusieurs fois durant le cycle de vie d'un projet…

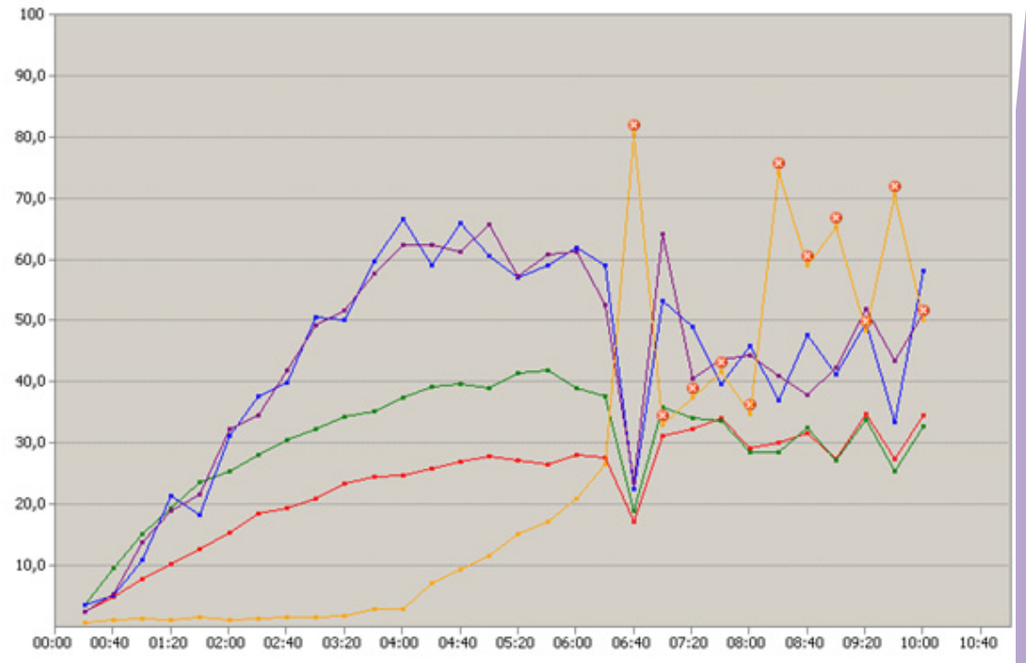
La plateforme Windows Azure offre donc clairement une réponse à ce type de scénario. L'approche que nous proposons de présenter dans ce document va donc se fonder sur l'expérience acquise au MTC sur le bon usage de la Plateforme Windows Azure pour héberger les services d'injection associés à Visual Studio Ultimate.

L'offre Visual Studio pour les tests de charge

1. Les tests de charge dans le contexte Visual Studio et TFS

A. Objectifs

Le test de charge est le seul test dont le résultat n'est pas binaire. Tous les résultats de test sont stockés dans la base de données « TFS Data Warehouse » ce qui facilite ensuite la création de rapports de synthèse de test de charge détaillés pour l'analyse de résultats de test performance. Le rapport généré peut être consulté au fur et à mesure ou à postériori, car stocké dans une base de données relationnelle. Si Team Foundation Server est utilisé, les résultats peuvent également y être stockés. Il est possible d'industrialiser l'exécution des tests de charge et la mise à disposition des rapports en utilisant conjointement Visual Studio et Team Foundation Server (service de build). Des modèles de rapports dédiés aux tests de charge sont disponibles dans ce dernier. L'intégration VS-TFS aide donc à la génération de rapports personnalisés qui peuvent être utilisés pour l'analyse des causes des échecs observés lors des tests de charge.



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
% User Time	Total	Processor	BL465C16-2		100	2,63	34,9	23,7
% User Time	Total	Processor	BL465C16-4		100	3,72	42,2	30,3
Bytes Received/sec	Intel[R] 82566DM Gigabit Network Connection	Network Interface	DC7800-003		10000000	3760 8	66872 6	43361 6
Bytes Received/sec	Intel[R] 82566DM Gigabit Network Connection	Network Interface	DC7800-004		10000000	2608 4	65867 1	43435 1
Avg. Response Time	Total	LoadTest: Request	DC7800-002		10	0,089	8,07	2,69

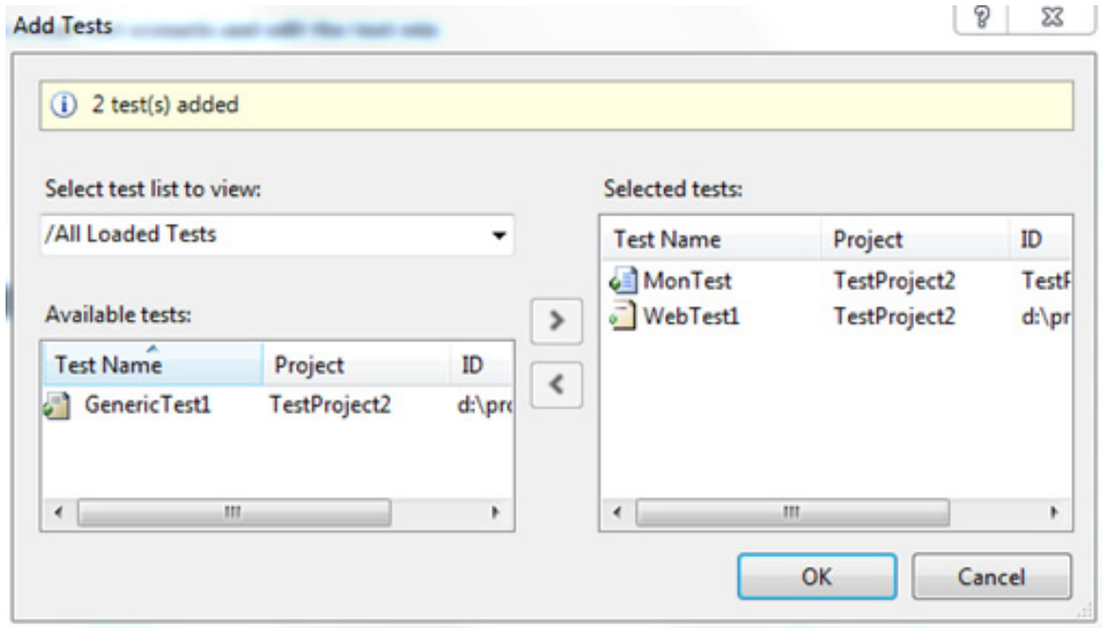
B. Cible

L'outillage Microsoft permet de tester en charge les applications spécifiques Web, Windows (client/serveur) en environnement .Net, Java, Php aussi bien que les logiciels (SharePoint, SQL Server ou autre SGDB, SQL Server Reporting Services…).

Seules contraintes :

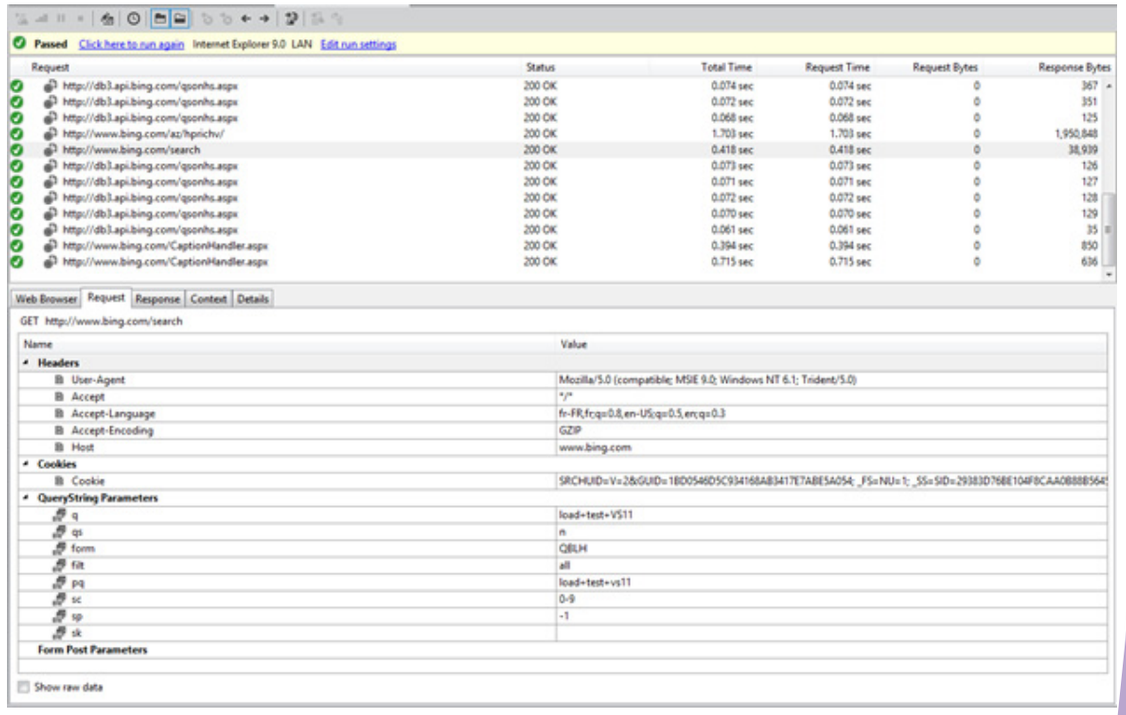
- Etre capable d'écrire un type de test simulant l'utilisation
- Etre capable de récolter les compteurs de performance

Un test de charge n'est rien d'autre qu'un test capable d'exécuter en boucle plusieurs autres types de test. Le test de charge est souvent utilisé avec des tests Web pour valider des applications intra/extra/internet mais peut très bien être utilisé avec des tests unitaires ou SQL pour valider la solidité de tout autre type d'application.



La création d'un test web et son exécution est donc souvent la première étape de la création d'un test de charge.

L'exécution unitaire d'un test Web dans Visual Studio 2012 permet de voir chaque page telle qu'elle est affichée dans le browser, la requête, la réponse et des informations liées au contexte de son exécution.



Exécution unitaire d'un test Web dans Visual Studio 2012

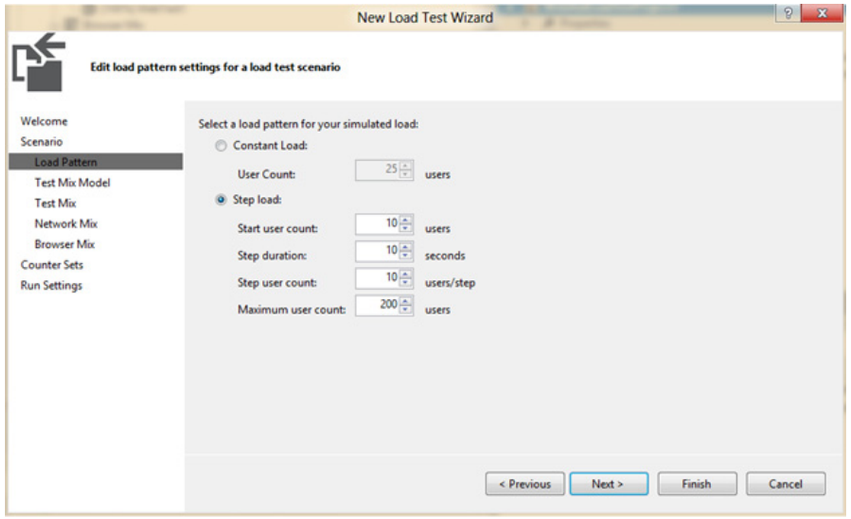
C. Préparation du Test

La phase de préparation du test est très importante et passe par plusieurs étapes.

- **Définition des scénarii :** un test de charge doit se rapprocher au mieux de la réalité, il est donc important lors de la création de scénarii de bien cerner le contexte ciblé. Seront ainsi pris en compte les navigateurs utilisés (une application web peut générer de l'HTML différent si elle s'adresse à un Internet Explorer sur Windows ou un SmartPhone , le temps d'attente entre les clics des pages (le but n'est pas de reproduire du clic en continu mais de simuler des utilisateurs) et la bande passante (l'objectif est d'avoir des temps de réponse acceptables pour tous les utilisateurs, si certains utilisent des modems 56k, ils seront à surveiller en priorité). Plusieurs scénarii peuvent être capturés pour un test de charge, et cha-

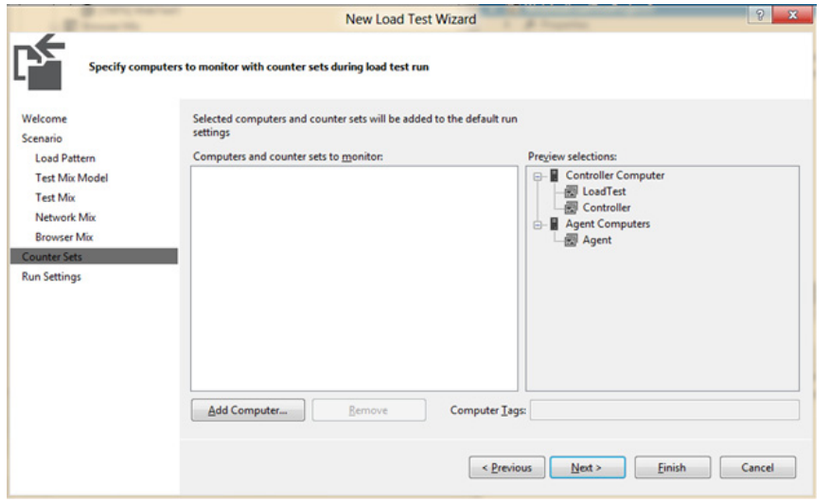
cun peut posséder un poids. Par exemple, si 80% des utilisateurs restent sur la page d'accueil de l'application, un scénario doit le simuler, si 10% créent un compte, 10% naviguent dans un catalogue, des scénarios doivent être créés avec une distribution similaire dans le test de charge.

- **Choix du type de charge parmi les trois proposés :**
    - Montante : le nombre d'utilisateurs augmente pendant toute la durée du test, même si la plateforme s'écroule
    - Constante : le nombre d'utilisateurs est stable sur la durée
    - Par Objectif : le nombre d'utilisateurs augmente jusqu'à atteindre un objectif défini (processeur à moins de 80%, temps de réponse à moins de 2 secondes…), lorsque l'objectif est atteint, la charge est lissée pour ne pas provoquer l'écroulement du système



Assistant de création de test de charge dans Visual Studio 2012

- **Les indicateurs de performance :** les compteurs Windows à prendre en compte dans les résultats de ce test de charge



Assistant de création de test de charge dans Visual Studio 2012

2. Création d'un test de charge

A. Création du scénario de charge

La création d'un test de charge se fait en deux étapes. La première consiste à utiliser un assistant permettant de configurer le test dans les grandes lignes. Il permet d'avoir accès facilement aux éléments principaux et doit être vu comme une « checklist » des paramètres standards que l'on ne doit pas oublier de configurer.

Comme tout type de test, les tests de charge doivent être créés à partir d'un projet de type Test. Pour ajouter un nouveau test de charge :

- Effectuer un clic droit sur le projet de type test
- Choisir Nouveau puis Nouveau test…
- Sélectionner Test de charge
- Entrer le nom désirée (d'extension .loadtest)
- Cliquer sur OK

L'assistant de création nous est alors proposé. Il propose un enchaînement de différentes étapes, chacune permettant de configurer un aspect différent du test.

Le premier élément à configurer est le scénario principal ainsi que ses paramètres. Après avoir choisi un titre, il est possible de choisir le profil de temps de réflexion. Pour rappel, le temps de réflexion correspond à la simulation de l'attente d'un utilisateur sur une page, telle que la lecture du contenu ou la saisie d'information dans un formulaire. Il est très important pour conserver une simulation de charge réaliste, proche du comportement réel des utili-

sateurs. Pour un test de charge, il est possible de choisir la manière dont ce temps de réflexion sera exploité. Nous pouvons choisir d'appliquer exactement les valeurs enregistrées dans les scénarii de tests, d'appliquer une légère modification à ces valeurs, ou de ne pas les utiliser pour appliquer une charge plus stressante à l'application. Le mode le plus utilisé reste la «distribution normale» qui permet d'avoir un comportement utilisateur varié. C'est également à partir de cette étape de l'assistant que nous pouvons choisir un temps d'attente entre deux exécutions de tests pour chaque utilisateur virtuel.

B. Choisir un modèle de charge

L'étape suivante consiste à choisir un modèle de charge. Celui-ci permet de définir combien d'utilisateurs virtuels seront simulés et à quel rythme ils seront ajoutés. Dans cet assistant, deux modes sont possibles : une «charge constante», qui définit un nombre fixe d'utilisateurs pour toute la durée du test ou une «charge dans l'étape», qui permet de demander une augmentation progressive du nombre d'utilisateurs. Un troisième modèle existe, la «charge par objectif», mais n'est accessible que par la suite, lors de l'édition du test de charge.

