Architecture pour le Cloud : bonnes pratiques avec les Amazon Web Services (AWS)

neoxia

powered by amazon web services™

Préambule

Ce Livre Blanc est une adaptation non-officielle du document *« Architecting for the Cloud : Best Practices »* rédigé par Jinesh Varia.

La société Amazon© ne peut donc pas être tenue responsable pour son contenu ou les propos exprimés dans le présent de document.

Cette adaptation française est le résultat de la collaboration de la société NEOXIA et la communauté Française AWS (www.aws-ug.fr).

Le Group des utilisateurs AWS France : www.aws-ug.fr

Le club des utilisateurs Amazon Web Services est une association regroupant les utilisateurs ou les professionnels intéressés par les Infrastructures As A Service (IaaS). Il s'agit d'une communauté de passionnés et d'individus fortement convaincus des bénéfices immédiats des services proposés par Amazon.

Loin de tout mirage marketing, le club AWS France a pour vocation à réunir, animer et coordonner les expériences et initiatives autours des IaaS et plus spécifiquement des services d'infrastructure proposés par Amazon depuis 2002.

La Mission du Club des utilisateurs AWS France est :

- Promouvoir l'utilisation de l'offre IaaS d'Amazon
- Accompagner les entreprises et administration dans l'adoption de ces technologies
- Créer et animer une communauté technique Amazon (Architecte, Directeur de Production, Responsable Infrastructure, Développeurs)
- Produire du contenu technique, d'usage et schéma d'adoption des technologies AWS
- Suivre et décrypter l'actualité du SaaS, PaaS et IaaS
- Organiser des évènements

Neoxia : www.neoxia.com

NEOXIA est un cabinet d'architecture et de gouvernance du SI. Depuis, 2000, les Architectes NEOXIA accompagnent les grandes entreprises et administrations dans l'évolution de leur Système d'informations. En plaçant le SI comme un levier réel et important de création de valeur, NEOXIA permet à ses clients une adoption réfléchie, progressive et optimale de l'outil informatique. Fortement convaincu du gain important qu'apportent les solutions SaaS/IaaS, NEOXIA a très tôt développé une offre de service et une expertise sur les Infrastructures AMAZON.

Acteur important au sein de la communauté technique Amazon France, NEOXIA vous accompagnera vers une adoption réfléchie et adaptée du Cloud Computing.

Votre contact :

eric.kdual@neoxia.com
01 58 36 40 60

Remerciements

Nous tenons à remercier tout particulièrement les différents architectes Amazon Web Services suivants :

- Nicolas Dasriaux (nicolas.dasriaux@neoxia.com) pour la traduction du document original
- Majid Bouazza (majid.bouazza@neoxia.com) pour sa relecture technique

Par ailleurs, nous remercions la société Amazon pour la richesse de son contenu technique disponible et tout particulièrement Jinesh Varia pour la qualité de son document initial.

Introduction

Depuis des années déjà, les architectes ont mis à jour et implémenté diverses approches, afin de construire des applications présentant un haut niveau de scalabilité. À l'ère du téra, ces pratiques sont d'autant plus pertinentes que les volumes de données ne cessent de croître, que les niveaux d'utilisation deviennent de plus en plus difficiles à prévoir, et que les exigences de temps de réponse sont de plus en plus pressantes. Face à constat, ce document reprend diverses approches et concepts d'ores et déjà connus, et se pose la question de leur évolution dans le contexte du cloud computing. Il présente également des concepts nouveaux tels que l'élasticité, qui sont apparus en réponse à la nature extrêmement dynamique du cloud.

Ce document se destine aux architectes cloud qui désirent acquérir un bagage dans l'objectif de transférer une application d'entreprise d'un environnement physique statique à un environnement virtualisé de type cloud. L'accent est mis principalement sur les principes et les bonnes pratiques permettant de construire des applications nouvelles fondées sur le cloud, ou de migrer des applications existantes vers le cloud.

Contexte

Un architecte cloud doit en premier lieu comprendre les apports du cloud computing. Dans cette partie, vous découvrirez quelques uns des apports économiques du cloud. Vous prendrez également connaissance des divers services AWS disponibles à ce jour.

Apports économiques du cloud computing

Construire des applications pour le cloud présente des avantages certains sur le plan économique. Voici quelques uns de ces avantages.

Suppression quasi-totale des investissements amont d'infrastructure : Construire des systèmes de grande capacité nécessite des investissements très importants en surface de locaux, en sécurité physique, en hardware (racks, serveurs; routeurs, alimentation de secours), en fonctionnement de hardware (coûts énergétiques et climatisation), et en personnel. De part leur ampleur, ces coût doivent être anticipés en amont et mettent en œuvre un processus décisionnel long et complexe, et ceci avant même que le projet ne puisse démarrer. Avec le cloud, de tels frais fixes ou de démarrage n'existent plus.

Infrastructure à la demande : Auparavant, si votre application attirait les utilisateurs en nombre et que le l'application ou l'infrastructure n'était pas capable de faire face, vous vous retrouviez victime de votre propre succès. Ou alors, si vous aviez investi lourdement et que les utilisateurs ne venaient pas aussi nombreux qu'espéré, vous étiez cette fois victime de votre échec. Déployées sur le cloud, les applications bénéficient d'un dimensionnement à la demande de l'infrastructure. Il n'est plus nécessaire de recourir à l'acquisition anticipée de capacité d'infrastructure. Vous augmentez la capacité au fur et à mesure de vos besoins, et payez uniquement ce que vous utilisez vraiment. La flexibilité s'en trouve augmentée, et les risques et les coûts d'exploitation, réduits.

Utilisation plus efficace des ressources : Les administrateurs système sont habituellement préoccupés par l'acquisition de hardware supplémentaire (quand la capacité est insuffisante) ou par le désir de mieux utiliser la capacité disponible (en cas d'excès de capacité ou de capacité inexploitée). Avec le cloud, ils peuvent utiliser les ressources plus efficacement, en laissant l'application demander et restituer elle-même les ressources selon ses besoins.

Coûts modulés selon l'usage : Avec une tarification au service, vous êtes facturé uniquement pour l'infrastructure que vous utilisez. Vous ne payez pas pour de l'infrastructure qui vous est affectée, mais que vous n'utilisez pas. Ceci fait apparaître une nouvelle perspective de réduction des coûts. Ainsi, quand vous

déployez un correctif d'optimisation des performances, vous constatez une réduction quasi-immédiate de vos coûts (parfois dès la fin du premier mois). Par exemple, dans le cas où l'introduction d'un cache permet de réduire de 70 % les requêtes de données, les économies réalisées démarrent immédiatement et continuent de s'accumuler avec le temps ; la récompense ne tarde pas à tomber dès la facture suivante. De plus, si vous construisez une plateforme fondée sur le cloud, vous pouvez propager le mode de tarification modulé par l'usage à vos propres clients.

La réduction des délais de livraison : La parallélisation est l'un des moyens les plus efficaces pour accélérer la vitesse de traitement. Si un traitement lourd, mais parallélisable, dure 500 heures sur une seule machine, avec les *architectures cloud*, il est possible de répartir ce traitement sur 500 instances, réduisant ainsi son exécution à 1 heure. Disposer d'une telle infrastructure 'élastique' permet à l'application de mettre à profit la parallélisation, et de réduire pour le métier le délai de mise à disposition du résultat.

Apports techniques du *cloud computing*

Parmi les apports techniques du *cloud*, on trouve :

Automatisation – 'l'infrastructure scriptable' : Vous pouvez mettre en place des processus reproductibles de construction (*build*) et de déploiement applicatif, et ceci en pilotant l'infrastructure *via* une interface de programmation.

Adaptation automatique de la capacité (auto-scaling) : Vous pouvez adapter la capacité en fonction du besoin de vos applications, que ce soit pour la réduire ou pour l'augmenter, et ceci sans intervention humaine. L'*auto-scaling* facilite l'automatisation, et amène d'avantage d'efficacité.

Adaptation proactive de la capacité (proactive scaling) : Vous pouvez également anticiper, en planifiant à l'avance la capacité en fonction de la fréquentation attendue.

Cycle de développement plus efficace : Les environnements de production peuvent être aisément dupliqués pour fournir les environnements de développement, de test, et de pré-production. Les environnements pré-production peuvent aisément passer au stade de la production.

Amélioration de la testabilité : Pour le test, vous n'êtes jamais à court de hardware. Vous pouvez introduire les tests et les automatiser à tous les stades du processus de développement. Pour une campagne de test, vous pouvez quasi-instantanément dériver un environnement de test préconfiguré, et le supprimer en fin de campagne.

Plans de reprise et de continuité d'activité facilités : Le *cloud* permet une approche plus efficace du plan de reprise d'activité et du plan de continuité d'activité, en permettant de disposer des serveurs et du stockage de données à moindre coût. Vous pouvez répartir vos infrastructures sur de multiples sites géographiques. Vous pouvez dupliquer un environnement complet sur un autre site en quelques minutes.

Utilisation du cloud comme 'trop-plein' : Quelques clics et une politique de *load balancing* adaptée permettent de déverser le trop-plein de fréquentation d'une application vers le *cloud*.