Chaque mode est intéressant et permet de valider un autre aspect de l'application. En général, la « charge constante » est utilisée pour vérifier que l'application se comporte correctement lors d'une utilisation «normale» de celle-ci, avec un nombre d'utilisateurs tel qu'on peut l'observer en production. C'est ce mode qui permet notamment de détecter des fuites mémoire potentielles.

Le mode « montant », lors duquel le nombre d'utilisateurs virtuels est augmenté régulièrement, permet de valider que l'application se comporte correctement lors d'une augmentation progressive du nombre d'utilisateurs. Il est possible de jouer sur la fréquence à laquelle les utilisateurs sont ajoutés pour simuler un pic de charge, par exemple lors des soldes pour un site de vente en ligne. Une autre utilisation de ce mode « montant » est de connaître le point de rupture de l'application et de répondre à la question : «A partir de quel nombre d'utilisateurs mon site ne se comporte plus correctement ?».

Pour configurer le modèle de charge constante, il suffit de le sélectionner et d'indiquer le nombre d'utilisateurs virtuels désirés.

En ce qui concerne le modèle de «charge dans l'étape», on indique le nombre d'utilisateurs au début du test, la durée (en secondes) pendant laquelle le nombre d'utilisateurs reste constant, le nombre d'utilisateurs à ajouter à chaque étape et enfin le nombre maximum d'utilisateurs qu'on souhaite atteindre.

C. Choisir un modèle de combinaison de tests

Dans la section précédente, nous avons choisi le modèle de charge qui définit le nombre d'utilisateurs que l'on souhaite simuler et la façon dont ils évoluent. Le modèle de combinaison de tests permet quant à lui de définir comment les différents scénarios de test seront répartis entre les utilisateurs virtuels.

En effet, lors de l'utilisation d'une application distribuée, tous les utilisateurs n'effectuent pas les mêmes opérations au même moment. Il est très probable par exemple que le processus d'inscription à un site web soit effectué moins fréquemment que celui permettant de consulter l'information qu'il contient.

Au sein d'un test de charge, on identifiera quatre modèles de combinaison de tests différents.

Le premier, basé sur le nombre total de tests effectués à un instant T, permet de définir que tel scénario est réalisé 20% du temps, qu'un autre l'est 40% du temps et que le reste est consacré à un dernier scénario. Par exemple, grâce à cette répartition, on pourra définir un test simulant des utilisateurs faisant une recherche sur un site 20% du temps et de la consultation et lecture de contenu le reste du temps.

Le second choix possible est basé non plus sur le nombre de tests, mais sur le nombre d'utilisateurs virtuels. Un pourcentage d'utilisateurs effectuera un tel ou tel scénario en boucle. Ce mode est utilisé pour simuler des profils d'utilisateurs utilisant toujours l'application pour la même fonctionnalité. C'est en exploitant cette combinaison de tests qu'on pourra simuler des populations d'utilisateurs. Par exemple, sur une application de gestion de la facturation, il est probable qu'un nombre fixe d'utilisateurs effectue les saisies de factures et uniquement cela, tandis que la validation de ces factures se fera par une autre population (qui elle aussi ne réalisera que ce type d'opérations).

Un troisième choix nous propose de répartir les scénarios de tests en se basant sur le rythme de l'utilisateur. Il s'agit là d'un compromis entre les deux premières combinaisons. En choisissant cette option, nous pourrons définir la liste des scénarios qui seront exécutés par chacun des utilisateurs et pour chacun de ces scénarios, combien chaque utilisateur en exécute par heure. Par exemple, il sera possible de simuler une population d'utilisateurs qui effectue une recherche d'information 3 fois par heure et ajoutent une facture 8 fois par heure.

La dernière combinaison qui nous est proposée est basée sur un ordre séquentiel de tests. Chaque utilisateur virtuel va exécuter un ensemble de scénarios dans un ordre prédéfini. Ce mode est très utile lorsque l'on souhaite simuler un flux de travail précis, tel qu'un scénario d'ajout de facture, suivi d'un scénario qui valide cette facture, puis d'un autre qui valide cette dernière, et ainsi de suite.

D. Définir la combinaison de tests, de réseaux et de navigateurs

Le modèle de combinaison de tests définit comment seront répartis les tests (ou scénarios) entre les différents utilisateurs. La combinaison de tests, quant à elle, représente la liste des tests qu'il faudra répartir.

Pour cela, il suffit de choisir les tests à inclure dans la liste en question et comment les distribuer entre les utilisateurs. L'outil de définition de la distribution peut changer légèrement en fonction du mode de combinaison de tests choisi dans l'étape précédente.

Il est possible d'ajouter ici n'importe quel type de test automatisé. Bien que les tests de charge sont souvent associés à des tests de performance web, l'objectif d'une charge est de mettre en avant les potentiels problèmes que l'on rencontre lors d'une utilisation classique ou non d'un système distribué et ce quelle que soit la technologie utilisée. Cela peut être un test web (pour les sites web et les services web), un test automatisé d'interface graphique, un test unitaire, etc.

Une fois la distribution choisie et figée pour un test, il est possible de cocher la case correspondant à ce dernier en fin de ligne pour faire en sorte que celle-ci ne soit pas modifiée automatiquement. Par exemple, 10% du temps les utilisateurs virtuels effectuerons une inscription, 25% du temps une navigation sur le site et 65% une recherche (les 3 scénarios sont des tests de performance web préalablement enregistrés et configurés).

Deux autres types de configuration sont possibles par rapport au comportement des utilisateurs virtuels.

La première concerne le choix d'un type de réseau à simuler. Par défaut tous les utilisateurs virtuels n'ont aucune limitation de réseau ou plus exactement aucune par rapport aux potentielles limitations physiques de la connexion utilisée par la machine exécutant le test. Il est bien évidemment impossible de simuler une connexion de type LAN en étant séparé de l'environnement testé par une connexion de type 3G! Il est par contre tout à fait possible de faire l'inverse et de simuler une connexion moins performante. Cette fonctionnalité est très utilisée pour pouvoir valider le comportement de l'application sur des périphériques mobiles principalement en obtenant des informations répondant à des questions du type : «Est-ce que le temps de réponse reste inférieur à 2 secondes sur un périphérique mobile ?»

Sur le même principe que la sélection des tests à exécuter il est possible de sélectionner les types de connexions à utiliser.

Il est également possible de configurer le type de navigateur utilisé par les utilisateurs virtuels. Les tests de performance web n'exécutent pas le rendu HTML, mais uniquement les échanges HTTP entre le client et le serveur. Néanmoins, il peut être intéressant de simuler un certain nombre d'utilisateurs virtuels utilisant des navigateurs différents, car l'application cible peut changer la quantité d'information qu'elle retourne en fonction de ce critère. Le type de navigateur est appelé User Agent.

E. Sélectionner des ensembles de compteurs

Les tests de charge avec Visual Studio ne se limitent pas à la simulation de la charge. L'objectif est également de récolter des indicateurs qui permettront d'obtenir des informations sur le comportement de l'application en cas de dysfonctionnement.

Les différentes versions des systèmes d'exploitation Windows proposent nativement un système permettant de surveiller leur comportement via des indicateurs : les compteurs de performance. Le plus généralement, on utilise l'outil intégré permettant de les afficher : le moniteur de performance (perfmon.exe).

Les compteurs de performances sont des variables mises à jour régulièrement avec la dernière valeur d'un indicateur. Les indicateurs les plus courants sont le pourcentage d'utilisation du processeur, la mémoire vive disponible, l'utilisation du réseau,… Il existe également des compteurs créés et maintenus par des applications tierces serveur, telles que Microsoft SQL Server ou encore Internet Information Services (IIS). On retrouvera ainsi des indicateurs tels que le nombre de requêtes SQL par seconde ou le nombre de requêtes web en file d'attente pour IIS.

Il est également possible de créer ses propres compteurs de performance Windows de façon à pouvoir les exploiter, soit quotidiennement dans le cadre du suivi de l'exploitation, soit au sein d'un test de charge.

L'organisation des compteurs de performance Windows est relativement simple : des compteurs sont organisés en catégories et chaque compteur peut être associé à une ou plusieurs instances. Par exemple, le compteur «% temps processeur» est contenu dans la catégorie «Processeur» et contient une instance par cœur du processeur de la machine.

Dans le cadre de la définition d'un test de charge, Visual Studio gère des ensembles de compteurs qui permettent de regrouper des indicateurs de performance par profil de machine que l'on souhaite suivre. Suivant que l'on souhaite observer le comportement d'un serveur de base de données ou d'un serveur Web, on ne s'intéressera pas aux mêmes informations. Il existe cinq ensembles de compteurs par défaut, disponibles à partir de l'assistant de création. Les ensembles «Application .NET», «ASP.NET», «IIS» et «SQL» portent des noms relativement explicites. «Application», quant à lui, permet d'accéder aux compteurs de performances classiquement utilisés pour suivre tout application qui ne rentre pas dans le cadre des quatre précédentes.

Bien que non modifiables à partir de l'assistant de création, ces ensembles seront personnalisables et pourront être complétés par d'autres via l'interface d'édition du test de charge.

F. Paramètres d'exécution

Les dernières informations à configurer via l'assistant de création sont quelques paramètres d'exécution. Il ne s'agit ici que des paramètres principaux et cette liste est complétée au sein de l'éditeur de test de charge, après sa création.

La durée d'exécution du test de charge peut être configurée de deux façons:

- En choisissant le mode «Durée du test de charge», de façon à fixer la durée.
- En sélectionnant le mode «Itérations de tests», pour définir le nombre de tests que devront exécuter les utilisateurs virtuels, peu importe le temps nécessaire.

Le mode «Durée du test de charge» permet d'indiquer la durée d'exécution mais également le «Temps de préparation» qui correspond à un «temps de chauffe» de l'application. En effet,

il est fréquent de mettre en œuvre un système de cache au sein d'une application distribuée. Le chargement du cache n'étant pas instantané, les résultats obtenus peuvent être incohérents par rapport à la «vitesse de croisière» d'utilisation de l'application. Le temps de préparation permet de définir un temps pendant lequel on estime que le cache n'est pas encore en place et d'écarter les résultats correspondant à cet intervalle de façon à ne pas perturber l'analyse de ceux-ci.

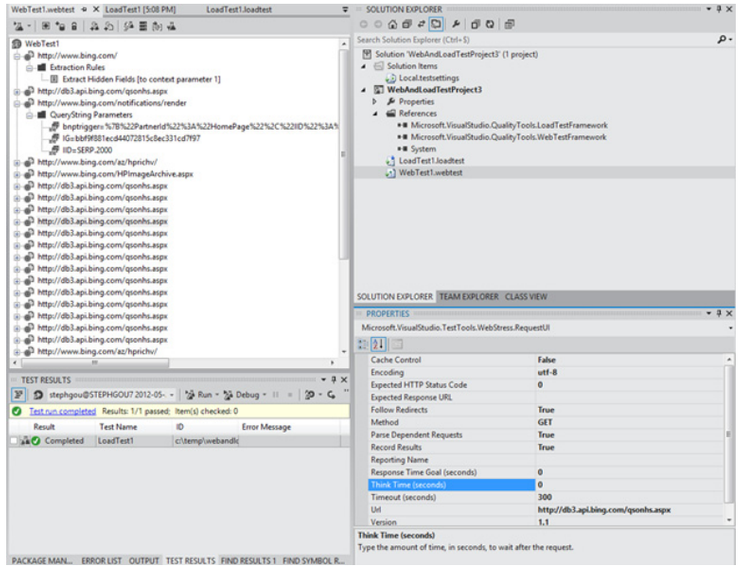
Lorsque l'on choisit le mode «Itération de tests», le seul élément sur lequel il est possible d'agir est le nombre de tests qui devront être réalisés par les utilisateurs virtuels et ceci indépendamment du scénario.

La partie «Détails» contient des informations plus spécifiques, telles que le taux d'échantillonnage qui correspond à l'intervalle de temps entre deux collectes des compteurs de performance, la description du scénario, l'activation de l'enregistrement du journal d'un test lorsqu'il échoue au sein du test de charge et le niveau des règles de validation des scénarios web que l'on souhaite activer.

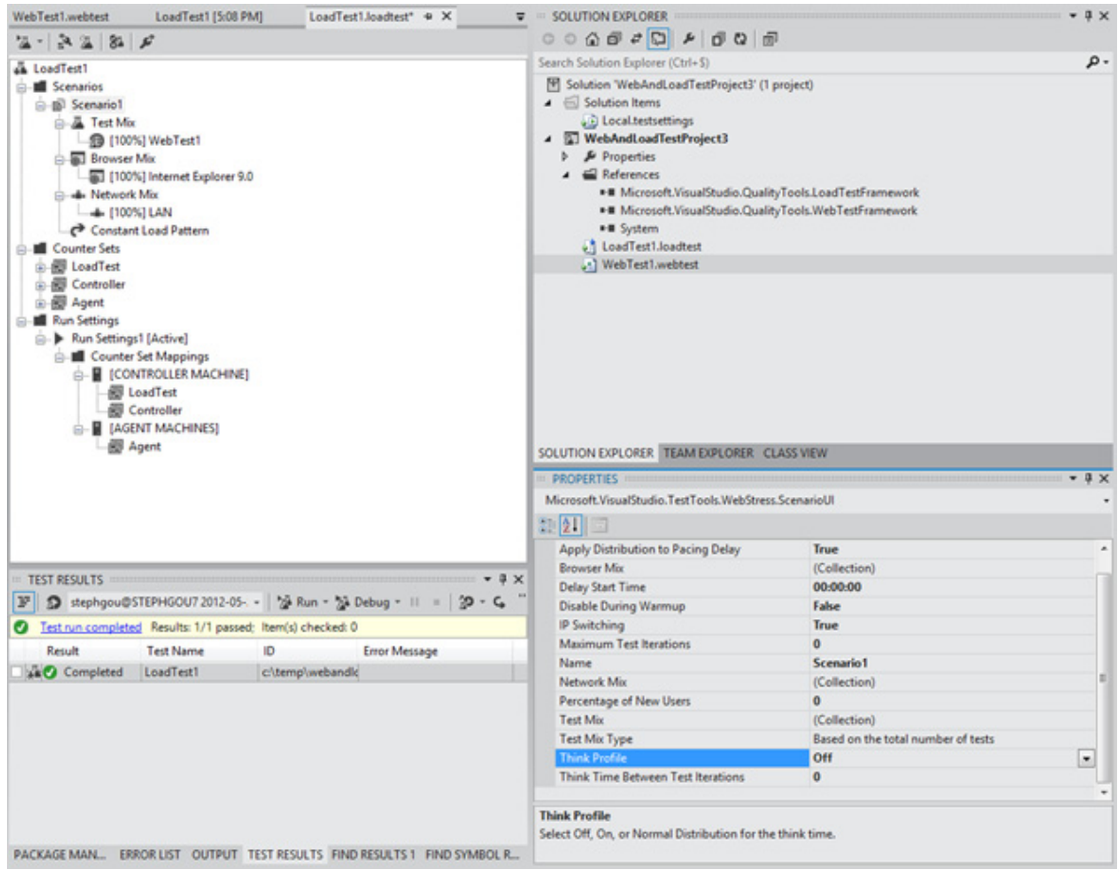
Ces derniers paramètres sont à prendre en considération lorsque l'on rencontre des problématiques de performance du test de charge en lui-même. Lorsque l'application est sous la charge par exemple, le réseau peut être saturé et 5 secondes ne suffisent plus à récolter l'ensemble des indicateurs ce qui traduira par une « interruption de données affichées » dans les graphiques. De même, si les agents de tests sont saturés, limiter le nombre de règles de validation qu'ils ont à traiter peut permettre d'alléger les machines qui les hébergent.

Paramétrage des tests dans Visual Studio

- Attention au "thinktime" dans le web test (Think Time 0)



Ou penser à le désactiver dans le load test : Think profile Off)



3. Maintenir un test de charge

A. L'éditeur de test de charge

De la même manière que dans le cas d'un test de performance web, un test de charge est matérialisé par un fichier XML. L'édition de ce fichier ne se fait pas directement en modifiant sa définition XML mais en utilisant un éditeur graphique permettant de personnaliser et configurer le test.

Tous les paramètres qu'il est possible de définir lors de la création du test au travers de l'assistant peuvent être directement mis à jour à partir de cet éditeur et il est fréquent d'apporter des modifications sur ces paramètres avant chaque exécution.

Cet éditeur est composé de trois sections :

- Les Scénarios : qui permettent de définir les scénarios de charge
- Les Ensemble de compteurs : qui contiennent la liste des catégories de compteurs activables sur les serveurs que l'on teste
- Les Paramètres d'exécution : contenant la liste des paramètres applicables sur les scénarios de charge

B. Les compteurs de performance

Lors de la création du test, il est possible de sélectionner les machines pour lesquelles on souhaite récolter des indicateurs. Ces indicateurs, ou compteurs de performance, sont organisés au sein d'ensembles de compteurs. Il en existe certains par défaut mais il est possible de modifier ces catégories ou d'en créer de nouvelles.

Ces catégories peuvent être appliquées sur plusieurs machines. Il faut donc prendre soin de ne pas sélectionner des compteurs qui n'existeraient pas sur toutes les machines. Par exemple, les compteurs de performance Microsoft SQL Server ne seront présents que si le produit est installé sur l'environnement.

On essaye en général de créer des ensembles de compteurs qu'on pourra réutiliser le plus souvent possible. On créera par exemple une catégorie transversale (contenant le processeur, la mémoire, l'utilisation disque, etc.) qui pourra s'appliquer sur la majorité voire l'ensemble des machines. On se contentera alors pour les ensembles de compteurs spécifiques de ne sélectionner que ce qui concerne le type de machine. Par exemple, on créera une catégorie SQL Server et une catégorie IIS 7.0. Pour pouvoir suivre une machine SQL Server, il suffira d'y appliquer la catégorie «transversale» et «SQL Server».

Une catégorie ou «Ensemble de compteurs» est matérialisée par un nom (ici «SQL»), un ensemble de catégories de compteur (ici «Memory»), un ensemble de compteurs pour chaque catégorie (comme par exemple «Available MBytes») et des règles de seuil pour chacun de ces compteurs.

Pour ajouter un compteur, il suffit d'effectuer un clic droit sur l'ensemble de compteurs et de choisir «Ajouter des compteurs».

Les règles de seuil permettent de définir des seuils critiques pour un compteur.

Il existe deux règles de seuils par défaut :

- « Comparer une constante » : Permet de comparer la valeur d'un compteur de performance à une valeur de constante. Il est possible de définir deux seuils, le premier indiquant la nécessite de lever un avertissement, le deuxième étant là pour indiquer une erreur.
- « Comparer des compteurs » : Permet de comparer la valeur du compteur sur laquelle la règle s'applique à celle d'un autre compteur. On peut également appliquer un multiplicateur à la valeur du compteur distant.

Ces règles de seuil peuvent être enrichie par des règles personnalisées qu'il est possible de créer en utilisant le Framework de Visual Studio.

C. Paramètres d'exécution

Les paramètres d'exécution ont plusieurs rôles. Ils permettent de définir les machines que l'on souhaite suivre et surtout à l'aide de quels indicateurs. C'est ici que l'on retrouve les relations entre les machines et les ensembles de compteurs définis précédemment. Les paramètres d'exécution permettent également de configurer un certain nombre d'éléments qui régiront le déroulement du test.

Il est possible et conseillé de créer plusieurs paramètres d'exécution dès que le besoin s'en fait sentir. Par exemple, on pourra utiliser le même test de charge pour valider deux environnements distincts. Il suffira alors de créer deux paramétrages différents, le premier récoltant les indicateurs du premier environnement, le second celui du deuxième. Il ne restera plus qu'à activer le bon paramétrage avant l'exécution du test en fonction de l'environnement (clic droit sur le paramétrage puis «Définir comme actif»).

Au-delà de la liste des machines à suivre et des indicateurs à récolter sur ces machines, les paramètres d'exécution permettent de définir des notions temporelles importantes. On retrouvera bien évidemment la durée d'exécution et le taux d'échantillonnage, déjà présent lors de la création via l'assistant, mais également une «durée de préchauffage» ainsi que de «refroidissement». Il s'agit de période avant («préchauffage») et après le test («refroidissement») pendant laquelle Visual Studio n'enregistre pas les valeurs des indicateurs récoltées. Cela permet de laisser le système mettre en place le cache et de ne pas inclure la période de fin pendant laquelle les sessions des utilisateurs virtuels sont arrêtées brutalement et qui pourraient contenir des informations non concluantes.

On retrouvera également la possibilité alternative d'exécution nommée «Itérations de tests». Si elle est activée, elle permet de ne plus se baser sur la durée du test de charge mais plutôt sur le nombre de tests exécutés, et ce indépendamment du nombre d'utilisateurs. Dès que N tests ont été exécutés, le test de charge est arrêté.

Il peut être intéressant de récolter des informations que l'on ne peut pas obtenir lors de l'utilisation des compteurs de performance dans le cadre de Microsoft SQL Server. SQL Server possède un système de trace permettant de connaître un certain nombre d'informations concernant notamment les requêtes exécutées (le nombre d'exécution, la durée moyenne, etc. et ceci par requête). La trace doit être activée au sein de SQL Server et comme toute trace, elle peut avoir un impact sur les performances. C'est pourquoi on évite en général de laisser cette fonctionnalité activée en permanence. Il est possible, au sein d'un test de charge, d'activer cette fonctionnalité à la volée, pour chaque exécution du test.

Le principe est simple :

- Visual Studio démarre l'exécution de la trace sur le serveur SQL

- Le serveur SQL enregistre la trace tout au long de l'exécution, dans un fichier sur un répertoire partagé
- A la fin de l'exécution du test de charge, Visual Studio récupère ce fichier et inclut les informations qu'il contient au rapport de test de charge global

Pour que Visual Studio puisse activer et désactiver la trace SQL Server à distance, il faut que le compte Windows qui exécute ce dernier possède les droits suffisants.

Les performances de l'exécution du test de charge ainsi que la taille des données récoltées possèdent une grande importance. On peut tenter d'optimiser ces éléments via un certain nombre de paramètres:

- Le modèle de connexion permet de spécifier si l'on souhaite que chaque utilisateur virtuel possède son propre canal de communication HTTP avec le serveur ou si l'on utilise plutôt un pool de connexions recyclées au fur et à mesure. Il existe également un niveau intermédiaire permettant de conserver la même connexion tout au long d'une itération de test.
- L'enregistrement des journaux (logs) d'exécution des tests web dans le cas où l'un de ces tests échoue. Il est possible d'activer ou désactiver cette fonctionnalité mais on perd alors la simplicité d'analyse lorsqu'un problème survient lors d'un test. Le plus simple est de jouer sur le nombre de journaux enregistrés. Visual Studio conservera alors les N derniers journaux.

L'outil principal d'analyse des résultats de tests est avant tout l'historique des différentes valeurs collectées pour les indicateurs sélectionnés. Bien que très important, cet historique peut rapidement devenir très volumineux et atteindre des limites en termes de stockage. Il est possible de choisir parmi trois options :

- Aucun : aucune donnée de temporisation n'est stockée. Seules les moyennes sont conservées
- Statistiques seulement : les moyennes sont conservées mais également certains échantillons (centiles)
- Tous les détails individuels : toutes les valeurs collectées sont conservées et peuvent être exploitées

En général, on essaye de conserver tous les détails. Il faut alors prévoir une base de données de stockage pouvant contenir une grande quantité de données. Il est complexe de déterminer la taille que prendra cette base car les données et leur quantité varient en fonction du nombre d'indicateurs collectés, du nombre de machines surveillées, de la fréquence à laquelle les données sont collectées, etc.

Dans la section «Général», on retrouve également une notion de la quantité d'information qui sera conservée concernant les erreurs survenant tout au long de l'exécution du test de

charge. Deux variables sont alors configurables :

- Le nombre maximal de règles de seuil : Le nombre d'alertes apportées par le dépassement d'un seuil défini sur un compteur.
- Le nombre maximal d'erreurs par type : il existe plusieurs types d'erreurs. Le nombre qui est indiqué ici limitera la conservation des messages d'erreurs aux N derniers pour chaque type.

4. Analyse des résultats

A. Exécution du test et gestion des résultats

Un test de charge est exécuté soit à partir du test lui-même, soit, comme n'importe quel type de test, à partir de la fenêtre «Affichage des tests».

Lors de l'exécution du test de charge, Visual Studio donne accès à un rapport préliminaire mis à jour en temps réel avec les informations collectées à partir des différents indicateurs. Ce rapport en temps réel n'est là que pour permettre d'avoir un aperçu de ce qui se passe. Les seules actions possibles lors de l'exécution du test sont l'affichage de données, la construction de graphiques et l'arrêt manuel du test.

Très souvent, il est inutile de continuer l'exécution d'un test de charge jusqu'à la fin réelle du temps alloué à l'origine car l'application ne répond plus, ou est trop surchargée par exemple. Dès qu'on estime qu'aucune nouvelle donnée ne pourra être cohérente pour l'analyse, il est préférable d'arrêter le test pour économiser de l'espace de stockage, d'autant plus que cela n'aura absolument aucun impact sur les données récoltées, l'analyse pourra se faire de la même manière.

Le rapport présenté lors de l'exécution correspond en tout point à celui qui est disponible en fin de test, modulo une fiche de résumé supplémentaire et quelques fonctionnalités.

Ce rapport interactif est divisé en deux parties majeures:

- La partie haute qui permet de changer de vue dans la partie gauche et un certain nombre d'outils sur la partie droite
- La partie basse qui comprend les données affichées suivant la vue sélectionnée.

Les différentes vues comprennent le résumé, donnant accès à des informations générales, les graphiques, les tables et le détail.

Toutes les données sont stockées dans une base de données, en général local à la machine exécutant le test. Chaque résultat associé à une exécution de test de charge est conservé et il est possible de consulter un rapport d'un test précédemment effectué.

La gestion de ces résultats se fait à partir du fichier de définition de test de charge, dans le menu contextuel de presque tous les éléments de l'arborescence. Cela donne accès à un as-

sistant permettant d'afficher la liste des tests réalisés, par scénario de test de charge. Il est possible d'ouvrir un résultat de tests pour accéder à son rapport ou encore d'exporter les informations d'un ou plusieurs résultats dans un fichier que l'on pourra réimporter dans une autre base de données de test de charge si besoin. Idéal pour faire des sauvegardes sélectives sans pour autant sauvegarder l'intégralité de la base de données.

Il n'est pas possible de «nommer» un résultat de test de charge. Il est par contre possible et recommandé d'indiquer une description explicite d'autant plus qu'elle apparaît dans la liste de l'historique des tests. Pour ajouter une description, il suffit de cliquer sur le bouton «Ajouter une note d'analyse» dans le rapport de test de charge au niveau de la barre d'outils dans la partie haute du rapport.

B. Résumé

Le résumé est accessible uniquement lorsque le test de charge est complétement achevé. Il centralise l'accès aux informations les plus significatives. On retrouvera les statistiques principales telles que les dates et heures de début et de fin d'exécution, la durée du test, le nombre d'agents de tests et le jeu de paramètres utilisés pour l'exécution.

Ce rapport donne également accès à la liste des cinq pages les plus lentes. Cette option est nommée «95% du temps de réponse de la page (s)», ce qui signifie que ne sont pris en compte que les 95 premiers pourcents du total des requêtes les plus rapides. En d'autres termes, la moyenne de temps de réponse affichée ici a été calculée après avoir retirée de la liste les requêtes les plus lentes. Ceci est une pratique courante dans l'analyse de données de tests de charge car on estime que les requêtes les plus lentes ne donnent pas une information significative et qu'il s'agit souvent d'exceptions.

Une autre section donne la liste des cinq tests les plus lents (nous parlons ici de tests de performance web). Les valeurs affichées ont été calculées en suivant la même règle que celle expliquée précédemment.

Sont également présentes des informations concernant la totalité des tests, notamment leur durée moyenne en seconde ainsi que le nombre total d'exécutions réalisées au cours du test de charge global. Une donnée intéressante lors de l'analyse concerne le pourcentage de tests exécutés ayant donné lieu à au moins une erreur. Cela peut signifier deux choses : soit la charge importante a entraîné une mauvaise exécution de ce test (temps de réponse trop lent, timeouts, inter-blocage en base de données etc.), soit le test lui-même comporte une erreur de conception. Si ce pourcentage d'erreur est très bas (en dessous de 3%), il s'agit très certainement d'exceptions pouvant facilement être mises de côté et non prises en compte lors de l'analyse.

Ce résumé nous propose également une liste plus détaillée, contenant les résultats pour chacune des pages exécutées. Attention, comme cela est le cas pour la plupart des indications «par page» que l'on pourra retrouver au sein du rapport de test de charge, une page

correspond à une ressource. De fait, on pourra retrouver dans la liste des pages : des images, des feuilles de style et bien évidemment des pages html, asp.net, …. Pour chacune de ces pages, on connait ici le temps d'exécution moyen et le nombre d'exécutions. Ce type d'information est important pour l'analyse car une page exécutée une seule fois peut se permettre d'être plus «lente» que d'autres exécutées plus fréquemment.

Figurent aussi des informations sur les transactions provenant des tests de performance web, s'il y en a. Dans le cas où l'application ne possède pas de nombreuses pages différentes, mais où tout se réalise avec la même page (par exemple, la consultation, l'achat, l'ajout d'un produit se fait via la même page «Produit.aspx» avec des paramètres différents), il est conseillé d'utiliser des transactions pour obtenir des statistiques parlantes.

Le résumé se termine sur des éléments concernant le système testé, l'environnement de test (contrôleur et agents) ainsi que la liste des types d'erreurs rencontrées.

C. Graphiques

Une autre vue disponible pour les tests de charge est la vue «Graphiques». Comme son nom l'indique, elle permet d'afficher les valeurs récoltées pour les différents indicateurs sélectionnés sous forme de graphiques.

Les graphiques permettent de suivre la tendance de l'évolution de certaines valeurs clés. Ils sont également utilisés pour mettre en avant la corrélation de certains indicateurs. Il est fréquent d'afficher sur le même graphique l'évolution de l'utilisation du processeur, de la ram et le temps de réponse. Très souvent, le temps de réponse diminue lorsque le processeur est saturé et inversement.

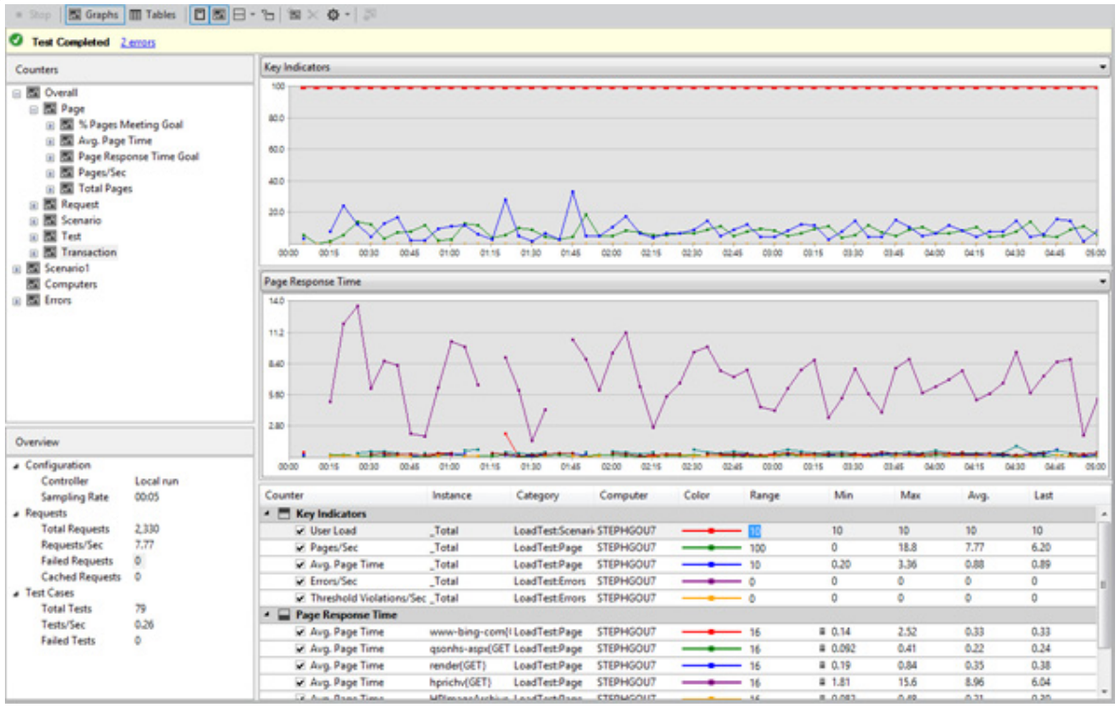
Plusieurs graphiques sont proposés par défaut et permettent d'obtenir certaines informations sans avoir à construire soi-même ces représentations.