Le *cloud* d'Amazon Web Services

Le *cloud* d'Amazon Web Services (AWS) fournit une infrastructure extrêmement fiable et scalable, permettant de déployer des applications à l'image des sites Web et applications Web les plus fréquentés. Le tout avec des coûts de support et d'administration extrêmement réduits, et d'avantage de flexibilité que vous ne pouvez en attendre de votre propre infrastructure, qu'elle soit sur site, ou dans un datacenter.

AWS offre toute une palette de services d'infrastructure disponibles dès aujourd'hui. Le diagramme ci-dessous présente la terminologie d'AWS, et met en évidence les interactions entre une application et les différents services AWS, ainsi que les interactions entres les services eux-mêmes.

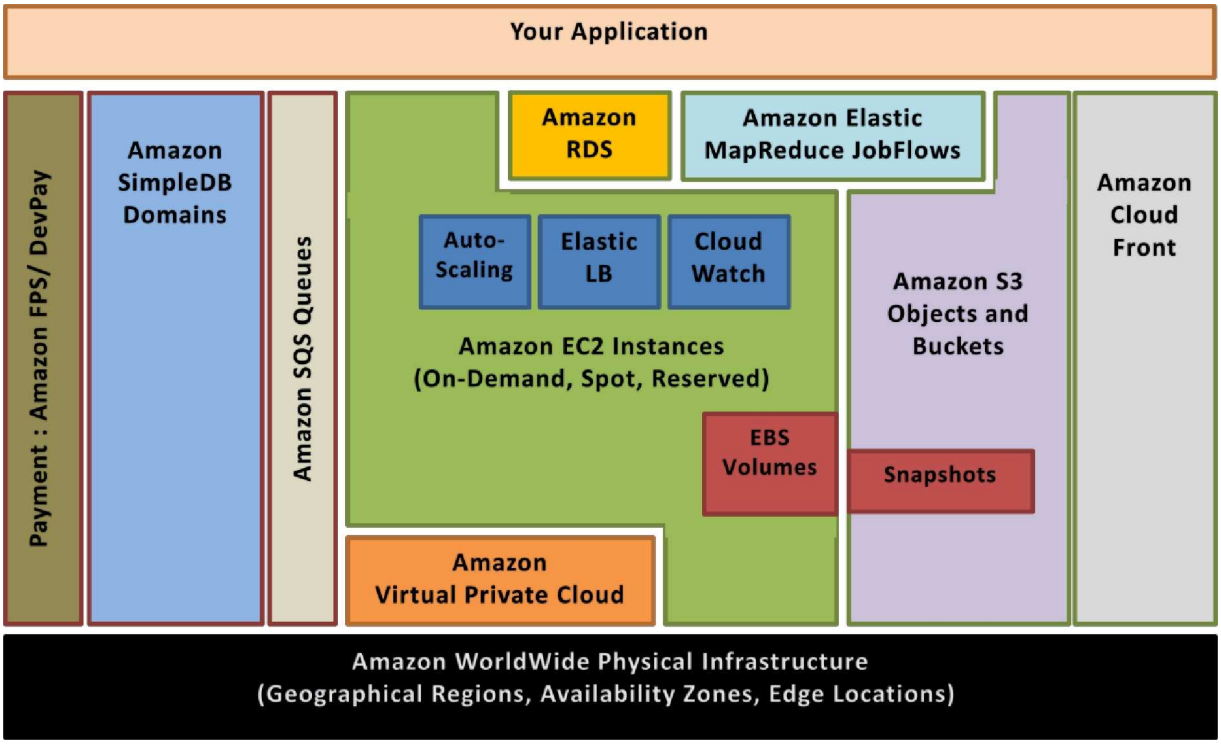

Figure 1: Amazon Web Services

Amazon Elastic Compute Cloud (Amazon EC2)[1] est un service Web qui met à disposition sur le *cloud* de la capacité de traitement ajustable. Une *Amazon Machine Image* (AMI) est le regroupement, au sein d'une même unité, d'un système d'exploitation, des logiciels applicatifs, et des paramètres de configuration associés. Vous pouvez déployer une AMI sous forme de multiples instances virtualisées. Un service Web simple permet d'allouer et de supprimer les instances virtualisées, permettant ainsi d'ajuster rapidement la capacité de traitement en fonction des besoins. Vous pouvez ainsi acheter des instances 'à la demande' (*on-demande instances*), avec une tarification à l'heure, des instances réservées (*reserved instance*), avec un paiement initial et une tarification de l'usage plus avantageuse, ou encore des instances *spot* (*spot instances*), pour lesquels vous faites des offres de prix sur le 'marché' de la capacité non-utilisée, et pouvez ainsi réduire encore vos dépenses. Les instances peuvent être démarrées sur une ou plusieurs régions géographiques. Chaque région dispose d'un ensemble de zones de disponibilité (*availability zones*). Les zones de disponibilités sont des emplacements séparés conçus pour être insensibles aux pannes des autres zones de disponibilité, et également pour fournir des échanges réseau peu couteux et à faible latence avec les autres zones de disponibilité de la même région. Les adresses IP élastiques permette de réserver une adresse IP et

[1] Pour en savoir plus sur Amazon EC2, voir http://aws.amazon.com/ec2

de l'affecter à une instance de manière programmatique. Avec Amazon CloudWatch[2], vous pouvez activer le *monitoring* d'une instance Amazon EC2, et suivre l'utilisation des ressources, la performance en production, et les profils de consommation des ressources (y compris avec des métriques telles que la consommation CPU, les lectures et écritures disque, et le trafic réseau). Avec la fonctionnalité d'*auto-scaling*[3], vous pouvez créer des groupes d'*auto-scaling* (*auto-scaling groups*) de façon à ajuster automatiquement la capacité en fonctions de règles basés sur les métriques relevées par Amazon CloudWatch. Vous pouvez également répartir le trafic entrant en créant un *elastic load balancer* grâce au service Elastic Load Balancing. [4]Enfin, les *volumes* Amazon Elastic Block Storage (EBS) [5]fournissent du stockage persistant de type NAS.

Amazon S3 est un système sûr et distribué de stockage de données (*data store*). *Via* un service Web simple, vous pouvez stocker et consulter de très importants volumes de données sous forme d'objets contenus dans des *buckets* (des conteneurs de données), et ceci depuis n'importe quel point du Web en utilisant simplement les verbes HTTP standard. Les copies des objets peuvent être réparties et mises en cache sur 14 emplacements de stockage (des *edge locations*) situés partout dans le monde. Pour ce faire, vous créer une *distribution* à l'aide du service Amazon CloudFront[7], un service Web de mise à disposition de contenus statiques ou diffusés en *streaming*. Amazon Simple DB[8] est un service Web simple qui fournit les fonctionnalités essentielles d'une base de données (consultation élémentaire en temps réel, requêtes simples de données structurées), sans la complexité d'exploitation habituelle. Vous pouvez organiser les données en domaines et pouvez exécuter des requêtes portant l'intégralité des données d'un domaine. Les domaines sont des ensembles d'*items*, décrits chacun sous forme de paires attribut / valeur. Amazon Relational Database Service[9] (Amazon RDS) permet de mettre en place une base de données sur le *cloud*, de l'exploiter et d'en ajuster la capacité, le tout très simplement. Vous pouvez démarrer une instance de base de données et bénéficier de l'intégralité des fonctionnalités d'une base de données MySQL, le tout sans avoir a vous soucier des tâches d'administration telles que les sauvegardes, et l'application des patchs.[6]

Amazon Simple Queue Service (Amazon SQS) est un service hébergé, fiable et hautement *scalable* de files de messages distribuées ; il permet de stocker les messages en transit entre les ordinateurs et les composants applicatifs.[7]

Amazon Elastic MapReduce met à disposition le *framework* Hadoop sous une forme hébergée. Son exécution repose sur l'infrastructure Web d'Amazon Elastic Compute Cloud (Amazon EC2) et d'Amazon Simple Storage Service (Amazon S3). Amazon Elastic MapReduce vous permet de créer vos propres traitements sous formes de *job flows*, composés des étapes de traitement d'un processus *map* / *reduce*[8].

Amazon Virtual Private Cloud (Amazon VPC) permet l'extension du réseau de l'entreprise vers un *cloud* privatif pris en charge par AWS. Amazon VPC utilise le mode tunnel d'IPsec pour instaurer un canal sécurisé entre une passerelle située coté data center et une passerelle situé coté AWS.[9]

AWS propose divers services de règlement et de facturation[13] qui mettent contribution l'infrastructure de paiement d'Amazon.

Tous les services d'infrastructure AWS sont disponibles selon des tarifications au service, sans engagement à long terme, ni contractualisation.

---

[2] Pour en savoir plus sur CloudWatch, voir http://aws.amazon.com/cloudwatch

[3]Pour en savoir plus sur l'*auto-scaling*, voir http://aws.amazon.com/auto-scaling

[5] Pour en savoir plus sur l'Elastic Block Store, voir http://aws.amazon.com/ebs

[6]Pour en savoir plus sur Amazon S3, voir http://aws.amazon.com/s3

[7]Pour en savoir plus sur Amazon SQS, voir http://aws.amazon.com/sqs

[8]Pour en savoir plus sur Amazon Elastic MapReduce, voir http://aws.amazon.com/elasticmapreduce

[9] Pour en savoir plus sur Amazon Virtual Private Cloud, voir http://aws.amazon.com/vpc

Notez qu’utiliser AWS n’implique en aucun cas de renoncer au niveau de flexibilité et de contrôle que vous connaissiez jusqu’alors.

Vous êtes ainsi libre d’utiliser le modèle de programmation, le langage, ou le système d’exploitation (Windows, OpenSolaris, toute variante de Linux) de votre choix.

Vous également libre de choisir les produits AWS qui correspondent le mieux à vos besoins — vous pouvez utiliser seul chaque services, ou utiliser un sous-ensemble de services.

AWS vous permet d’ajuster les ressources (stockage, bande passante and capacité de traitement) ; vous êtes donc libre de consommer autant ou aussi peu de ressources que vous désirez, et vous payez uniquement pour ce que vous utilisez réellement.

Concepts du *cloud*

Le *cloud* remet en avant des approches relativement anciennes de la construction d'architectures Internet hautement *scalables ;* il propose aussi de nouvelles approches qui modifient considérablement la façon de construire et de déployer des applications. C'est pourquoi, lors de votre cheminement de l'idée à l'implémentation, vous aurez sans doute le sentiment paradoxal que « tout a changé, mais que rien n'est vraiment différent ». Ainsi, le *cloud* modifie substantiellement de multiples processus, *patterns*, pratiques, et approches tenus pour acquis. Il remet aussi en avant certains principes connus des architectures orientées service, principes qui deviennent plus fondamentaux encore qu'auparavant. Dans cette partie, vous rencontrerez autant des concepts nouveaux issus du *cloud* que des concepts repris des architectures SOA.

Les applications traditionnelles ont été construites dans un état d'esprit qui reflétait les préoccupations économiques et l'état de l'art en matière d'architecture de l'époque. Le *cloud* est porteur de nouvelles idées qu'il est nécessaire de s'approprier. Elles sont développées ci-dessous.

Construire des architectures *scalables*

Il est absolument essentiel de construire une architecture qui soit *scalable*, si l'on veut tirer partie d'une infrastructure *scalable*.

Le *cloud* est conçu pour offrir une scalabilité quasi-infinie. Néanmoins, vous ne récolterez pas les bénéfices d'une telle scalabilité d'infrastructure, si votre architecture n'est pas elle-même *scalable* ; les deux doivent aller de concert. Vous devrez ainsi mettre à jour les composants monolithiques et les goulets d'étranglement dans votre architecture. Vous devrez également identifier les points qui empêchent votre architecture de bénéficier de l'ajustement à la demande de la capacité, et remodeler l'application afin de tirer partie de l'infrastructure *scalable* et du *cloud* en général.

Un service véritablement *scalable* se caractérise par :

- Augmenter les ressources augmente la performance dans la même proportion
- Un service *scalable* est capable de supporter une sollicitation hétérogène dans le temps
- Un service *scalable* est toujours en condition opérationnelle
- Un service scalable reste opérationnel même après une forte sollicitation
- Un service *scalable* devient relativement moins couteux à mesure que la sollicitation augmente (le coût par unité baisse avec le nombre d'unités)

Vous devriez tenir compte de ces idées dans toutes vos applications, et les garder en tête, lorsque que vous concevez votre architecture. Vous obtiendrez alors la scalabilité que vous recherchez, simplement parce que votre architecture et votre infrastructure travailleront de concert.

Qu’est-ce que l’élasticité ?

Le graphique ci-dessous présente les différentes approches qu’un architecte *cloud* peut mettre en œuvre pour améliorer la scalabilité des applications, et leur permettre de faire face à la demande.

Approche scale-up (ou scalabilité verticale) : cette approche consiste à ne pas prêter attention à la scalabilité de l’architecture applicative, et à investir lourdement dans des serveurs plus gros et plus puissants (scalabilité verticale), dans l’objectif de faire face à la demande. Cette approche fonctionne raisonnablement, jusqu’à un certain point, mais peut s’avérer extrêmement dispendieuse (voir ‘colossales dépenses d’investissement’ sur le graphique), ou alors l’application peut se trouver submergée par la demande avant même que la ‘grosse artillerie’ ne soit déployée (voir ‘vos client sont partis voir ailleurs’ sur le graphique).

Approche scale-out (ou scalabilité horizontale) traditionnelle : cette approche consiste à élaborer une architecture possédant des propriétés de scalabilité horizontale, et à investir dans l’infrastructure par incréments successifs. La plupart des entreprises et des applications Web de grande taille adopte cette posture, en distribuant leurs composants applicatifs, en répartissant et en fédérant leurs données, et en recourant aux architectures orientées service. Cette approche est souvent plus efficace que l’approche par scalabilité verticale. Elle nécessite cependant de prévoir régulièrement la demande, et d’augmenter par à-coups la capacité de l’infrastructure pour faire face à la demande anticipée. Il en résulte fréquemment un excès de capacité d’infrastructure (gaspillage financier) et la nécessité constante de surveiller le niveau d’utilisation de l’infrastructure (gaspillage de temps de travail).

Note: les deux approches implique des coûts initiaux au démarrage, et sont intrinsèquement réactives.

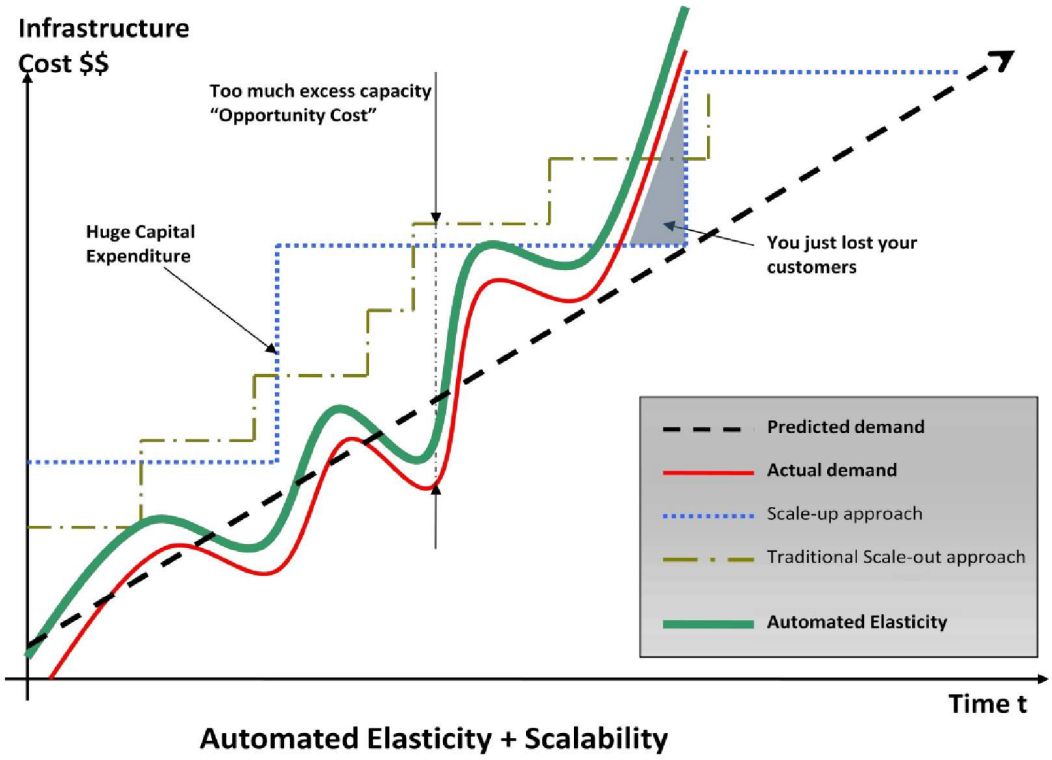

Figure 2: élasticité automatisé

[14] http://en.wikipedia.org/wiki/Slashdot_effect

Avec l'infrastructure traditionnelle, il est bien souvent nécessaire d'anticiper la capacité de traitement pour une période de plusieurs années. Si vous sous-estimez la capacité, vos applications n'auront pas à leur disposition la force de frappe nécessaire pour faire face à une fréquentation inhabituelle, au risque d'entrainer l'insatisfaction de vos clients. Si vous la surestimez, vous gaspillerez probablement un part de vos budgets en ressources inutiles.

L'ajustement à la demande et l'élasticité caractéristiques (élasticité automatique) de l'approche *cloud* permettent à l'infrastructure de s'ajuster au plus près de la demande réelle, qu'elle croisse ou décroisse. Ce faisant, les ressources sont utilisées au mieux, et les coûts sont amoindris.

L'élasticité est l'une des caractéristiques fondamentales du *cloud*. L'élasticité est la capacité d'ajuster les ressources à la hausse ou à la baisse avec le minimum de frottements. Il est important de remarquer que c'est bien de l'élasticité que résulte à terme l'essentiel des apports du *cloud*. Un architecte *cloud* devrait faire sienne la notion d'élasticité, et l'intégrer dans ses architectures applicatives, afin de tirer le meilleur parti du *cloud*.

Traditionnellement, les applications sont construites pour des infrastructures figées, rigides et acquises à l'avance. En entreprise, l'acquisition et l'installation de serveurs sont relativement exceptionnelles, et ne font pas partie des opérations quotidiennes. C'est pourquoi la majorité des architectures logicielles ne tentent pas d'apporter une réponse aux problématiques de déploiement rapide, ou encore de réduction des besoins en *hardware*. Le délai d'acquisition de nouvelles ressources et les investissements en amont sont à ce point élevés, que les architectes logiciels n'ont jamais ressenti le besoin de consacrer du temps et des moyens humains à l'optimisation de l'usage du hardware. Que le hardware sur lequel l'application s'exécute soit sous-utilisé semblait finalement acceptable. Obtenir de nouvelles ressources dans un délai de l'ordre de la minute était clairement impossible, aussi il n'est pas étonnant qu'en matière d'architecture, on ait pu passer à coté de la notion d'élasticité.