Trois graphiques sont en rapport avec le temps de réponse. Le « temps de réponse de la page » inclut automatiquement toutes les pages testées et l'évolution du temps de réponse moyen pour chacune d'elles. Le « temps de réponse de la transaction » et le « temps de réponse du test » contiennent exactement les même informations, pour les transactions s'il y en a et pour les tests exécutés par les utilisateurs virtuels.

Le graphique « Indicateurs clés » est celui qui est le plus souvent suivi en temps réel. Il regroupe : le nombre d'utilisateurs virtuels (autrement dit la charge appliquée sur l'application), les nombre de pages vues par secondes, le temps de réponse moyen pour l'ensemble des pages, le nombre d'erreurs et de dépassements de seuil se produisant par seconde. Ceci permet d'avoir d'un seul coup d'œil un bon aperçu du déroulement du test de charge.

La santé des environnements engagés dans le test de charge doit être contrôlée constamment pendant et analysée après l'exécution du test. Pour cela, deux graphiques sont disponibles : le premier, permettant de visualiser l'état des machines que l'on teste, c'est-à-dire

la liste des serveurs sous la charge, est nommé « Système testé ». Le second, permettant quant à lui d'obtenir les informations relatives aux systèmes hébergeant les utilisateurs virtuels, est nommé « Contrôleur et agents ». Ces deux graphiques contiennent automatiquement l'ensemble des indicateurs de mémoire disponibles et d'utilisation du processeur pour toutes les machines suivies par Visual Studio au sein du test de charge.



Vue Graphe présentant les résultats d'un test de charge VS 2012

La capture d'écran ci-dessus représente le graphique « Système testé ». Il permet de remarquer que l'écran du rapport correspondant à la section «Graphiques» est organisé en trois parties.

La première à gauche contient l'ensemble des compteurs de performances accessibles et utilisables au sein des graphiques. Il existe deux types de compteurs dans cette arborescence : les compteurs internes à Visual Studio et les compteurs de performance Windows. . Les compteurs internes à Visual Studio fonctionnent sur le même principe que les compteurs de performance, par contre, ils ne sont calculés que pendant le test de charge. Contrairement aux compteurs de performance Windows, il n'est donc pas possible de connaitre la valeur d'un compteur interne Visual Studio en dehors du contexte d'un test de charge.

L'objectif de ce document n'est en aucun cas de donner une explication détaillée de tous les indicateurs présents et utilisables pour l'analyse d'autant plus qu'ils sont relativement auto

descriptifs. On retrouve les compteurs internes à Visual Studio dans les sections «Général», «Erreurs» et autant de sections que de scénarios de charge définis (en général, un seul). La section «Ordinateurs» quant à elle, correspond à l'intégralité des compteurs de performances Windows définis pour chaque machine, qu'elle fasse partie de l'infrastructure liée à l'application ou de l'environnement de test.

Pour chaque Scénario de charge, on retrouve l'ensemble des tests et des transactions ayant été exécutés. Pour chacune de ces transactions, il existe un ensemble de compteurs tels que : le nombre de tests échoués, réussis, le détail pour chaque sous-requête ou chaque page, et ainsi de suite. Est également mis à notre disposition le nombre de tests en cours d'exécution et la charge utilisateur (le nombre d'utilisateurs virtuels présents).

La section regroupant les informations relatives aux erreurs permet d'avoir une vision fine de l'évolution des erreurs en fonction de leur type. Par exemple, il est possible de connaître la courbe du nombre d'erreurs http tout au long du test de façon à savoir si ce nombre est lié à l'utilisation processeur de telle ou telle machine, ou à n'importe quel autre indicateur.

La partie de l'arborescence concernant les compteurs de performance Windows est organisé par ordinateurs, puis par catégorie de compteurs, par compteurs et enfin par instance de compteur.

La partie droite de la section «Graphiques» contient les graphiques en tant que tels et la légende associée. Il est très simple de modifier ou créer un nouveau graphique par un simple clic droit puis «Ajouter un graphique».

Pour ajouter un nouveau diagramme sur le graphique, il suffit d'effectuer un glisser déposé d'un compteur à partir de l'arborescence gauche, ou de double- cliquer sur celui-ci. Il apparaîtra automatiquement sur la grille ainsi que la légende qui lui est associée. Cela permet de construire très simplement les graphiques nécessaires à l'analyse.

D. Tableaux

Autre outil très utilisé pour l'analyse d'un résultat de test de charge, les tableaux permettent d'avoir des statistiques différentes de celles que l'on retrouve au sein des graphiques, par exemple Tests, Transactions ou encore Pages, qui donnent des informations principalement en rapport avec le temps de réponse. Les tables «Erreurs» et «Seuils» concernent plus précisément la gestion des erreurs.

La table «Traces SQL» n'est renseignée et ne contient de valeurs que lorsque l'exécution du test est terminée. Elle permet de connaitre la liste des requêtes SQL effectuées sur la base de données SQL Server pour laquelle la trace a été activée dans la configuration du test de charge. Pour chacune de ces requêtes on obtiendra le temps d'exécution, le nombre de lectures et d'écritures effectuées en base, l'utilisation du processeur ainsi que l'heure de début et de fin de requête. Ces informations peuvent être vitales dans certains cas pour identifier

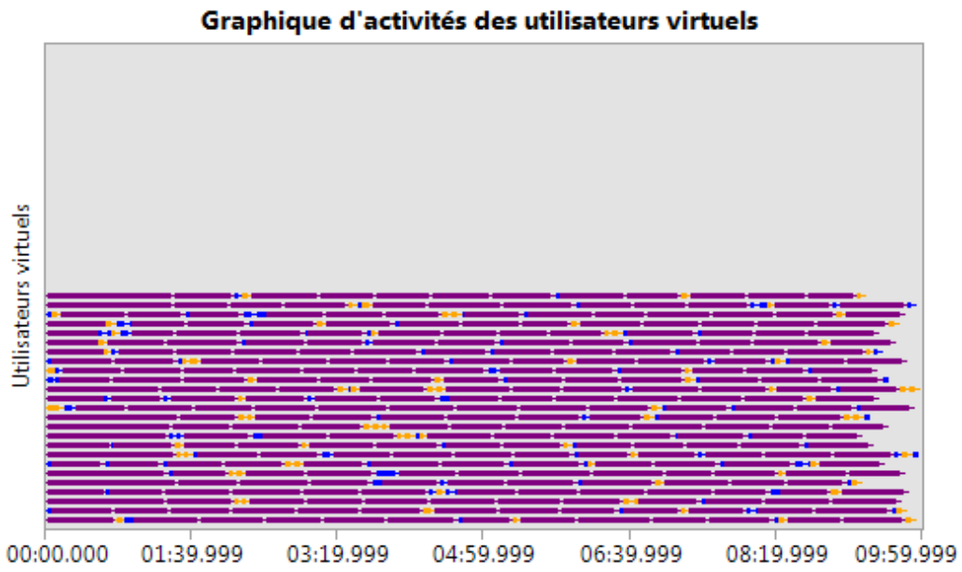
la cause de ralentissements voire de dysfonctionnements de l'application.

Enfin, la table «Détail du test», permet d'avoir des informations unitaires pour chaque test exécuté par les utilisateurs virtuels comme l'heure de début, la durée et le résultat (réussite ou échec).

Contrairement aux graphiques, il n'est pas possible de modifier une table existante ni d'en créer une nouvelle.

E. Détail

La vue Détail apporte des informations importantes pour l'analyse puisqu'elle permet d'obtenir une représentation visuelle de la densité d'exécution d'un test par rapport à d'autres. Autrement dit, on aura une cartographie à travers le temps et les utilisateurs virtuels des scénarios en cours d'exécution à un instant T.



Ci-dessus, un exemple de graphique de détail disponible au sein d'un rapport de test de charge. On peut remarquer qu'il y a eu ici l'exécution de trois scénarios distincts (en jaune, bleu et violet). Chaque ligne correspond à un utilisateur virtuel précis. Dans le cas d'un grand nombre d'utilisateurs simulés, chacun d'entre eux, sera tout de même représenté sur ce graphique.

Il est utile d'exploiter ce rapport qui permet de répondre à de fréquentes questions lors de l'analyse d'un problème de performance identifié au cours d'un test de charge comme par exemple : «Quels tests étaient en cours d'exécution au moment où les problèmes ont commencé ?».

Ce graphique est configurable via un volet à gauche permettant :

- De sélectionner les données à afficher : pouvant être les tests comme sur la capture ci-dessus, ou les pages, donnant un niveau de détail encore plus avancé
- De choisir les tests ou pages à afficher sur le graphique en fonction des données affichées
- De filtrer par type de résultat pour chaque test

F. Rapports Excel

Un rapport de test de charge permet d'analyser le test lui-même. Tous les outils qui ont été abordés jusqu'à présent permettent de comprendre ce qui s'est passé au sein du même test.

Cependant arrive un moment où il faudra comparer deux exécutions différentes du même test de charge. Une première exécution permet souvent de connaitre les faiblesses du système que l'on teste. S'ensuit une phase de développement et d'optimisation pour atténuer voire faire disparaître ces faiblesses. Cette phase d'optimisation sera validée par un second test de charge.

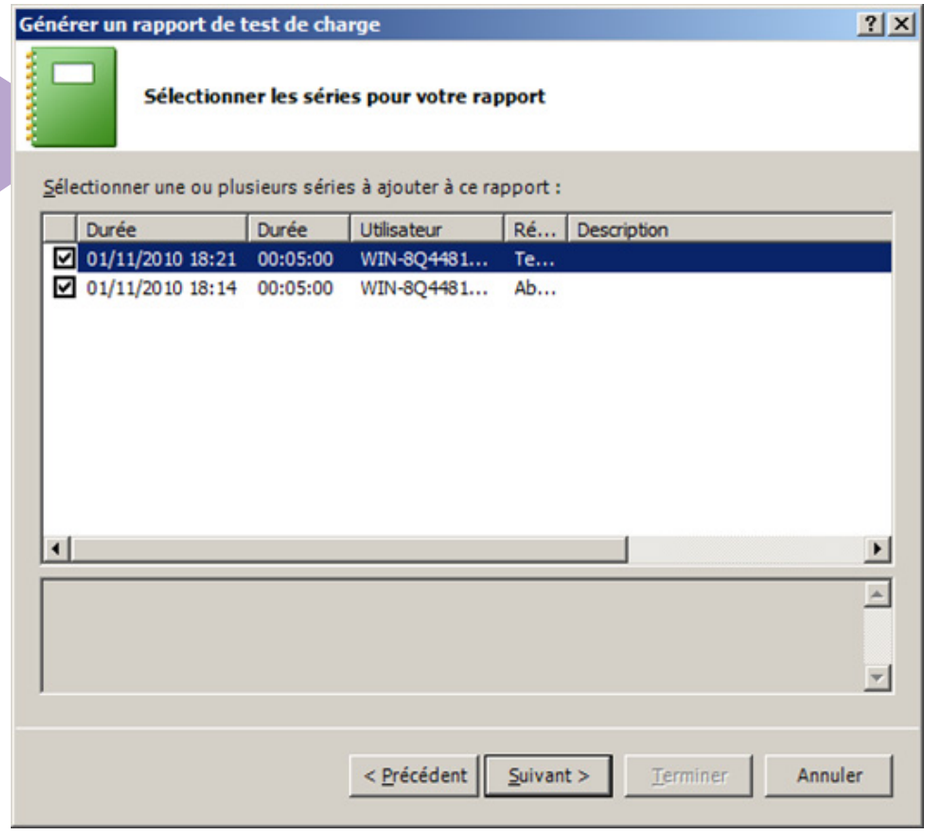
Une fois ce second test effectué, il est possible de demander la génération de rapports au sein de l'outil Microsoft Excel qui permettront de mettre en évidence les différences entre les deux exécutions, et par conséquent les gains et les pertes de performances.

Visual Studio permet de choisir entre deux modes de génération de graphiques. Le premier permet de créer une comparaison entre deux rapports précis tandis que le second créera un ensemble de graphiques de l'évolution des performances à travers le temps en se basant sur deux rapports ou plus.

La création de ces rapports Excel se fait de deux manières : soit à partir d'un rapport de test de charge dans Visual Studio en utilisant le bouton «Créer un rapport Excel» disponible dans la barre d'outils supérieure du rapport, soit à partir d'Excel en utilisant l'onglet «Load Test».

Si l'on choisit de créer le rapport à partir d'Excel, on obtiendra une étape préliminaire nous demandant une chaine de connexion SQL vers la base de données stockant les résultats des tests

Une fois le choix de rapport effectué, il suffit de nommer le rapport, de sélectionner le test de charge puis de choisir les résultats à comparer, comme le montre la capture d'écran suivante :

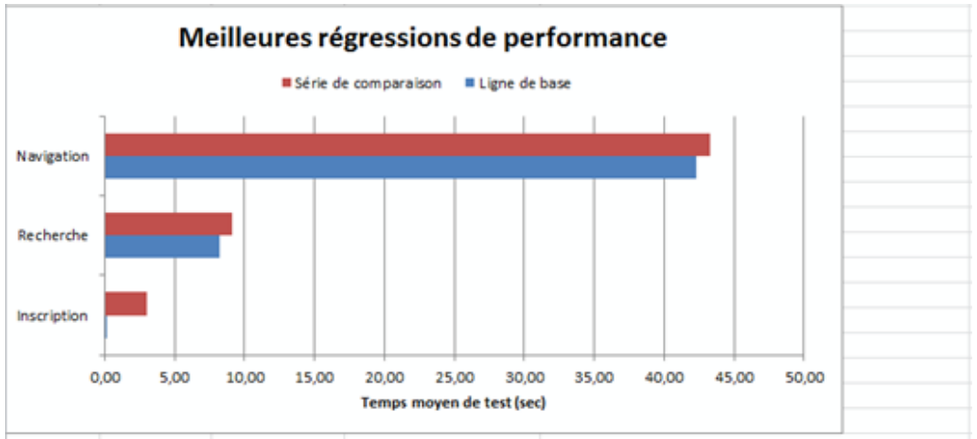


Le principe même de ces rapports sous Excel est de comparer les valeurs récoltées lors des différentes exécutions pour les différents indicateurs suivis. C'est pour cette raison que l'étape suivante permet de choisir les compteurs que l'on souhaite voir apparaître dans la comparaison au sein du rapport Excel.

Ci-dessous, un exemple de sélection de compteurs. On remarque qu'il est possible de choisir à la fois des compteurs de performance Windows et des compteurs internes à Visual Studio.

L'outil génère alors un certain nombre de feuilles dans le fichier Excel. Ce nombre sera différent en fonction du type de rapport choisi et du nombre de compteurs sélectionnés.

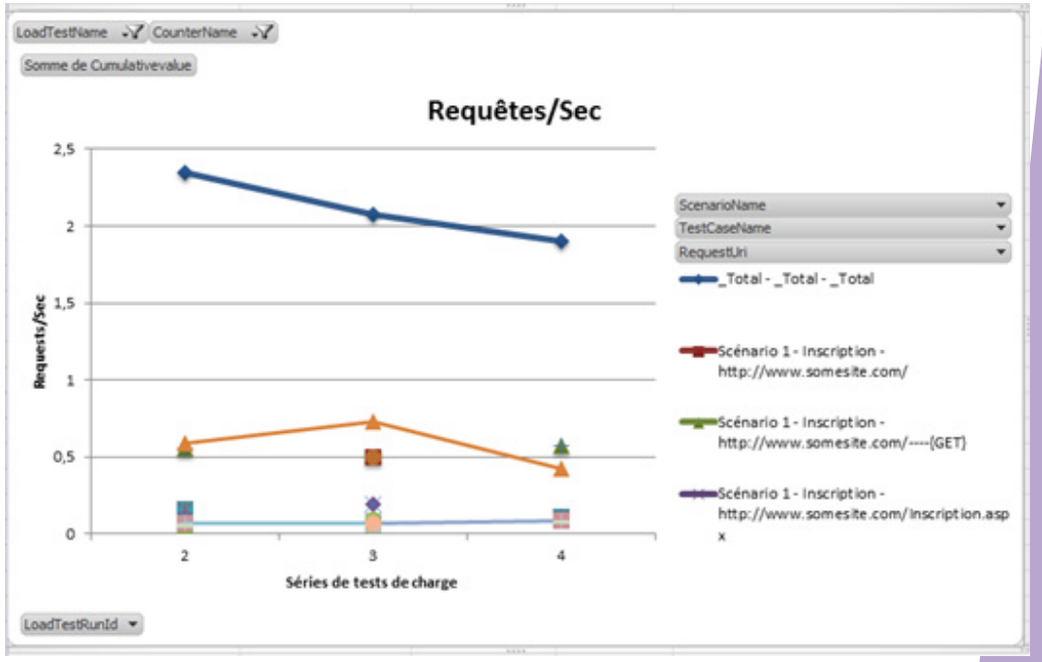
Voici un exemple de graphique généré dans le cas d'une comparaison. Il s'agit du graphique de comparaison de temps de réponse des pages du site :



Scénario	Cas de test	Ligne de base	Série de comparaison	Variation en % par rapport à la planification initiale
Scénario 1	Inscription	0,11	2,99	-2643%
Scénario 1	Recherche	8,22	9,10	-11%
Scénario 1	Navigation	42,34	43,25	-2%

On retrouvera également une comparaison plus détaillée pour chaque feuille, disponible en plus du graphique.