Avec l'avènement du cloud, cet état d'esprit doit évoluer. Ainsi, le *cloud computing* assouplit considérablement le processus d'obtention des ressources ; il n'est plus nécessaire de passer commande à l'avance, ou de 'garder en captivité' le hardware *non-utilisé*. Au contraire, les architectes *cloud* peuvent obtenir les ressources en quelques minutes à peine, ou même automatiser le processus d'acquisition des ressources, le tout en tirant partie de la scalabilité et de la réactivité naturelle du *cloud*. Bien entendu, ce schéma reste valable pour la restitution des ressources non-utilisées, ou sous-utilisées.

S'il vous est impossible de faire évoluer l'architecture de votre application, et de mettre en œuvre l'élasticité, vous pourriez vous retrouver en situation de ne pas pouvoir profiter pleinement des apports du *cloud*. En tant qu'architecte *cloud*, vous devriez faire preuve d'imagination et réfléchir aux moyens de mettre en œuvre l'élasticité dans vos applications. Par exemple, le *build* et les tests de régressions (mis en place par l'équipe qualité) démarrent toutes les nuits à 2 heures, et utilisent l'infrastructure pendant 2 h uniquement. L'infrastructure dédiée reste inactive pendant le restant de la journée. Avec une infrastructure élastique, le *build* de nuit peut tourner sur des instances, actives uniquement le temps du build, et facturées pour les 2 h d'usage effectif. De façon similaire, l'application Web interne de suivi des incidents se voit allouée en permanence une capacité maximale (5 serveurs 24/24 7/7), alors qu'elle est essentiellement utilisée durant la journée. La capacité peut être en fait être allouée à la demande pour tenir compte de la fréquentation réelle (5 serveurs de 9 heures à 17 heures, et 2 serveurs de 17 heures de 9 heures).

La conception d'architectures *cloud* intelligentes et élastiques, faisant de sorte que l'infrastructure ne soit active que lorsqu'elle est utilisée, tient parfois de l'art. À terme, l'élasticité devrait devenir une exigence d'architecture et une caractéristique des systèmes. Quelques questions que devriez vous poser : Quelles composants ou couche de mon architecture applicative peuvent être rendues élastiques ? Quelles conséquences aura la mise en place de l'élasticité sur l'architecture système ?

La partie suivante présentera des procédés spécifiques permettant de mettre en place l'élasticité dans vos

applications. Pour tirer réellement profit du *cloud*, il est important de construire votre architecture dans cet état d’esprit.

Nulle raison de redouter les limitations

Quand vous déciderez de migrer vos applications sur le *cloud*, et tenterez de faire correspondre les caractéristiques techniques de vos systèmes avec celle proposées par le *cloud*, vous remarquerez probablement que le *cloud* ne vous fournit pas tout à fait les mêmes caractéristiques de ressource. Qui de dire alors « le cloud ne me propose pas plus de telle quantité X de mémoire vive sur le serveur », ou encore « une instance seule devrait pouvoir fournir à ma base de données un meilleur débit d'entrée / sortie ».

Comprenons nous bien ; le *cloud* fournit des ressources logiques, qui prennent toute leur dimension lorsqu'elles sont utilisées conjointement avec l'ajustement à la demande de la capacité. Vous ne devriez pas craindre d'être limité, quand vous utilisez des ressources sur le *cloud*. En effet, même si vous n'obtiendrez pas toujours, sur le cloud, une configuration à l'identique de votre *hardware*, vous pourrez pallier à cela en obtenant plus de ressources.

Par exemple, si le *cloud* ne vous offre pas suffisamment de mémoire sur un serveur, utilisez une solution de cache distribué tel que *memcached*[15], ou alors partitionnez vos données et répartissez les sur de multiples serveurs. Si les besoins en entrée / sortie de votre base de données ne correspondent pas directement à ceux offert par le *cloud*, vous avez alors plusieurs solution de rechange en fonction du type et du mode d'utilisation de vos données. Si votre application accède en lecture aux données de manière intensive, vous pouvez répartir la charge en lecture sur une batterie d'instances esclaves dont les données sont répliquées. Ou alors, vous pouvez mettre en œuvre un algorithme d'éclatement des données (*sharding*) [10], qui aiguille et place les données à l'endroit approprié. Ou encore, vous pouvez utiliser diverses solutions de *clustering* de base de données.

Après coup, si vous alliez ajustement de la capacité à la demande et flexibilité, vous vous rendrez compte que vous pouvez aller bien au-delà des limites que vous aviez cru percevoir au premier abord, et que vous améliorerez véritablement la scalabilité et la performance globale du système.

Administration virtuelle

L'arrivée du *cloud* a transformé le rôle d'administrateur système en celui d'administrateur de système virtuel. Cela signifie que les tâches quotidiennes effectuées par les administrateurs ont plus d'intérêt encore, ceci d'autant plus que les administrateurs en connaissent maintenant d'avantage sur les applications, et prennent des décisions plus optimales pour le métier. L'administrateur système n'a désormais plus à acquérir les serveurs, à installer les logiciels, et à câbler les équipements réseaux ; ce travail fastidieux est en effet remplacé par quelques clics et manipulations en ligne de commande. Le *cloud* favorise l'automatisation ; son infrastructure est en effet programmable. Les administrateurs sont donc encouragés à prendre de la hauteur technologique et à apprendre à gérer les ressources logiques du *cloud via* des scripts.

De la même manière, le rôle d'administrateur de base de données devient celui d'administrateur de base de données virtuelle. Ce nouveau rôle consiste notamment à gérer les ressources au travers d'une console d'administration Web, à exécuter des scripts afin d'ajouter de la capacité supplémentaire à la base de données, et à automatiser les tâches et processus quotidiens. Le DBA virtuel doit acquérir des compétences nouvelles ayant rapport aux méthodes de déploiement (les images de machines virtuelles), ou à de nouveau modèles (parallélisation des requêtes, redondance géographique et réplication asynchrone [11]). Il doit également repenser son approche d'architecturale de la donnée (éclatement ou *sharding* [9], partitionnement horizontal [13], fédération [14]), et savoir tirer partie des différentes possibilités de stockage offertes par le cloud, en fonction du type des données.

Dans la plupart des entreprises, les développeurs d'application ne coopèrent pas étroitement avec les administrateurs réseau, et ces derniers ignorent les plus souvent tout des applications. Pour cette raison, le

potentiel d'optimisation au niveau du réseau et de l'architecture applicative est bien souvent négligé. Avec le *cloud*, les deux rôles ont, dans une certaine mesure, fusionné. Pour l'architecture des applications nouvelles, les entreprises devrait encourager la dissémination des savoirs entre les deux rôles, et prendre conscience que ces deux rôles sont bien entrain de fusionner.

[15] http://www.danga.com/memcached/

Bonnes pratiques pour le *cloud*

Cette partie présente les bonnes pratiques qui vous aideront à construire des applications pour le *cloud.*

Concevez pour la panne, et il n'y en aura point

En règle générale, soyez pessimiste quand vous concevez une architecture pour le cloud ; considérez que les défaillances sont inéluctables. Autrement dit, concevez, implémentez et déployez dans l'objectif d'un rétablissement automatique après une défaillance.

En particulier, considérez qu'il y aura des pannes de *hardware*. Considérez qu'il y aura des coupures de service. Considérez qu'une catastrophe frappera votre application. Considérez qu'un jour ou l'autre vous serez assailli par un nombre de requête par seconde bien plus élevé que prévu. Considérez que vos composants applicatifs failliront aussi. En étant pessimiste, vous incorporerez le rétablissement après défaillance dans votre conception, ce qui permettra de construire un système globalement plus sûr.

Si vous prenez conscience que tout finit par faillir, vous pourrez en tenir compte dans votre architecture, et incorporer des mécanismes de reprise sur panne qui agiront avant que le dysfonctionnement ne tourne à la catastrophe. Vous aurez alors créé une architecture résistante aux pannes et optimisée pour le *cloud*.

Voici quelques questions que vous devriez vous posez : Que se passe-t-il si un nœud de mon système tombe ? Comment identifier la panne ? Comment remplacer le nœud ? Quels scénarios dois-je envisager ? Quel est le talon d'Achille (*single point of failure*) de mon système ? Si un *load balancer* répartit la charge sur une batterie de serveur, que se passe-t-il si ce *load balancer* crashe ? Si l'architecture comporte un nœud maître et des nœuds esclaves, qu'advient-il si le nœud maître tombe ? Comment la reprise sur panne (*failover*) se déroule-t-elle ? Comment un nouveau nœud esclave est-il instancié et rendu synchrone avec le nœud maître ?

De la même façon que vous concevez pour la panne matérielle, vous devez également concevoir pour la panne logicielle. Voici d'autres questions que vous devriez aussi vous poser : Que se passe-t-il si les services sur lesquels repose mon application voient leur interface modifiée ? Que se passe si un service ne répond pas dans un temps limité (*time-out*) ou retourne une exception ? Que se passe si le cache grossit au point de dépasser la mémoire disponible sur une instance ?

Il est indispensable de mettre en place des mécanismes de prise en charge des pannes. Par exemple, les approches suivantes peuvent être d'un grand secours en cas de dysfonctionnement.

1. Mettre en place un plan de sauvegarde et de restauration de vos données, et l'automatiser.

2. Construire les processus de traitement de façon à ce qu'ils se rétablissent au redémarrage du système

3. Permettre au système de se remettre en état synchrone en rechargeant des messages à partir de files.

4. Conserver des images virtuelles préconfigurées et pré optimisées afin que (2) et (3) soient effectifs au démarrage et au redémarrage du système.

5. Éviter les sessions en mémoire et les contextes utilisateur avec état (*stateful*), les loger dans la base de données

Les bonnes architectures *cloud* doivent être insensibles aux redémarrages du système et aux relances.

Dans le cas de GrepTheWeb (cas développé dans le document d'architecture *cloud* [6]), la mise en œuvre d'Amazon SQS et d'Amazon SimpleDB permet à l'architecture du contrôleur d'être globalement très résistante et résiliente aux dysfonctionnements présentés dans cette partie. Par exemple, si l'instance, sur laquelle tourne le *thread* du contrôleur, tombe, l'instance peut être rétablie et revenir dans l'état précédent, comme si rien ne s'était passé. C'est une image Amazon Machine Image préconfigurée qui rend tout ceci possible. Au redémarrage, elle récupère ainsi tous les messages trouvés sur une file Amazon SQS, et lit l'état associé sur un domaine Amazon SimpleDB.

Si vous concevez vos applications en considérant que le hardware sous-jacent finira par tomber panne, vous serez prêt quand ce moment arrivera finalement.

Ce principe de conception vous aidera à élaborer des applications ayant une meilleure exploitabilité, comme souligné par le document d'Hamilton [11]. Si, en plus de cela, vous pouvez également mesurer et répartir la charge dynamiquement, vous devriez être capable de faire face aux variations de performance (au niveau du réseau et des disques) caractéristiques du *cloud* et de sa nature 'multi-locataires' (*multi-tenant*).

Voici quelques approches possibles pour mettre en place cette bonne pratique avec AWS :

Implémenter un *failover* (reprise sur panne) transparent en utilisant des Elastic IP : Une adresse Elastic IP est une adresse IP statique qui est dynamiquement réaffectable vers un autre serveur. Vous pouvez ainsi réaffecter les adresses IP, et effectuer une reprise sur panne en redirigeant le trafic vers d'autres serveurs. Ceci est particulièrement efficace lorsque vous effectuez une mise à jour d'une ancienne version vers une nouvelle version, ou encore, en cas de panne hardware.

1. Utiliser des zones de disponibilités (*availability zones*) multiples : Les zones de disponibilité sont des sortes de *datacenter* logiques. En déployant votre architecture sur de multiples zones de disponibilité, vous vous assurez des niveaux de disponibilité particulièrement élevés.

2. Tenir à jour une image Amazon Machine Image de façon à pouvoir rétablir et dupliquer très simplement les environnements dans une zone de disponibilité différente. Pour les bases de données, entretenir des nœuds esclaves dans des zones de disponibilité différentes, et mettre en place une réplication continue des données.

3. Utiliser Amazon CloudWatch (ou tout autre outils *open source* de *monitoring* en temps réel) pour surveiller les instances, et mettre en œuvre les actions nécessaires en cas de panne *hardware* ou de dégradation de performance. Configurer un groupe d'*auto-scaling* pour conserver un nombre fixe d'instances, de façon à ce qu'une instance 'mal-en-point' soit remplacée par une nouvelle instance Amazon EC2.

4. Stocker les données dans Amazon EBS (*elastic block storage*), en dehors des instances elle-même, et mettre en place des tâches planifiées de façon à sauvegarder vos données sous forme de *snapshots* différentiels dans Amazon S3.

5. Utiliser Amazon RDS (*relational database service*), et fixer le délai de conservation des sauvegardes, de manière à ce qu'Amazon RDS effectue automatiquement les sauvegardes.

Découplez les composants

Le cloud met l'accent sur l'un des principes de conception du SOA : plus faible est le couplage entre les composants du système, mieux il est capable de s'adapter aux augmentations des exigences et des sollicitations.

Le point-clé est de construire des composants qui ne dépendent pas étroitement les uns des autres. Ainsi, si l'un des composants vient à avorter (réponse en erreur), à se figer (absence de réponse) ou à être débordé (réponse tardive), les autres composants sont construits de telle manière qu'ils continuent de fonctionner, comme si aucun problème ne s'était produit. Essentiellement, le couplage faible permet d'isoler les diverses couches et composants les uns des autres, de façon à ce qu'ils interagissent de manière asynchrone, et se voient chacun comme des boites noire. Par exemple, dans le cas d'une architecture d'application Web, vous pouvez isoler le serveur Web de la base de données. Le serveur d'applications ignore tout du serveur Web and réciproquement ; il en résulte un découplage entre ces deux couches, ainsi qu'une absence de dépendance que ce soit vis-à vis du code ou d'un point de vue fonctionnel. Dans le cas d'un architecture de traitement par lots, vous pouvez créez des composants asynchrones indépendants les uns des autres.

Interrogez vous sur les questions suivantes. Quels composants ou fonctionnalités métier pourraient être extraites de l'application monolithique actuelle, et s'exécuter de manière dissociée et autonome ? Le cas échéant, comment puis-je disposer d'instances supplémentaires de ce composant de façon à pouvoir faire face à d'avantage d'utilisateurs et sans déstabiliser mon système actuel ? Quel est l'effort nécessaire afin d'encapsuler le composant et lui permettre d'interagir de manière asynchrone avec les autres composants ?

Dans le contexte du *cloud*, il devient particulièrement crucial de découpler les composants, d'utiliser un mode de communication asynchrone et de mettre à contribution la scalabilité horizontale. Le *cloud* ne permet pas seulement d'améliorer la capacité de traitement en déployant d'avantage d'instances du même composant. Il permet également des approches hybrides nouvelles, où des composants continuent d'être déployés sur site, tandis que d'autres composants tirent parti du cloud, et en particulier de la puissance de calcul et de la bande passante supplémentaire qu'il offre. Vous ainsi simplement déverser le trop-plein de fréquentation vers le *cloud*, en mettant en place une politique de *load balancing* intelligente.

L'utilisation des files de messages est autre moyen de construire un système faiblement couplé. Si une file ou un tampon est utilisé pour relier deux composants, cette file ou ce tampon permet de gérer les accès simultanés, d'assurer la haute disponibilité, et de faire face aux pics de charge.

Ainsi, le système continue de fonctionner globalement, même si certains composants sont momentanément indisponibles. Si l'un des composants *crashe* ou devient temporairement indisponible, le système accumule les messages et s'assure qu'ils sont traités quand le composant redevient disponible.

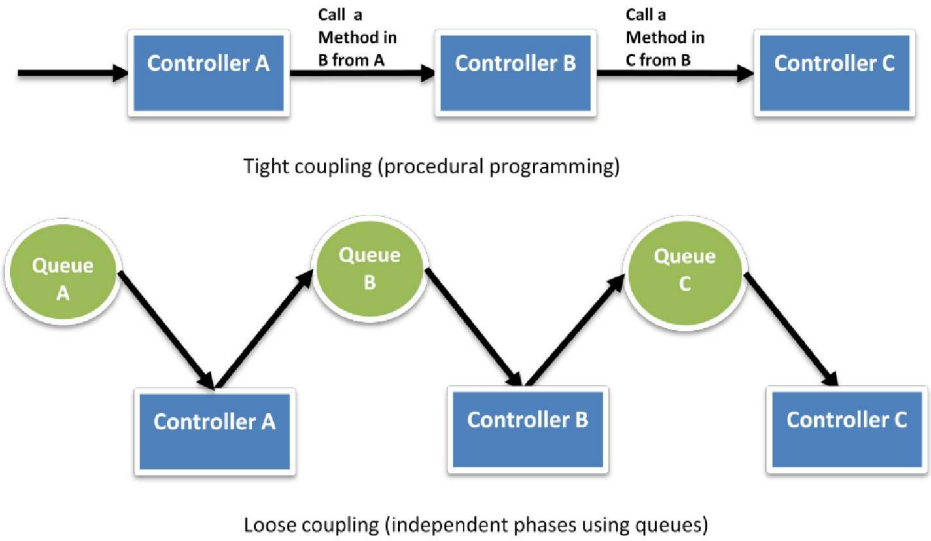

Figure 3: découplage de components avec des files de messages

Vous noterez un usage intensif des files de messages dans l'architecture de GrepTheWeb, illustrée dans le document d'architecture *cloud* [6]. Dans GrepTheWeb, il peut arriver qu'une quantité très importante de requêtes assaille soudainement le serveur (situation de surcharge classique sur Internet), ou que le traitement des expressions régulières nécessite d'avantage de temps que le temps médian (temps de réponse lent de la part d'un composant). De cas de figure, Amazon SQS se charge d'accumuler et de conserver les requêtes dans des files de messages, qui agissent comme un tampon. De cette façon, les retards de réponse n'affectent pas les autres composants.