Dans le cas d'un rapport d'évolution, les graphiques générés ne sont plus des histogrammes :



5. Infrastructure de tests de charges

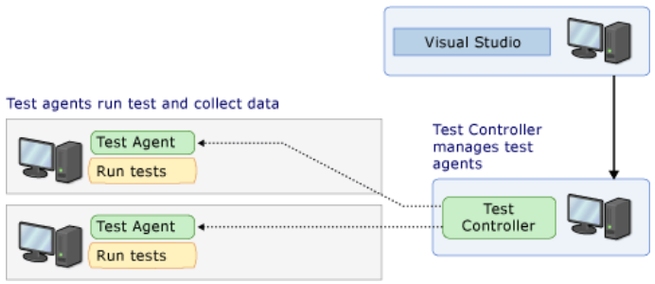
Visual Studio Ultimate permet de générer des tests de performance des applications Internet. Il permet aux entreprises d'améliorer la qualité de service en testant de manière plus efficace et plus proche de la charge future des serveurs, le comportement de leurs applications Web. Visual Studio simule la charge utilisateur sur les applications Internet et les serveurs pour donner aux équipes "qualité" des informations précises sur le comportement de leurs applications dans un mode de fonctionnement proche de la réalité. Les résultats permettent de connaître les performances de ses applications, de dimensionner les serveurs pour des performances optimales, en un mot, d'anticiper. Visual Studio fonctionne avec toutes les technologies d'applications Web.

Les tests web Visual Studio peuvent être également être utilisés dans les tests de performances en les ajoutant dans les tests de charge Visual Studio. Une documentation accompagne le produit et intègre une liste de compteurs, de seuils prédéfinis et leur effet sur les serveurs en charge.

La combinaison de Load Test et des tests Web fournit des règles de validation et d'extraction qui sont essentielles pour rendre les scripts robustes : les règles d'extraction aident à définir comment paramétrer les données générées dynamiquement.

Pour les mesures de performance du code et de la mémoire sur un comportement spécifique d'une portion du code en charge, il est recommandé d'ajouter les tests unitaires dans les tests de charge et d'activer l'analyseur de performance Visual Studio Team System pour pouvoir faire un « profiling » prenant en compte le comportement en charge.

Jusqu'à présent, l'ensemble des explications étaient principalement centrée sur l'exécution à l'aide d'une machine hébergeant les utilisateurs virtuels. Ce type d'infrastructure peut très bien répondre à un petit nombre de tests de charge, tendant à valider un nombre restreint d'utilisateurs simultanés. Dans ce cas simple, Visual Studio est en charge de l'exécution du test de charge, il joue le rôle à la fois du contrôleur de test et de l'agent de test. Il est important de savoir que ce mode vous permettra de simuler jusqu'à 250 utilisateurs virtuels simultanés mais pas au-delà. Dans un cas plus complexe, demandant un nombre plus important d'utilisateurs, il est préférable et quasiment inévitable de mettre en place ce qui est appelé un « Rig de tests



Test controller and agents Visual Studio 2012

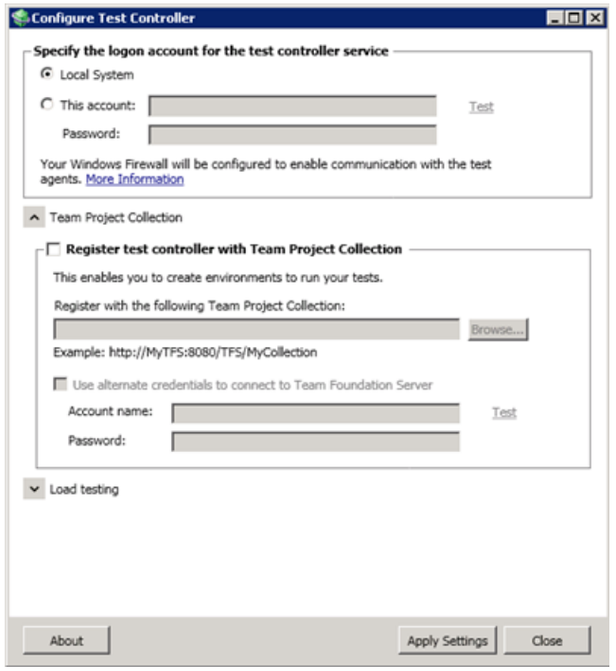
Un « Rig de tests » est constitué d'au moins un contrôleur de test et d'un agent.

Un contrôleur de test est un service Windows qui a pour rôle de connaître un ensemble d'agents de tests et de les piloter. Les agents de tests sont également des services Windows qui quant à eux ont pour rôle d'héberger et de simuler les utilisateurs virtuels.

Le déploiement d'un « Rig de tests » se fait en général sur des machines physiques car le but est d'optimiser les performances. Ceci dit, depuis la version 2010 de Visual Studio, le mode de licence permet de profiter d'un ensemble de machines moins puissantes, voire de machines virtuelles.

Avec le Visual Studio 2010 Load Test Feature Pack, les abonnés Visual Studio 2010 Ultimate avec abonnement MSDN disposent désormais d'un nombre illimité d'utilisateurs virtuels inclus dans leur abonnement pour effectuer leurs tests de performance et de montée en charge.

Pour permettre l'exécution du test au sein d'un « Rig de tests », il suffit de configurer le fichier « .testsettings » de la solution Visual Studio et de demander une exécution sur un contrôleur.



Configuration d'un contrôleur de test avec Visual Studio 2012

Cette exécution ne concerne pas que les tests de charge. Une fois le contrôleur sélectionné, tous les types de tests automatisés seront exécutés sur les agents du contrôleur, que ce soit un test web, un test unitaire ou encore un test d'interface graphique automatisé.

Construire une infrastructure de tests de charge pour la plateforme Windows Azure

1. La plateforme Azure

A. Description de la plateforme Azure

La Plateforme Windows Azure est la réponse qu'apporte Microsoft aux nouvelles architectures et applications d'entreprise adossées au Cloud Computing. Ainsi la plateforme Windows Azure propose pour le Cloud différents services en s'inspirant du modèle qu'offre la plateforme Windows Server pour l'entreprise.

Cette plateforme comprend un socle d'exécution, Windows Azure, que l'on peut considérer comme l'équivalent d'un système d'exploitation dans le Cloud. De la même manière que l'on développe des applications pour Windows Server, il est désormais possible de développer des applications Cloud pour Windows Azure qui se charge de les exécuter.

Windows Azure fait donc office d'environnement de développement, d'hébergement (hosting) de services et de gestion de services pour la plateforme Windows Azure. Windows Azure offre aux développeurs des capacités de calcul et de stockage à la demande pour l'hébergement, la mise à l'échelle et la gestion d'applications Web sur Internet par le biais des centres de données Microsoft.

Windows Azure est une plateforme flexible qui prend en charge plusieurs langages et permet dès aujourd'hui aux développeurs de concevoir et réaliser des applications hybrides, vivant en partie dans l'entreprise et en partie dans le Cloud. Pour développer des applications et des services sur Windows Azure, les développeurs peuvent utiliser leur connaissance actuelle de Microsoft Visual Studio. En outre, Windows Azure prend en charge les normes et standards, protocoles et langages les plus répandus, notamment SOAP, REST, XML, Java, PHP et Ruby.

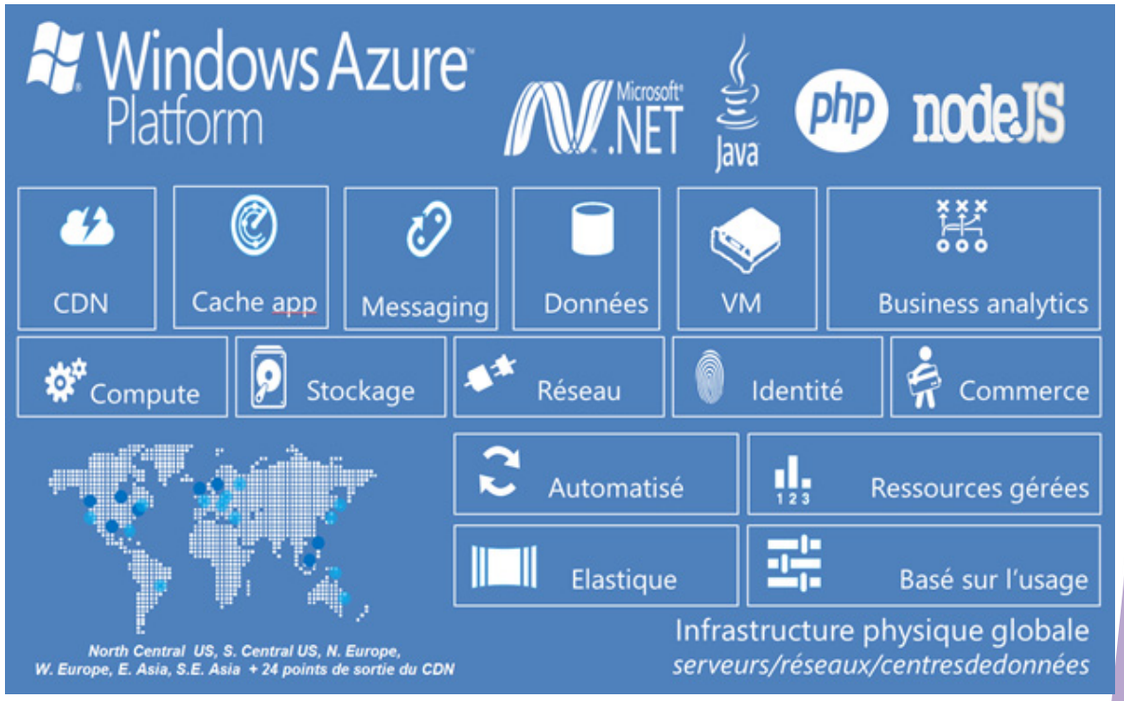
La plateforme propose également un ensemble de services (comme SQL Azure ou Service Bus) destinés à être utilisés individuellement ou de manière combinée par les développeurs d'applications.

Microsoft patterns & practices proven practices for predictable results Pour de plus amples détails, vous pouvez consulter le guide

DEVELOPING APPLICATIONS FOR THE CLOUD ON THE MICROSOFT WINDOWS AZURE™ PLATFORM1.

Windows Azure est actuellement commercialisé dans 41 pays. Il est possible d'accéder à Windows Azure avec un paiement à l'usage, sans engagement, ou bien des forfaits comme décrit à l'adresse http://www.microsoft.com/france/windows-azure/Offres.aspx.

Windows Azure permet d'héberger des applications Web, des Services Web, et de bénéficier d'une infrastructure de stockage fiable et robuste. L'administration de ces environnements est réduite au minimum, Microsoft s'occupant de la maintenance des systèmes d'exploitation, de la redondance des applications et données, et de la disponibilité des plateformes.



B. Les rôles Azure

Les services et solutions sont construits avec une combinaison quelconque de Web Roles, Worker Roles ou VM Roles, se répartissant les tâches à exécuter. Ces rôles peuvent écouter des requêtes provenant de l'extérieur, et peuvent aussi échanger des messages entre eux, sur un VLAN interne sécurisé. Les protocoles acceptés sont HTTP/HTTPS ou TCP/IP, et les machines virtuelles exécutant ces rôles peuvent exécuter du code développé avec des outils Microsoft ou non Microsoft: ASP.NET, WCF, autres outils .NET, mais aussi Java, Python, Ruby, Node.Js...

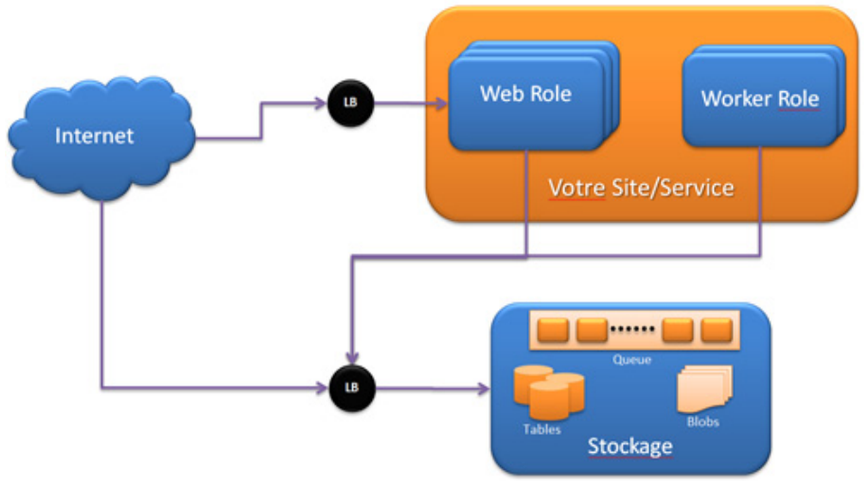
Les différents types de rôles candidats au déploiement d'un environnement de test de charge Visual Studio sont décrits ci-après :

Worker Role

- Machine virtuelle basée sur Windows Server 2008 SP1 ou Windows Server 2008 R2 (configurable)
- Exécute une DLL similaire à un service Windows (méthode Start / Run / Stop à implémenter)
- Peut aussi héberger d'autres applications classiques (JVM, exécutables natifs / .Net, serveurs Tomcat, Apache, SQL Server…) si leur installation peut être automatisée
- Deux modèles de développement classiques :
  - Un modèle réactif : Le VM rôle héberge un service exposé en interne ou en externe (par exemple avec WCF) et répond directement aux demandes
  - Un modèle batch : Le VM rôle consulte continuellement une file de messages contenant les tâches à exécuter (modèle asynchrone)
- Similaires à un "batch" ou un Service Windows

VM Role

- Permet d'exécuter sa propre machine virtuelle Windows Server 2008 R2 dans Windows Azure
- Plus grande flexibilité offerte aux développeurs pour migrer leurs applications dans Windows Azure
- Possibilité d'exécuter des programmes d'installation ne pouvant pas être automatisés

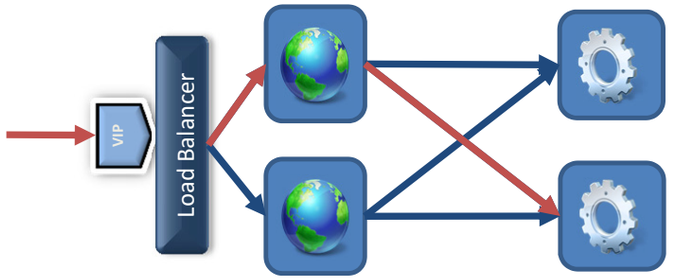


C. Prise de main à distance avec Remote Desktop

Le « Remote Desktop » intégré au portail permet d'accéder aux instances en mode interactif depuis une machine cliente Windows classique (utilisation du client « Remote Desktop » standard). L'accès à distance est supporté pour les trois types de rôle : Web, Worker et VM.

Ce type d'accès est rendu possible par l'activation du plugin remote desktop sur chaque rôle du service (dans son fichier de définition), et la configuration du nom de compte Windows, mot de passe et date d'expiration utilisés pour se connecter. C'est le plugin Remote Desktop qui assure la création du compte Windows correspondant au démarrage du rôle.