Voici quelques approches possibles pour mettre en place cette bonne pratique avec AWS :

1. Utiliser Amazon SQS (simple queue service) pour isoler les composants [18]
2. Utiliser Amazon SQS à la manière de tampons entre les composants [18]
3. Concevoir chaque composant de façon à ce qu'il expose une interface de service, qu'il soit responsable de sa propre scalabilité, et qu'il interagissent de manière asynchrone avec les autres composants
4. Joindre les éléments constitutifs du composant au sein d'une image Amazon Machine Image, de façon à pouvoir le déployer plus fréquemment
5. S'assurer autant que possible que les applications sont 'sans état'. Stocker l'état des sessions en dehors du composant (dans Amazon SimpleDB, si cela paraît indiqué)

Mettez en place l'élasticité

Le *cloud* apporte la notion nouvelle d'élasticité à vos applications. L'élasticité peut être mise en œuvre de 3 façons :

1. Adaptation proactive périodique de la capacité (*proactive cyclic scaling*) : ajustement de la capacité de traitement intervenant de façon régulière (quotidienne, hebdomadaire, mensuelle, trimestrielle)

2. Adaptation proactive occasionnelle de la capacité (*proactive event based scaling*) : ajustement occasionnel de la capacité de traitement en prévision d'un afflux important de fréquentation, conséquence d'une opération commerciale planifiée (lancement d'un nouveau produit, campagnes de *marketing*)

3. Ajustement automatique de la capacité à la demande (*auto-scaling based on demand*) : en utilisant un service de *monitoring*, le système peut déclencher les actions nécessaires afin d'ajuster la capacité à la hausse ou à la baisse, en fonction de métriques (taux d'utilisation des serveurs et des entrées / sorties réseau par exemple)

Pour mettre en place l'élasticité, il est d'abord nécessaire d'automatiser le processus de déploiement, et de rationaliser le processus de gestion de configuration et de *build*. L'ensemble pourra alors s'adapter avec un minimum d'intervention humaine.

Le résultat est une amélioration quasi-immédiate des coûts, conséquence d'un meilleur taux d'utilisation des ressources. Plutôt que d'avoir en exploitation des serveurs sous-utilisés, les ressources sont ajustées au plus près de la demande réelle.

Automatisez votre infrastructure

L'un des apports majeurs des environnements *cloud* est la possibilité d'utiliser les API du *cloud* pour automatiser le processus de déploiement. Vous devriez consacrez du temps à l'automatisation du déploiement dès le début du processus de migration, sans en attendre la fin. La création d'un processus de déploiement automatisé et reproductible permettra de réduire le taux d'erreurs et favorisera la mise en place d'un processus de mise à jour efficace et adaptable.

Pour automatiser le processus de déploiement :

- Créer une bibliothèque de 'fiches de cuisine' – un ensemble de petits scripts d'usage fréquent (pour l'installation et la configuration)

- Gérer le processus de configuration et de déploiement en incluant des agents directement à l'intérieur d'une image AMI

- Rendre les instances amorceables

Rendez vos instances amorceables

Au démarrage, une instance doit vous poser la question : « qui suis-je et quelle est mon rôle ? ». Toute instance doit ainsi avoir un rôle ('serveur de base de données', 'serveur d'applications', 'nœud esclave' pour une application Web, par exemple) qu'elle joue dans l'environnement. Le rôle peut être transmis en tant qu'argument au moment de la création d'une instance à partir d'une image AMI, afin d'indiquer les opérations à effectuer après démarrage.

Au démarrage, les instances devraient mettre la main sur les ressources nécessaires (code, scripts,

configuration) en fonction de leur rôle ; elles devraient ensuite se rattacher à un *cluster* afin entrer en fonction. Rendre, de cette façon, vos instances amorçables présente des avantages certains, et permet :

1. de récréer les environnements (développement, pré-production, production) en quelques clics, avec un minimum d'effort,

2. d'avantage de contrôle sur les ressources logiques du *cloud*,

3. de réduire la fréquence des erreurs humaines lors du déploiement,

4. de mettre en place des environnements qui se découvrent et se réparent eux-mêmes, et qui sont plus résistants aux pannes matérielles.

Voici quelques approches possibles pour automatiser l'infrastructure avec AWS :

1. Définir des groupes d'*auto-scaling* pour les divers *clusters* en utilisant la fonctionnalité d'auto-scaling d'Amazon EC2.

2. Activer le *monitoring* des métriques systèmes (processeur, mémoire, entrées / sorties disque et réseau) avec Amazon CloudWatch, et déclencher les actions ou notifications adéquates (par exemple, le lancement dynamique d'un nouvelle AMI par le service d'*auto-scaling*).

3. Stocker et récupérer les informations de configuration dynamiquement : utiliser Amazon SimpleDB pour récupérer les données de configuration au moment du démarrage de l'instance (par exemple, les chaînes de connexions à la base de données). SimpleDB peut aussi être utilisé pour stocker des informations à propos d'une instance telles que l'adresse IP, le nom de machine et le rôle.

4. Construire le processus de *build* de façon à ce qu'il dépose les livrables les plus récents, dans un bucket d'Amazon S3 ; télécharger la version la plus récente de l'application, à partir du bucket, au moment du démarrage du système.

5. Investir dans la construction d'outils de gestion des ressources (scripts automatisés, images préconfigurées), ou utiliser des outils *open source* de gestion de configuration tels que Chef[16], Puppet[17], CFEngine[18] ou Genome[19].

6. Co-livrer, dans une image Amazon Machine Image, à la fois juste ce qu'il faut de système d'exploitation (Just Enough Operating System, JeOS[20]) et juste les logiciels nécessaires ; l'image sera d'autant plus facile à gérer et à maintenir. Transmettre les fichiers de configuration ou les paramètres au moment du démarrage, et récupérer les données utilisateur (user data[21]) et les métadonnées après le démarrage.

7. Réduire les coûts de co-livraison et les temps de lancement en démarrant à partir de volumes[22] Amazon EBS et en rattachant plusieurs volumes Amazon EBS à la même instance. Créer des snapshots des volumes communs, et mutualiser les snapshots entre plusieurs comptes, quand cela parait indiqué.

8. Un composant applicatif ne devrait pas faire de supposition sur l'état de santé et l'emplacement du *hardware* sur lequel il fonde son exécution. Ainsi, rattacher dynamiquement l'adresse IP d'un nouveau nœud au *cluster*. Assurer une reprise automatique sur panne et démarrer un nouveau clone en cas défaillance.

Pensez pour la parallélisation

Le *cloud* rend la parallélisation quasi-naturelle. Que ce soit pour consulter, ou stocker des données, ou encore traiter ces données (ou exécuter une tâche) sur le *cloud*, un architecte cloud devrait avoir constamment à l'esprit la notion de parallélisation, lors qu'il conçoit des architecture *cloud*. Il est recommandé, non seulement de mettre en œuvre la parallélisation, chaque fois que cela est possible, mais aussi de l'automatiser. Le *cloud* rend, en effet, aisée la création d'un processus reproductible.

S'il s'agit d'accéder (consulter ou stocker) à des données, le *cloud* est particulièrement indiqué, de par sa capacité à exécuter des traitements de manière massivement parallèle. Afin d'atteindre des niveaux de performance et des débits de traitement optimaux, vous devrez mettre à contribution la parallélisation des requêtes. L'exécution parallèle des requêtes sur de multiples *thread*s concurrents permet de sauvegarder et de récupérer les données plus rapidement que selon un processus strictement séquentiel. C'est pourquoi, chaque fois que possible, les traitements d'une application *cloud* devraient mettre à contribution le *multi-threading* et être rendus compatibles avec une exécution concurrente (*thread safe*), en évitant de partager quoi que ce soit (approche *share-nothing*).

S'il s'agit d'exécuter des traitements ou des requêtes sur le *cloud*, il d'autant plus crucial de tirer parti de la parallélisation. Une bonne pratique générale, pour une application Web, est de répartir les requêtes entrantes sur une multitude de serveurs Web asynchrones, en utilisant un *load balancer*. Dans le cas d'une application de traitement par lots, vous pouvez mettre en place un nœud maître qui engendre de multiples nœuds esclaves, effectuant chacun en parallèle une partie du traitement (comme avec un *frameworks* de distribution de traitements, tel que Hadoop[24]).

[16] Pour en savoir plus sur Chef, voir http://wiki.opscode.com/display/chef/Home
[17] Pour en savoir plus sur Puppet, voir http://reductivelabs.com/trac/puppet/
[18] Pour en savoir plus sur CFEngine, voir http://www.cfengine.org/
[19] Pour en savoir plus sur Genome, voir http://genome.et.redhat.com/
[20] http://en.wikipedia.org/wiki/Just_enough_operating_system
[21] Pour en savoir plus sur les métadonnées d'instance et les données utilisateur, voir http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/index.html?AESDG-chapter-instancedata.html
[22] Pour en savoir plus sur Amazon EBS, voir http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3121
[23] Pour savoir comment partager un *snapshot*, voir http://aws.amazon.com/ebs/
[24] http://hadoop.apache.org/

Le *cloud* démontre toute son élégance dès lors que vous combinez élasticité et parallélisation. Votre application *cloud* peut ainsi mettre en place un *cluster* d'instances de traitement, simplement en appelant une API, et disponible en l'espace de quelques minutes d'API.