Sur un des rôles du service est activé le Remote Forwarder. En effet, chaque requête adressée au service (que ce soit en http ou en TCP/IP) est prise en charge par le Load Balancer Azure qui route les messages aléatoirement sur les instances configurées pour répondre sur le port demandé. Un composant logiciel doit donc intercepter les ordres Remote Desktop pour les rediriger vers la machine réellement ciblée :



« Web Role » avec « Remote Forwarder »

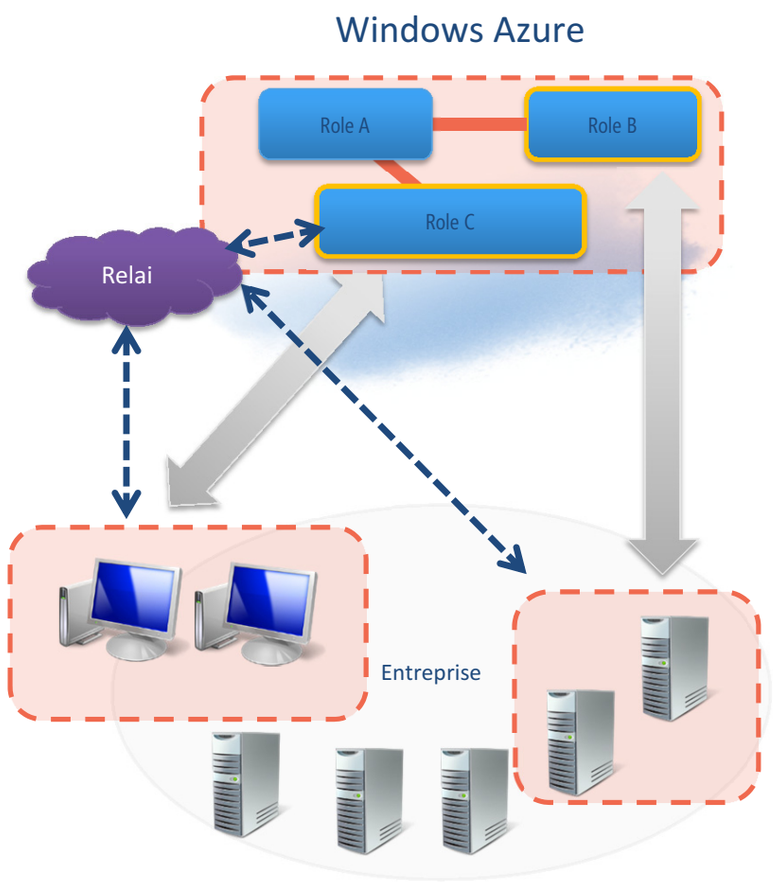
« Worker Role »

D. Création d'un VPN avec Azure Connect

Une politique de sécurité réseau, gérée directement, depuis le portail Windows Azure, permet un contrôle granulaire de la connectivité entre les rôles Windows Azure et les machines externes.

Le protocole IPSec est installé automatiquement. La traversée (« tunneling SSL ») des firewalls ou des NAT s'effectue grâce à l'hébergement d'un processus « relai ».

Les règles réseaux sont appliquées et le trafic sécurisé via IPSec (avec certificats). La résolution de nom DNS est fondée sur les noms des machines (« endpoint »).



2. Infrastructure de l'environnement de tests

A. Choix des composants de la plateforme

Pour héberger l'environnement de test, nous avons envisagé l'utilisation de Worker Roles et/ou des VM Roles.

Les composants à déployer sont :

- Le contrôleur de test
- L'agent de test
- SQL Server Express

Tous ces composants sont installables et configurables en ligne de commande (et peuvent donc être déployés au démarrage d'un Worker Role.

Par contre, certains d'entre eux sont peu adaptés à l'application d'un sysprep d'image Windows, et ne peuvent donc pas être facilement pré-déployés dans un VM Role.

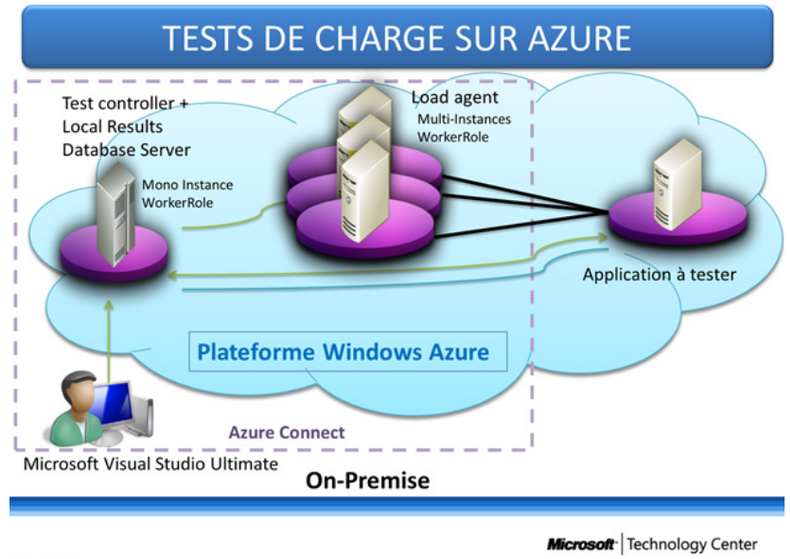
La topologie retenue est donc la suivante :

- 1 rôle de type Worker Role hébergeant le contrôleur et la base de données SQL Express (1 seule instance)
- 1 rôle de type Worker Role hébergeant l'agent (n instances)

En mode Workgroup, le mécanisme d'authentification utilisé par VS Test se fonde sur le principe des « Shadow Accounts » (toutes les machines ont un compte Windows du même nom, avec un mot de passe identique).

C'est au plug-in Remote Desktop qu'incombe la tâche consistant à créer ces comptes identiques. La machine locale exécutant le client Visual Studio doit elle aussi avoir un compte du même nom / mot de passe.

Afin de permettre la communication Netbios entre les instances et le client de test, il est nécessaire de configurer Azure Connect. Ainsi les rôles contrôleurs et agents, ainsi que la machine locale exécutant le client de test Visual Studio soient dans le même réseau VPN.



B. Implémentation de la solution proposée

La configuration des composants à déployer a fait l'objet du développement d'une application permettant de complétement industrialiser le processus. Cette solution est téléchargeable à l'adresse suivante :

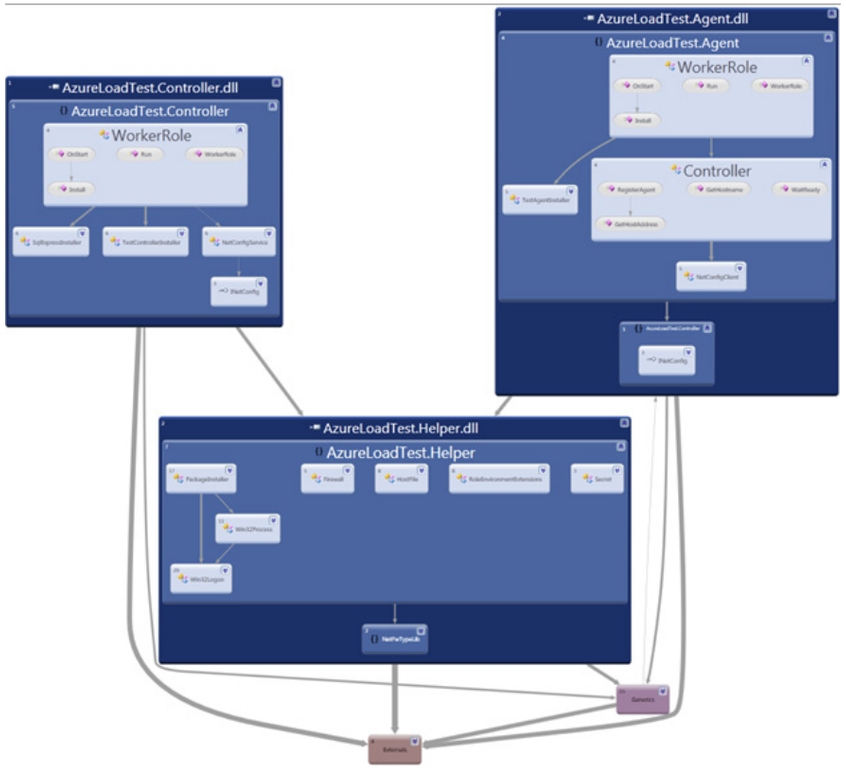
http://stephgou.blob.core.windows.net/azureloadtests/AzureLoadTest-V110902a.zip

Il s'agit d'une solution Visual Studio 2010 SP1 (le SDK Azure n'est pas encore disponible pour Visual Studio 2012). Les composants déployés correspondent également à la version Visual Studio 2010 SP1 des agents et contrôleurs de Load test. Toutefois, le déploiement de la version VS 2012 serait également possible.

Elle est constituée de composants communs (configurateurs de Firewall Windows, librairies de compression/décompression, téléchargement d'installeur depuis le blob storage…) utilisés dans chacun des rôles, ainsi que de composants spécifiques : code d'installation de SQL Serveur (générant un fichier de réponse puis pilotant l'installation), code d'installation du contrôleur de test, code d'installation de l'agent de test, contrôle des services de test, etc.

Un service WCF est également intégré dans cette solution. Il permet au rôle contrôleur de fournir son nom Netbios aux instances du rôle agent (afin que les agents puissent contacter le contrôleur de tests et s'y enregistrer)

Le diagramme suivant présente les différents composants de la solution Visual Studio à déployer sur Windows Azure et leurs dépendances.



La solution Visual Studio regroupe la configuration Azure et les trois projets Visual Studio associés (deux WorkerRoles et une librairie de fonctions utilisées pour l'automatisation des installations).

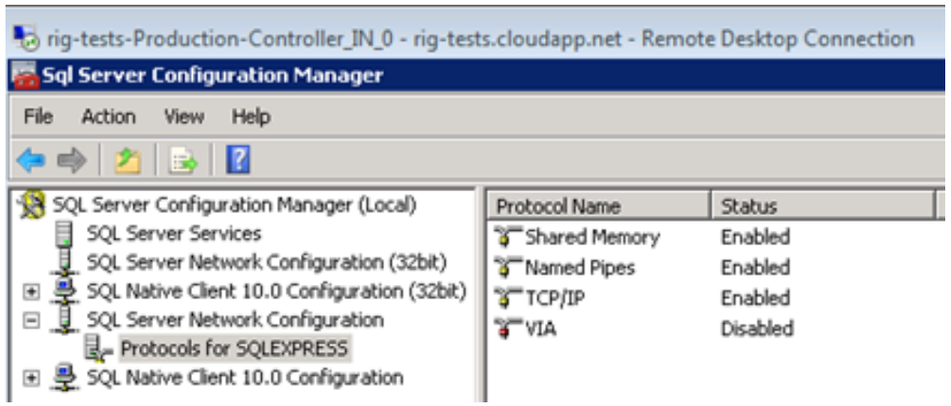
C. Descriptif de la solution déployée

La solution déployée dans un service Azure (« Azure Load Test ») est distribuée sur deux Worker Roles (une instance hébergeant le contrôleur, n instances hébergeant l'agent).

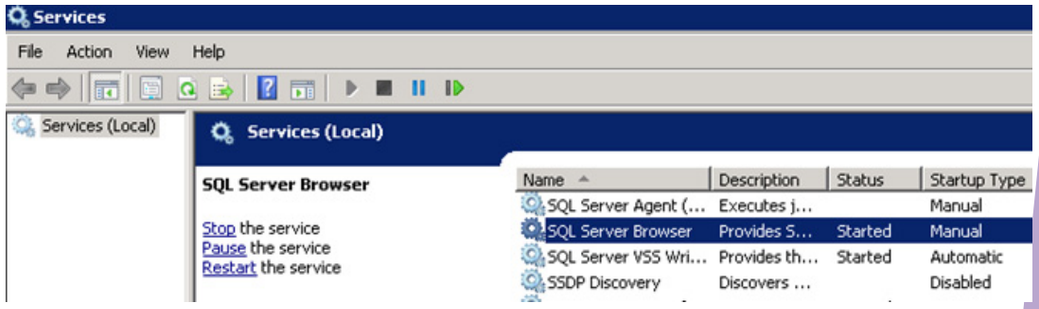
Field: stephgou	Subscription	Active	
Azure Load Tests	Hosted Service	Created	
Certificates			
AzureLoadTest - 4/30/2012 1:44:03 PM	Deployment	Ready	Production
Agent	Role	Ready	Production
Agent_IN_0	Instance	Ready	Production
Agent_IN_1	Instance	Ready	Production
Controller	Role	Ready	Production
Controller_IN_0	Instance	Ready	Production

La base de consolidation est automatiquement créée sur le contrôleur. Elle est accessible à distance depuis Visual Studio.

En effet, les protocoles TCP/IP et Name Pipes sont automatiquement activés sur le contrôleur.



Autre prérequis de cette connexion SQL distante, le Service SQL Server Browser est automatiquement démarré sur le contrôleur.



3. Configuration complémentaire

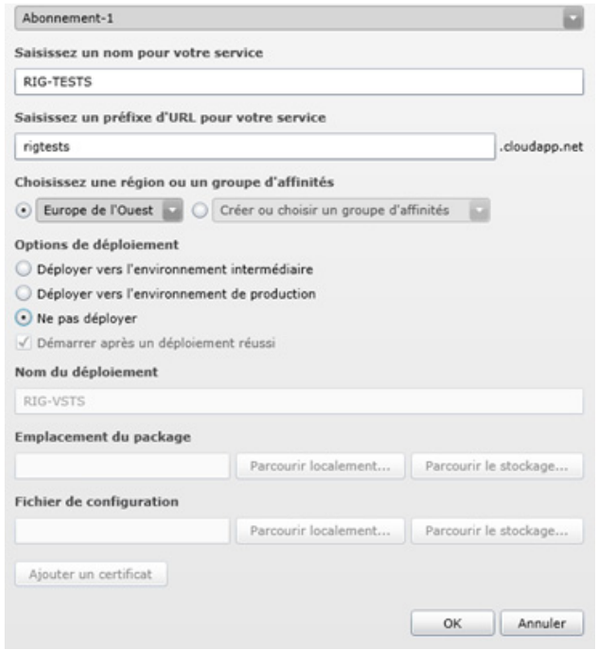
A. Configuration du Poste Visual Studio de publication du package

Sur le poste Visual Studio depuis lequel vont être lancées et suivies les itérations de tests doivent être installés les produits suivants.

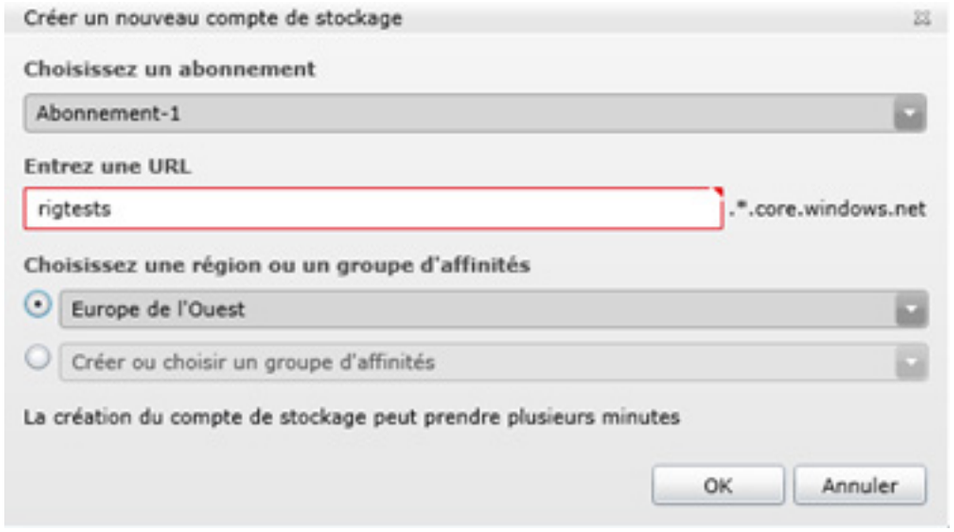
- Installation de Visual Studio Ultimate
- Installation des Windows Azure Tools 1.6 + Azure SDK 1.6 à télécharger depuis http://www.windowsazure.com/fr-fr/develop/net/

B. Configuration des éléments destinés à l'hébergement du « Rig de tests » dans Azure (service, stockage)

La première étape de cette configuration consiste à créer le Service hébergé dans Azure pour la publication de l'environnement de tests également appelé « Rig de tests » Azure. Un service est rattaché à un et un seul centre d'hébergement ou à une zone géographique, qui ne peut être changé une fois le service créé. Ce choix est donc très important au moment de la création.



Une fois ce service créé, il faut alors créer un compte de stockage.



C. Configuration Azure Connect (Première phase)

La configuration du VPN dans lequel fonctionneront les différents rôles du « Rig de tests » et le poste Visual Studio requiert certaines étapes de configuration :

- Obtention du Token depuis le portail Azure : par exemple a1a8f9f5-70f1-40ea-8757-97b036833a45
- Installation d'un point de terminaison local sur la machine Visual Studio
  - https://waconnecttokenpage.cloudapp.net/Default.aspx?token=a1a8f9f5-70f1-40ea-8757-97b036833a45
- Download puis setup du fichier « wacendpointpackage.exe » sur la machine sur laquelle est installé Visual Studio Ultimate. Il est alors nécessaire d'être connecté sous le compte « shadow account ».