Voici quelques approches possibles pour mettre en place la parallélisation avec AWS :

1. Exécuter parallèlement les requêtes Amazon S3 comme précisé dans le document de bonnes pratiques [2]

2. Exécuter parallèlement les requêtes GET et BATCHPUT d'Amazon SimpleDB [3] [4] [5]

3. Créer un *job flow* avec Amazon Elastic MapReduce pour chacun de vos traitements par lots quotidiens (indexation, analyse de *log*, etc.), ce qui permettra d'exécuter le traitement en parallèle, et de gagner du temps

4. Utiliser le service Elastic Load Balancing et répartir dynamiquement la charge sur une mutiplicité de serveurs d'application Web

Placez les données dynamiques auprès des traitements et les données statiques près de l'utilisateur final

De manière générale, c'est une bonne pratique que de conserver la donnée aussi près que possible des composants de calcul et de traitement de façon réduire la latence. Avec le *cloud*, cette bonne pratique est d'autant plus pertinente et cruciale, que vous serez confrontés aux latences de l'Internet. De surcroit, sur le *cloud*, vous payez pour la bande passante entrante et sortante sur la base du gigaoctet de données transférées, et les coûts peuvent grimper très rapidement.

Si une importante quantité de données à traiter est située hors du *cloud*, il peut être plus économique de transférer d'abord les données vers le cloud, et ensuite d'exécuter le traitement. Par exemple, pour une application de *data warehouse*, il est recommandé de déplacer le jeu de données vers le cloud, avant de d'exécuter des requêtes parallèles sur le jeu de données. Dans le cas d'une application Web, qui lit et écrit des données dans une base de données relationnelle, il est souhaitable de déplacer la base de données, ainsi que le serveur d'applications, vers le *cloud*.

Si les données sont produites au sein du *cloud* même, alors l'application qui utilise les données devrait aussi être déployée sur le *cloud*, de façon à ce qu'elle puisse bénéficier sans coût des transferts de données, et de latences moindres. Par exemple, dans le cas d'une application Web e-commerce, qui produit des *logs* et conserve l'historique des interactions utilisateur (*clickstream*), il est recommandé de faire fonctionner l'analyseur de *log* et les générateurs d'états sur le *cloud*.

À l'inverse, si les données sont statiques et ne changent que rarement (par exemple, des images, des vidéos, des contenus audio, des PDF, des scripts JavaScript, des fichiers CSS), il est souhaitable de s'appuyer sur un service de mise à disposition de contenus. Les contenus statiques sont ainsi conservés en cache sur un *edge location*, situé plus près de l'utilisateur final (le demandeur). La latence d'accès est également réduite. Grâce au cache, un service de mise à disposition de contenus accélère l'accès aux contenus les plus demandés.

Voici quelques approches possibles pour mettre en place cette bonne pratique avec AWS :

1. Transférer les disques de données vers Amazon avec le service d'importation / exportation[25]. Il peut se

révéler plus économique et plus rapide d'envoyer un support physique par courrier[26], plutôt que de passer par l'Internet.

2. Lancer les machines d'un *cluster* au sein de la même zone de disponibilité

3. Créer une distribution pour un *bucket* dans Amazon S3, et laisser Amazon CloudFront mettre en cache le contenu sur 14 *edge locations* répartis dans le monde

[25] Pour en savoir plus sur Amazon Import Export Services, voir http://aws.amazon.com/importexport
[26] http://en.wikipedia.org/wiki/Sneakernet

Bonnes pratiques de sécurité

Dans un environnement multi-locataires, les architectes cloud s'inquiètent fréquemment des problématiques de sécurité. La sécurité doit être mise en place dans toutes les couches de l'architecture d'une application *cloud*. La sécurité physique est classiquement prise en charge par votre fournisseur de service (document [7]), ce qui est un avantage supplémentaire découlant de l'utilisation du *cloud*. La sécurité réseau et la sécurité de niveau applicatif sont, en revanche, de votre responsable, et vous devriez mettre en place les bonnes pratiques en adéquation avec votre métier. Dans cette partie, vous découvrirez des outils et leurs fonctionnalités, ainsi que des éléments de démarche pour vous guider dans la sécurisation de vos applications *cloud* dans un environnement AWS.

Protégez les données transportées

Si vous devez échanger des informations sensibles ou confidentielles entre un navigateur et un serveur Web, mettez en place SSL sur votre instance serveur. Vous aurez alors besoin d'un certificat émis par une autorité de certification externe telle que VeriSign[27] ou Entrust[28]. La clé publique comprise dans le certificat authentifie votre serveur auprès du navigateur, et permet de créer une clé de session utilisée pour crypter les données dans les deux directions.

Créez un cloud privatif virtuel en quelques lignes de commandes (en utilisant Amazon VPC). Vous pouvez alors disposer de vos propres ressources, logiquement isolées à l'intérieur du cloud AWS. Vous pouvez ensuite lier ces ressources à votre propre *datacenter via* des connexions VPN cryptées, grâce au standard IPSec.

Vous pouvez également mettre en place [15] un serveur OpenVPN sur une instance Amazon EC2, installer le client OpenVPN sur tout poste utilisateur.

Protégez les données stockées

Si vous être préoccupé par le fait de conserver des données sensibles ou confidentielles sur le *cloud*, vous devriez crypter les données (chaque fichier individuellement), avant de les télécharger vers le *cloud*. Par exemple, cryptez les données avec un outil de type PGP, *open source*[29] ou commercial[30], avant de les stocker sous forme d'objets Amazon S3, et de les décrypter une fois téléchargées. Cette pratique est recommandée si vous souhaiter élaborer des applications conforme à la loi HIPPA [8], et détenant des données de santé protégées (*Protected Health Information* ou PHI).

Sur Amazon EC2 , le type du cryptage est lié au système d'exploitation. Les instances EC2 sous Windows peuvent utiliser la fonctionnalité intégrée *Encrypting File System* (EFS) [16]. Cette fonctionnalité prend en charge automatiquement le cryptage et le décryptage des fichiers et des dossiers, et rend ces opérations invisibles pour l'utilisateur [19]. Néanmoins, contrairement à ce son nom peut laisser penser, EFS ne crypte pas la totalité du système de fichier, mais seulement les fichiers. Si vous avez besoin de volumes intégralement cryptés, envisagez d'utiliser le produit *open source* TrueCrypt[31], qui s'intègre extrêmement bien avec les volumes EBS formatés en NTFS. Les instances Amazon EC2 sous Linux peuvent monter des volumes EBS en utilisant tout un choix de systèmes de fichiers cryptés (EncFS[32], Loop-AES[33], dm-crypt[34], TrueCrypt[35]). De la même manière, les instances Amazon EC2 sous OpenSolaris peuvent tirer parti de la prise en charge du cryptage par ZFS[36] (*ZFS Encryption Support* [20]).

[27] http://www.verisign.com/ssl/
[28] http://www.entrust.net/ssl-products.htm
[29] http://www.gnupg.org
[30] http://www.pgp.com/
[31] http://www.truecrypt.org/
[32] http://www.arg0.net/encfs
[33] http://loop-aes.sourceforge.net/loop-AES.README

[34] http://www.saout.de/misc/dm-crypt/
[35] http://www.truecrypt.org/
[36] http://www.opensolaris.org/os/community/zfs/

Quelque soit l'approche choisie, le cryptage des fichiers et des volumes dans Amazon EC2 contribue à protéger les fichiers et les *logs*, afin que seuls les utilisateurs et les processus internes au serveur puissent voir les données en clair, et que les utilisateurs et processus extérieurs au serveur voit uniquement les données sous leur forme cryptée.

Indépendamment du système d'exploitation ou de la technologie choisie, le cryptage des données stockées comporte un défi : la gestion des clés utilisées pour crypter les données. Si vous perdez vos clés, vous perdez vos données pour toujours, et si les clés sont compromises, les données sont sans doute en risque. Connaissant cela, prenez soin d'étudier les fonctionnalités de gestion de clés de l'outil que vous choisissez, quelque qu'il soit, et mettez en place une procédure qui minimise le risque de perte de clés.

En plus de protéger vos données des regards indésirables, envisager de les protéger également contre les catastrophes. Faites des *snapshots* réguliers de vos volumes Amazon EBS afin de les conserver durablement et de garantir leur disponibilité. Les *snapshots* sont différentiels ; ils sont stockés dans Amazon S3 (emplacement géographique distinct), et peuvent être restaurés en quelques clics ou lignes de commandes.

Protégez vos clés et certificats

AWS supporte deux formats pour les informations d'authentification : les clés d'accès AWS et les certificats X.509. Une clé d'accès AWS se compose de deux parties : l'identifiant de la clef d'accès et la clé d'accès secrète. Quand vous utilisez l'API REST ou l'API requête, vous devez utiliser votre clé d'accès secrète pour calculer une signature, que vous devez joindre à la requête, afin d'en prouver l'authenticité. Afin d'éviter toute interception et modification malintentionnées, les requêtes devrait être également envoyées au travers de HTTPS.

Lorsqu'une image Amazon Machine Image (AMI) contient des processus qui ont besoin de communiquer avec d'autres services Web d'AWS (par exemple, afin de consulter une file Amazon SQS, ou encore pour accéder à des objets dans Amazon S3), une erreur de conception courante consiste à conserver les informations d'authentification à destination d'AWS, directement à l'intérieur de l'image AMI. En lieu et place, les informations d'authentification devraient être transmises en tant qu'argument au moment du lancement, et cryptés avant d'être transférées sur le réseau.