D. Configuration des Rôles Azure depuis Visual Studio sur la Solution AzureLoadTest

Les rôles Azure correspondants aux machines du « Rig de tests » sont configurés dans le fichier « ServiceConfiguration.cscfg » dans lequel chaque élément Role référence, via son attribut Name, un rôle déclaré dans le fichier de définition, « Controller » ou « Agent ». Dans ce fichier, chaque « Role » dispose d'un élément « Instances » qui spécifie le nombre d'ins-

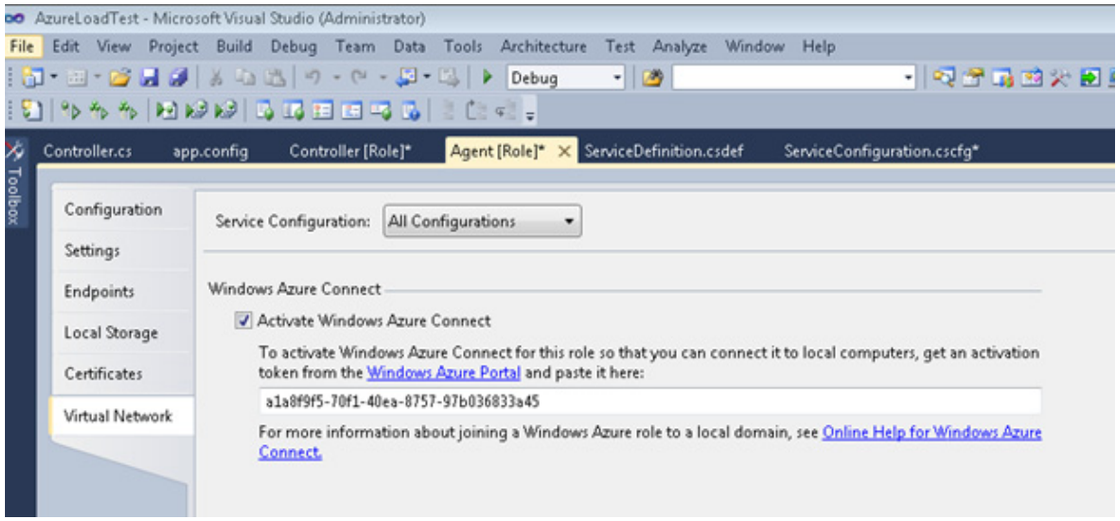
tances parallèles à exécuter pour chaque rôle : 1 pour le « Controller », N pour l' « Agent ».

Il contient également une section « ConfigurationSettings » qui permet de spécifier les valeurs de configuration globales du rôle ainsi que celles déclarées dans le fichier de définition comme, par exemple, les chemins d'accès des fichiers compressés des programmes d'installation des agents, contrôleur et SQL express, ainsi que la clé d'activation pour les tests de charge Visual Studio (« TestController_License »).

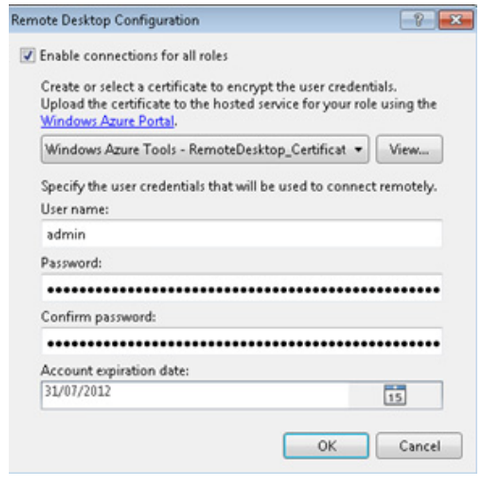
La configuration correspondante est décrite dans le fichier suivant :

```
  <Role name="Controller">
    <Instances count="1" />
    <ConfigurationSettings>
      <Setting name="SqlExpress_Zip" value="http://#your-storage-account-name#.blob.core.windows.
net/bin/sqlexprwt_x64_enu.zip" />
      <Setting name="TestController_Zip" value="http://#your-storage-account-name#.blob.core.
windows.net/bin/testcontroller.zip" />
      <Setting name="TestController_License" value="XXXXX-XXXXX-XXXXX-XXXXX-XXXXX" />
      <Setting name="Microsoft.WindowsAzure.Plugins.Connect.ActivationToken" value="812829bf-
c440-45bc-8b59-3ba4d1ac7262" />
      ...
    </ConfigurationSettings>
    <Certificates>
      <Certificate name="Microsoft.WindowsAzure.Plugins.RemoteAccess.PasswordEncryption" thumbpri
nt="39F3B59151B2F9FE4E6EBB1377D07399AED4940B" thumbprintAlgorithm="sha1" />
    </Certificates>
  </Role>
</ServiceConfiguration>
```

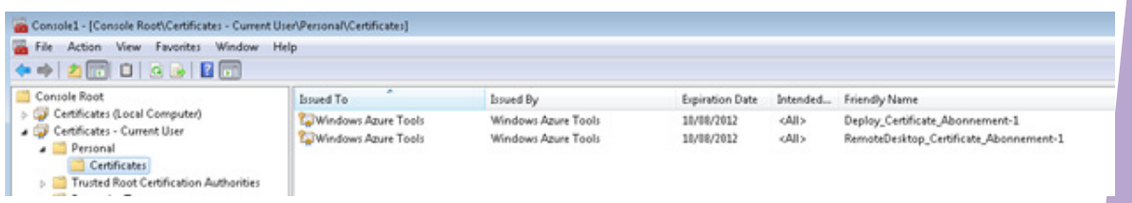
La configuration du VPN Azure Connect nécessite également l'association du token (« a1a8f9f5-70f1-40ea-8757-97b036833a45 ») précédemment copié avec les rôles à déployer. Cette association peut directement être renseignée dans l'assistant proposé par le SDK Azure dans Visual Studio.



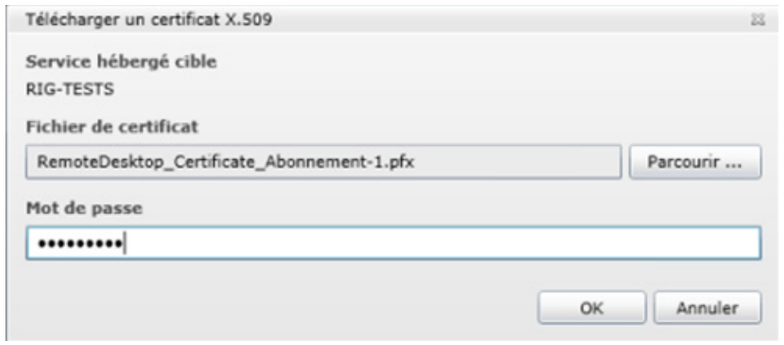
L'activation du Remote Desktop sur les instances hébergées dans Azure suppose la possibilité de disposer d'un certificat pour chiffrer le couple utilisateur/mot de passe utilisé pour se connecter via RDS. Elle requiert donc la création de ce certificat « RemoteDesktop_Certificate_Abonnement-1 » et la spécification des paramètres d'authentification (en utilisant le même compte pour chaque machine, compte identique à celui utilisé pour le poste Visual Studio : « Shadow Account »). Il n'est pas possible d'utiliser le compte « Administrator » dont l'usage est déjà réservé.



Le certificat créé dans Visual Studio est stocké dans le dossier « Personal » du magasin de certificats « Current User ».



Il faut alors l'exporter au format PFX (avec les clés privées et un mot de passe) afin de pouvoir ensuite l'uploader sur le portail Azure.

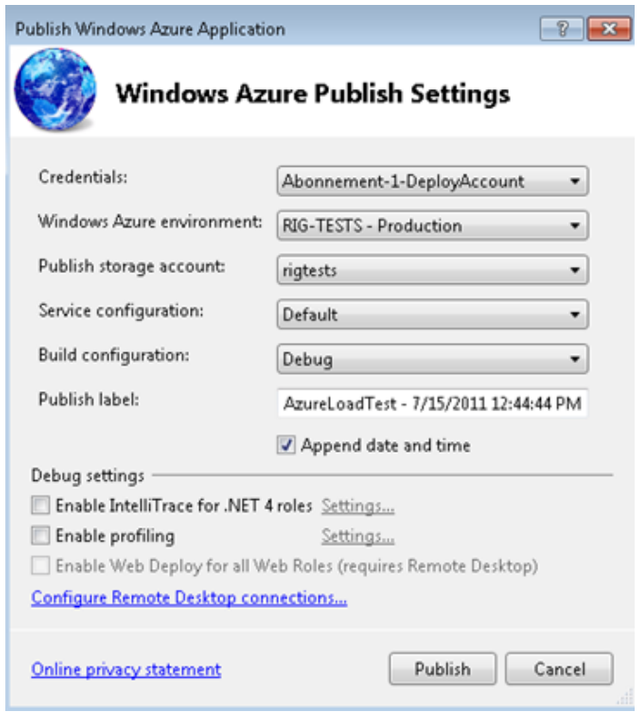


E. Publication depuis Visual Studio

Pour simplifier le déploiement sur Windows Azure, il est possible de déployer le service hébergé directement depuis la solution Visual Studio, sans passer par le portail Azure. Pour assurer l'authentification, Windows Azure se fonde sur l'utilisation d'un certificat.

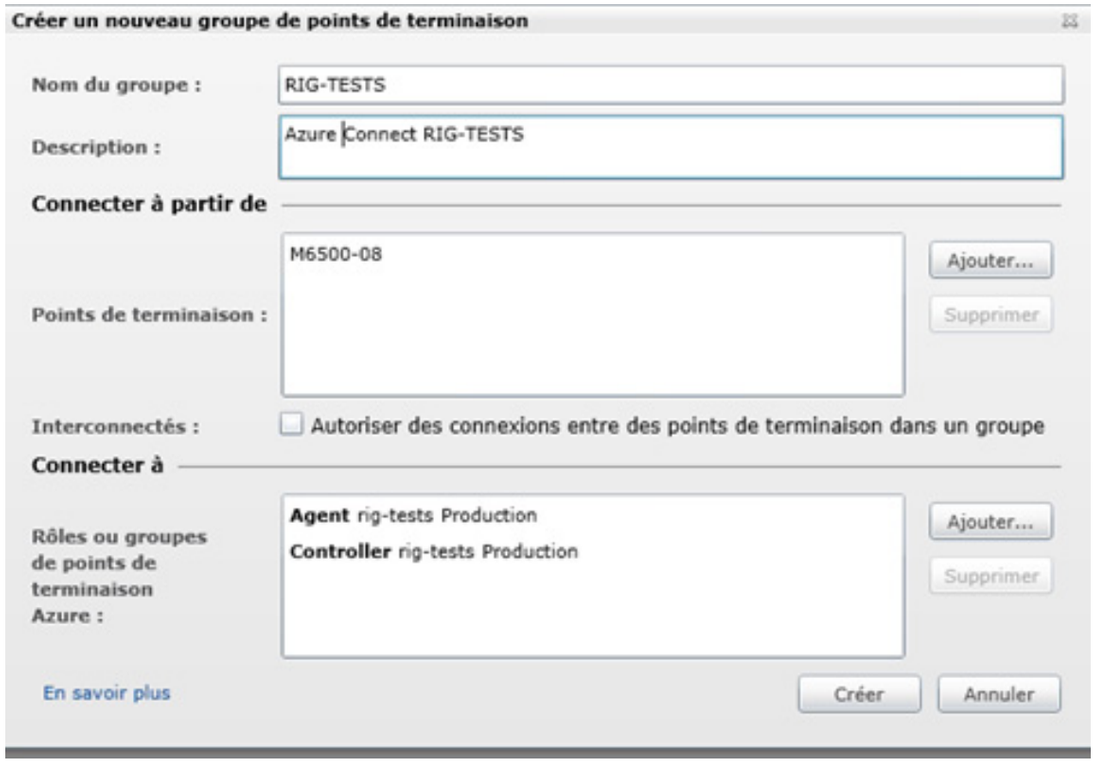
Il faut donc générer le certificat « Deploy_Certificate_Abonnement-1 » depuis Visual Studio, et uploader le fichier « Deploy_Certificate_Abonnement-1.cer » correspondant sur le portail comme certificat valide pour la publication.

Une fois cette configuration effectuée, le « Rig de tests » peut-être directement mis en ligne depuis la fenêtre de publication Visual Studio.



F. Configuration Azure Connect (Deuxième phase)

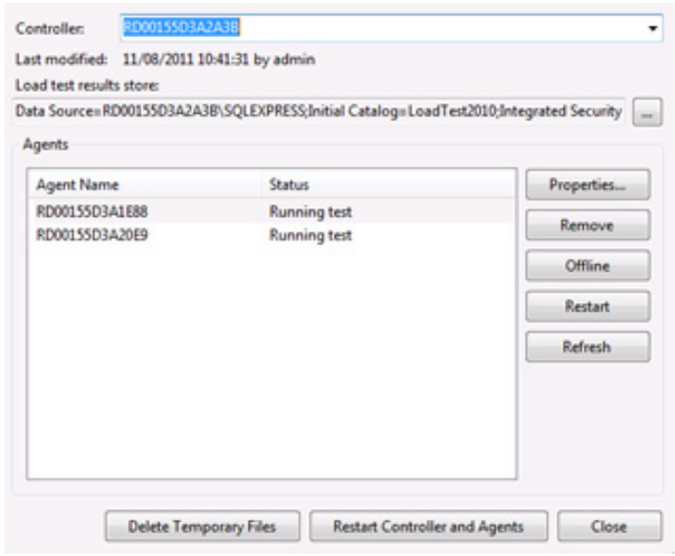
Lorsque les instances des rôles « Agent » et « Contrôleur » en ligne, il faut ajouter les « endpoints » correspondants dans un groupe « Azure Connect » dédié.



4. Paramétrage de l'environnement de tests dans Visual Studio

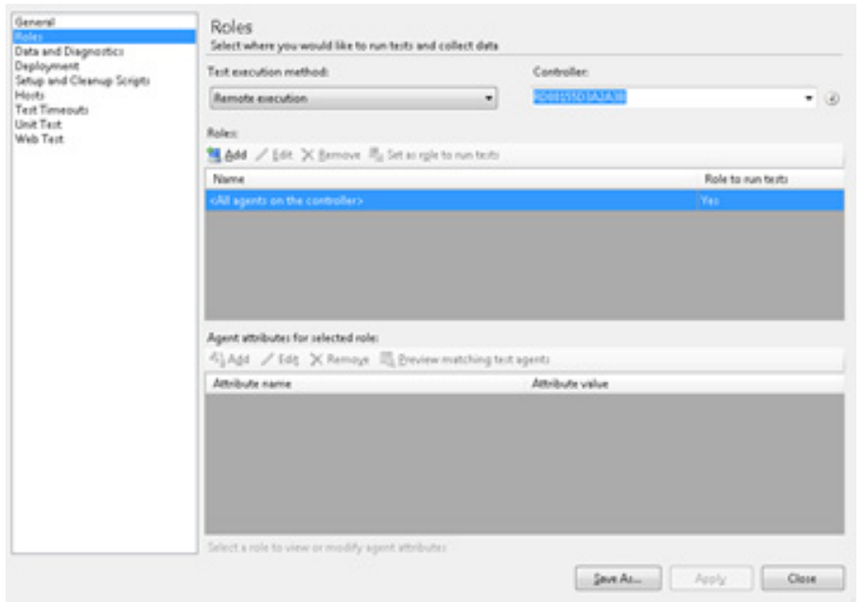
Une fois le « Rig de Tests » en ligne, il ne reste plus qu'à finaliser son paramétrage (configuration du contrôleur et de la base de consolidation des résultats) depuis Visual Studio.

Les tests de charge effectués à mainte reprise au MTC l'ont été depuis un poste Visual Studio 2010 SP1. Au vu des interfaces proposées avec Visual Studio 2012 (correspondant d'ailleurs aux copies d'écrans présentées dans ce document), cette nouvelle version devrait également pouvoir être utilisée pour piloter le « Rig de Test » hébergé dans Azure en suivant la procédure suivante :



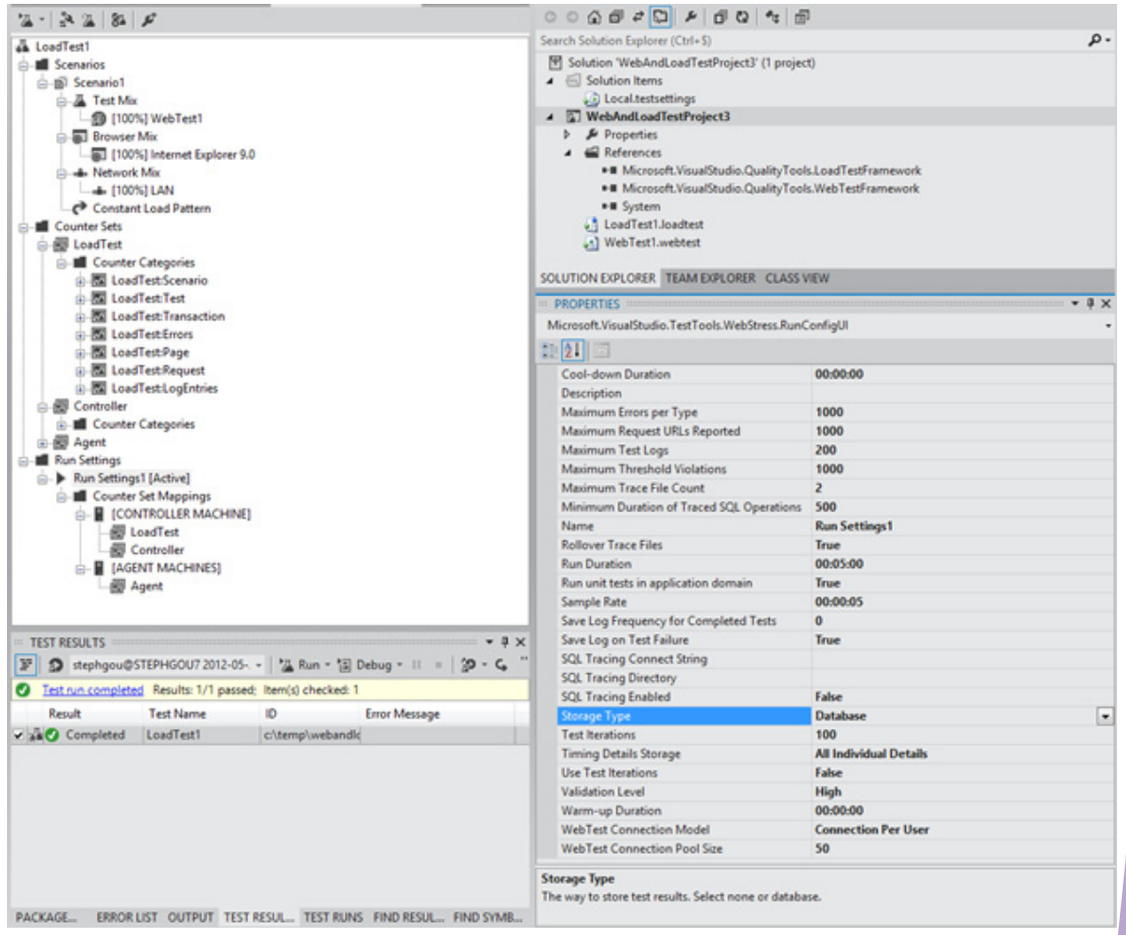
Configuration de contrôleur depuis Visual Studio 2012

Pour compléter cette configuration, il est nécessaire de préciser le mode d'exécution (en « remote » sur le Contrôleur).



Configuration du mode d'exécution depuis Visual Studio 2012

Enfin, pour que les résultats soient stockés en base, le paramètre Storage Type doit être configuré sur DataBase dans la définition des settings du Load Test (option par défaut).

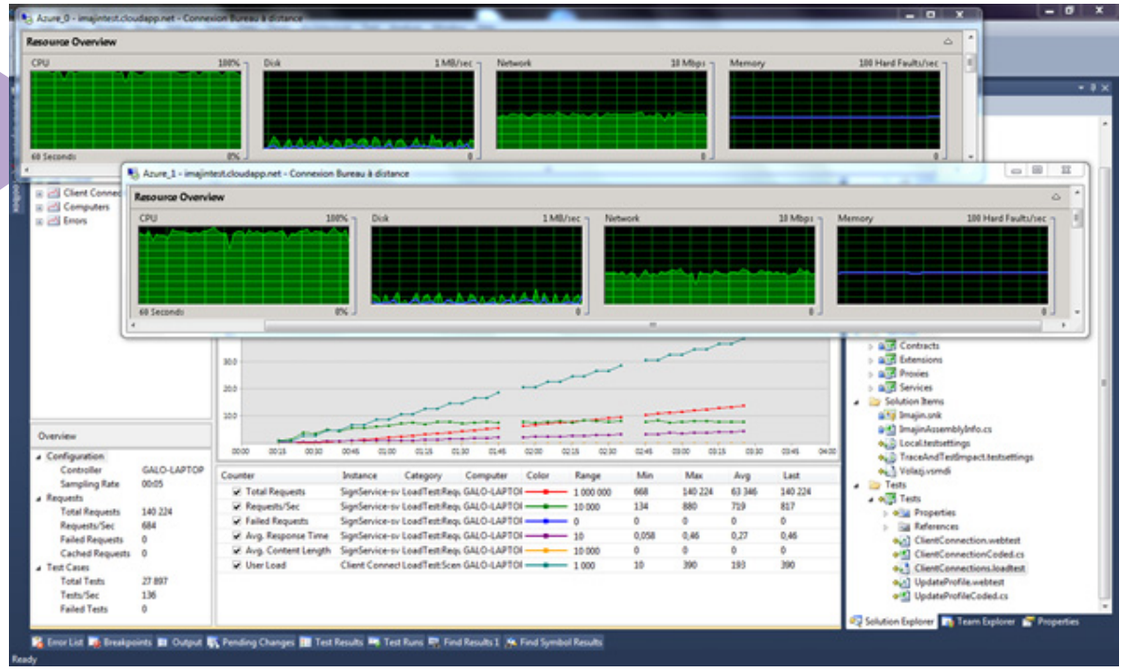


Configuration des settings du Load Test dans Visual Studio 2012

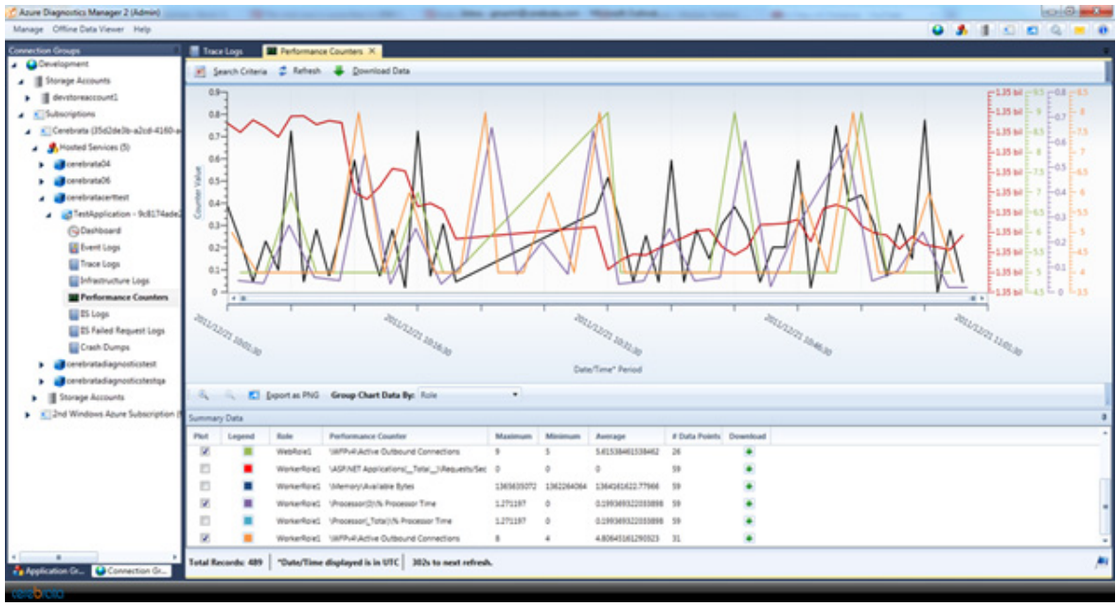
5. Suivi de l'exécution des tests

L'utilisation de la solution Visual Studio telle que décrite dans ce document permet de disposer nativement des compteurs de performances collectés sur les différents composantes du « Rig de tests ».

Par contre, la lecture des compteurs de performances et autres métriques issues de l'observation des machines hébergeant l'application Cloud requiert la mise en place d'une approche complémentaire. Celle-ci peut se limiter à l'utilisation des outils Windows avec une prise en main à distance des machines virtuelles hébergées sur Azure comme le montre la copie d'écran suivante :



Une autre solution que nous recommandons vivement se fonde sur l'utilisation d'outils tiers comme la solution Azure Diagnostics Manager de la société Redgate (Cerebrata : http://www.cerebrata.com/Products/AzureDiagnosticsManager).



Synthèse

Les tests de charge jouent un rôle essentiel dans la validation du comportement d'une application Cloud dans des conditions proches de son utilisation et dans sa capacité à répondre à la sollicitation envisagée. Ils permettent en outre de fournir de précieux éléments pour le dimensionnement initial.

Leur mise en œuvre requiert la mise en place d'une infrastructure dédiée à la simulation de la charge et à la collecte de résultats. La Plateforme Windows Azure permet de mettre en place simplement et rapidement cette infrastructure, en se fondant sur l'utilisation de la solution Visual Studio et sur l'automatisation de sa configuration. Ce livre blanc synthétise notre retour d'expérience sur les tests de charge Azure qui se sont déroulés depuis deux ans au MTC Paris pour de nombreuses applications aujourd'hui en production sur la plateforme Windows Azure.

Désormais, pour réaliser un test de charge, il n'est plus nécessaire de louer voire d'acheter un ensemble complet de machines, puis de procéder à une installation coûteuse en temps. Il suffit d'utiliser la Plateforme Windows Azure et Visual Studio…

A propos d'Infinite Square

Infinite Square est une société de conseil, expertise, réalisation, formation, spécialisée sur les technologies Microsoft. Créée début 2010, Infinite Square est réputée pour l'expertise technique et la qualité relationnelle de ses consultants, architectes, développeurs, formateurs. La société compte aujourd'hui 25 collaborateurs, disposant tous de multiples certifications Microsoft, parmi lesquels 10 Most Valuable Professionals (MVP), constituant ainsi une densité d'expertise et de savoir-faire sans équivalent sur les technologies Microsoft.

Infinite Square est spécialisée sur les technologies de développement d'applications et sur la plateforme applicative Microsoft. Ainsi, nous intervenons dans 4 grands domaines:

1. Le développement d'applications spécifiques :

a. Le framework et les languages .NET

b. Le cloud computing : conception, déploiement et migration d'applications dans Windows Azure

c. Les interfaces utilisateurs, depuis les clients riches WPF, Silverlight, HTML5, WP7, jusqu'aux nouveaux modes d'interaction avec Surface 2, Windows 8 Métro, et Kinect

2. Les solutions collaboratives et de portails avec SharePoint, dans des architectures ''on premises'' ou ''online'' :

a. Sites de publication, internet ou intranet

b. Espaces de collaboration d'entreprise, gestion documentaire, workflows

c. Applications métier et interopérabilité avec des systèmes existants

d. Réseaux sociaux d'entreprise

3. Les solutions décisionnelles / BI avec SQL Server

a. Modélisation dimensionnelle, architecture d'entrepôts de données, bases d'analyse multidimensionnelles

b. Solutions de reporting

c. Modèles d'exploration de données (datamining)

4.La gestion de cycle de vie des applications (ALM) avec Team Foundation Server

a. Mise en place des outils de développement TFS et Visual Studio

b. Méthodes de développement Agile

c. Packaging d'applications

d. Stratégie de tests unitaires, fonctionnels et de performance

Sur ces domaines d'expertise, nous intervenons sur toutes les phases du cycle projet, depuis la MOE et la définition des spécifications techniques, la conception d'architecture, la réalisation, jusqu'à la formation et le transfert de compétences.

Ainsi, Infinite Square travaille avec ses clients selon 3 modes d'engagement :

1. Conseil, expertise, coaching
2. Réalisation, au forfait ou en assistance technique
3. Formation inter-entreprises et intra-entreprise, standard ou personnalisée

Infinite Square est Centre de formation agréé par le Ministère du Travail, et a reçu l'agrément CIR du Ministère de l'Enseignement Supérieur et de la Recherche.

A propos du Microsoft Technology Center

Les **Microsoft Technology Centers (MTC)** sont des environnements uniques d'innovation et de collaboration qui offrent l'accès aux dernières technologies et à l'expertise permettant aux Entreprises, Services Publics et Partenaires de concevoir et de déployer les solutions qui répondent exactement à leurs besoins.

Le Microsoft Technology Center a pour triple mission d'apporter aux solutions envisagées une vision stratégique claire et des capacités de démonstration; de participer à la mise en place rapide d'une solution par un transfert de compétences et l'accompagnement des meilleurs experts; et enfin de minimiser les risques par une validation de concept ou un test de performances des solutions envisagées.

A Paris, le MTC (Microsoft Technology Center) offre une expérience unique d'innovation, et est l'environnement clé, accélérateur de décisions sur les plateformes Microsoft et celles de ses partenaires, matériels et logiciels. Depuis 2004, **architectes du MTC Paris et experts de la plateforme Microsoft bâtissent quotidiennement des solutions et jouent ainsi un rôle essentiel pour relever les défis posés par l'ensemble de ses clients**. Le MTC Paris constitue un environnement de choix pour accélérer les décisions technologiques en proposant un espace de travail de 20 salles collaboratives pour des présentations, démonstrations, ateliers d'architecture et mise en œuvre de prototypes. L'offre ALM est l'une des composantes majeures du catalogue des services proposé par le MTC (http://blogs.msdn.com/b/mtcparis/archive/tags/alm)

Pour permettre ces actions, le datacenter du MTC héberge plus de 50 solutions et acteurs concurrents sur une infrastructure dynamique (Private Cloud) de plus de 300 serveurs et 200 To de stockage mis en œuvre avec nos partenaires Alliance. Avec plusieurs centaines de projets par an depuis le démarrage de l'activité, l'équipe MTC apporte un retour d'expérience unique auprès de chaque nouveau projet.

Situé à Paris, au 39 quai du Président Roosevelt, 92130 Issy-les-Moulineaux ; le MTC Paris est l'un des principaux parmi les 20 centres à travers le monde (http://www.microsoft.com/en-us/mtc/locations/paris.aspx)

Annexe : Références techniques

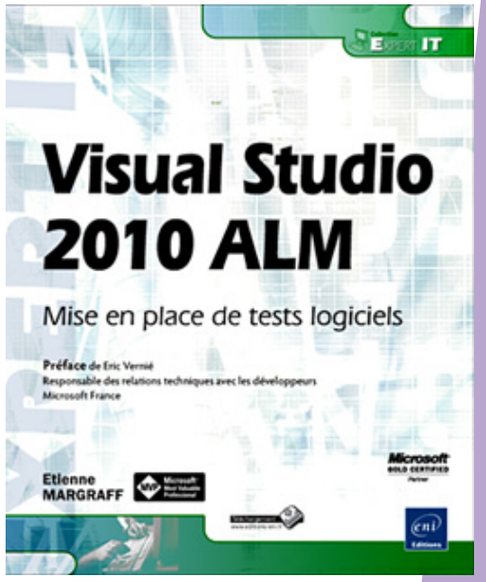
"Using Visual Studio Load Tests in Windows Azure Roles:"
http://msdn.microsoft.com/en-us/library/hh674501(v=VS.103).aspx

Blogs :

- Blog MTC Paris : http://blogs.msdn.com/b/mtcparis/
- Blog Simon FERQUEL : http://www.simonferquel.net/blog/
- Blog Florent SANTIN : http://blogs.developpeur.org/azra
- Blog Etienne MARGRAFF : http://blogs.developpeur.org/etienne
- Blog de Benjamin GUINEBERTIERE : http://blogs.msdn.com/b/benjguin/archive/2011/09/05/load-testing-from-azure-summary-tests-de-charge-depuis-windows-azure-r-233-sum-233.aspx

Editions ENI :

- Windows Azure Platform (Florent SANTIN)
- Visual Studio 2010 ALM (Etienne MARGRAFF)



Infinite Square est une société de conseil, expertise, réalisation, formation, spécialisée sur les technologies Microsoft.

Créée début 2010, Infinite Square est réputée pour l'expertise technique et la qualité relationnelle de ses consultants, architectes, développeurs, formateurs. La société compte aujourd'hui 25 collaborateurs, disposant tous de multiples certifications Microsoft, parmi lesquels 10 Most Valuable Professionals (MVP), constituant ainsi une densité d'expertise et de savoir-faire sans équivalent sur les technologies Microsoft. Infinite Square est spécialisée sur les technologies de développement d'applications et sur la plateforme applicative Microsoft avec quatre domaines d'intervention : le développement d'applications spécifiques, les solutions collaboratives et de portails avec SharePoint, dans des architectures "on premises" ou "online", les solutions décisionnelles / BI avec SQL Server et la gestion de cycle de vie des applications (ALM) avec Team Foundation Server.

INFINITE SQUARE

Microsoft Technology Centers