Si votre clé d'accès secrète est compromise, vous devez en obtenir une nouvelle par rotation[37] vers un nouvel identifiant de clé d'accès. De manière générale, il est recommandé d'intégrer un mécanisme de rotation de clé dans votre architecture applicative, afin de pouvoir le déclencher soit régulièrement soit ponctuellement (quand un employé mécontent quitte l'entreprise, par exemple). De cette façon, les clés éventuellement compromises ne le restent pas éternellement.

Vous pouvez également recourir à des certificats X.509, afin de vous authentifier auprès de certains services AWS. Le fichier de certificat contient une clé publique encodé en DER base64. Un fichier distinct contient la clé privée encodé en PKCS#8 également.

AWS prend en charge l'authentification multi-facteurs[38] (ex : Token RSA), qui permet une protection supplémentaire, lorsque vous accédez aux informations de votre compte sur aws.amazon.com ou bien *via* la console AWS Management Console[39].

Sécurisez votre application

Toute instance Amazon EC2 est protégée par un ou plusieurs groupes de sécurité[40] (*security groups*), des jeux de règles nommés qui décrivent les flux réseaux entrants autorisés sur l'instance. Vous pouvez ainsi

spécifier des ports TCP et UDP, des types et des codes ICMP, et des adresses d'origine. Les groupes de sécurité fournissent une protection élémentaire des instances de type *firewall*. Par exemple, le groupe de sécurité des instances composant une application Web pourrait être paramétré de la manière suivante :

[37] http://aws.amazon.com/about-aws/whats-new/2009/08/31/seamlessly-rotate-your-access-credentials/
[37] http://aws.amazon.com/about-aws/whats-new/2009/08/31/seamlessly-rotate-your-access-credentials/
[38] Pour en savoir plus sur l'authentification multi-facteurs, voir http://aws.amazon.com/mfa/
[39] Pour en savoir plus sur AWS Management Console, voir http://aws.amazon.com/console/
[40] Pour en savoir plus sur les groupes de sécurité, voir http://docs.amazonwebservices.com/AWSEC2/2009-07-15/UserGuide/index.html?using-network-security.html

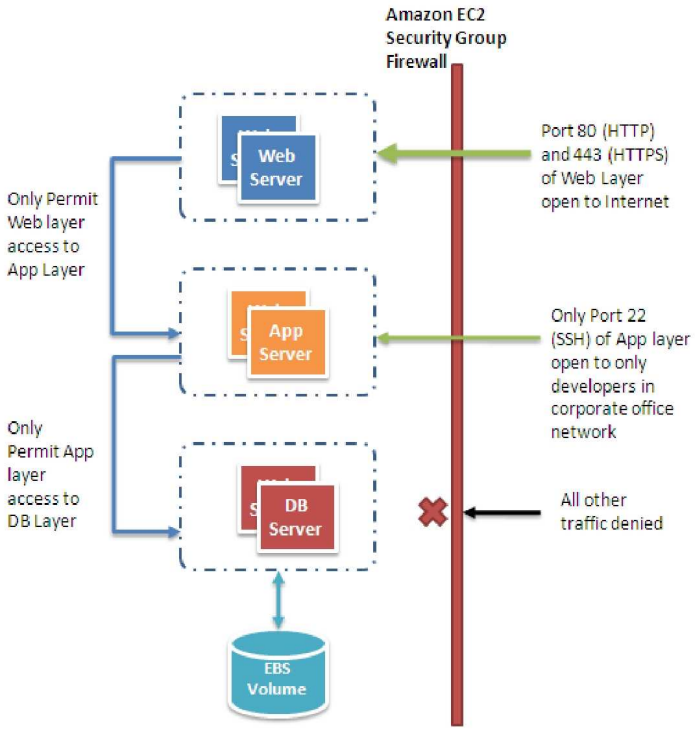

Figure 4: sécurisation d'une application Web avec les groupes de sécurité d'Amazon EC2

Une autre façon de restreindre le trafic entrant est de configurer des firewalls logiciels sur vos instances. Les instances Windows peuvent utiliser le *firewall* intégré. Les instances Linux peut utiliser netfilter[42] et iptables.

Au fil du temps, des *bugs* de sécurité sont découverts dans les composants logiciels, et des *patchs* sont mis à disposition. Vous devriez mettre en œuvre les recommandations suivantes pour optimiser le niveau de sécurité de votre application :

- Télécharger les patchs de sécurité sur le site de l'éditeur et mettre à jour les images AMI
- Redéployer les instances à partir des nouvelles images AMI, et tester les applications pour vérifier la non-régression. S'assurer que l'image AMI la plus récente est déployée pour toutes les instances
- Investir dans des scripts de test afin de pouvoir vérifier périodiquement la sécurité et automatiser le processus. S'assurer que les logiciels tiers sont configurés au plus haut niveau de sécurité
- Ne jamais exécuter un processus sous compte *root* ou Administrateur, sauf si c'est absolument nécessaire

Toutes les pratiques de sécurité standards d'avant le *cloud* sont toujours d'actualité et devraient être mises en place. Il s'agit notamment des bonnes pratiques de code et de l'isolation des données sensibles.

Au final, le *cloud* vous permet de vous abstraire de la sécurité physique, et vous donne le pouvoir de sécuriser votre application grâce à divers outils et fonctionnalités.

[41] http://technet.microsoft.com/en-us/library/cc779199(WS.10).aspx, mars 2003
[42] http://www.netfilter.org/

Perspectives

Le jour n'est plus très loin où les applications cesseront d'être conscientes du *hardware* physique. De la même façon que brancher un four à micro-ondes pour l'alimenter ne nécessite aucunes connaissances sur l'électricité, il devrait être possible de 'brancher' une application sur le *cloud*, afin de lui fournir la puissance dont elle a besoin pour fonctionner, à la manière d'un service. En tant qu'architecte, vous manipulerez des ressources logiques de traitement, de stockage et de réseau en lieu et place de serveurs physiques. Les applications continueront à fonctionner, même si le *hardware* physique sous-jacent tombe en panne, ou s'il est enlevé ou remplacé. Les applications s'ajusteront d'elle-même aux fluctuations de la demande, en déployant automatiquement et quasi-instantanément des ressources supplémentaires, assurant ainsi en continu un taux d'utilisation optimal des ressources. La scalabilité, la sécurité, la haute disponibilité, la tolérance de panne, la testabilité et l'élasticité deviendront des caractéristiques configurables de l'architecture, seront automatisées, et feront partie intégrante de la plateforme.

Mais voilà, nous n'en sommes pas encore tout à fait là. À ce jour, en mettant en œuvre les bonnes pratiques mises en exergue dans ce document, vous pouvez construire des applications sur le *cloud* et leur conférer quelques unes de ces qualités. Les bonnes pratiques relatives aux architectures *cloud computing* vont continuer à évoluer, et en tant que découvreur du *cloud*, nous devrions consacrer notre énergie, non seulement à améliorer le *cloud*, mais aussi à élaborer des outils, des technologies et des processus qui vont permettre aux développeurs et aux architectes de brancher plus facilement leur application sur le *cloud*.

Conclusion

Ce document a présenté des recommandations destinées aux architectes *cloud* et leur permettant de concevoir des applications *cloud* efficaces.

En mettant l'accent sur les notions fondamentales et les bonnes pratiques – comme concevoir pour la panne, découpler les composants applicatifs, comprendre et mettre en place l'élasticité, l'associer avec la parallélisation, et intégrer la sécurité dans tous les aspects de l'architecture applicative –, les architectes *cloud* peuvent mieux appréhender les considérations de conception utiles dans la construction d'applications *cloud* hautement scalables.

Le *cloud* AWS offre un ensemble de services d'infrastructure extrêmement fiables, et facturés à l'usage. Les approches spécifiques soulignées dans ce document aideront à concevoir des applications *cloud* avec les services d'AWS. Dans votre processus de découverte, il est recommandé de manipuler concrètement ces services commerciaux, de tirer parti des travaux tiers disponibles, et de vous appuyer sur les fondations actuelles du *cloud computing* pour les améliorer encore et innover.

Références et bibliographie

1. **Amazon S3 Team, Best Practices for using Amazon S3,**
   http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1904, 2008-11-26

2. **Amazon S3 Team, Amazon S3 Error Best Practices,**
   http://docs.amazonwebservices.com/AmazonS3/latest/index.html?ErrorBestPractices.html, 2006-03-01

3. **Amazon SimpleDB Team, Query 201: Tips and Tricks for Amazon SimpleDB Query,**
   http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1232&categoryID=176, 2008-02-07

4. **Amazon SimpleDB Team, Building for Performance and Reliability with Amazon SimpleDB,**
   http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1394&categoryID=176, 2008-04-11

5. **Amazon SimpleDB Team, Query 101: Building Amazon SimpleDB Queries,**
   http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1231&categoryID=176, 2008-02-07

6. **J. Varia, Cloud Architectures,**
   http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf, 2007-07-01

7. **Amazon Security Team, Overview of Security Processes,**
   http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, 2009-06-01

8. **Amazon Web Services Team, Creating HIPPA-Compliant Medical Data Applications With AWS,**
   http://awsmedia.s3.amazonaws.com/AWS_HIPAA_Whitepaper_Final.pdf, 2009-04-01

9. D. Obasanjo, Building Scalable Databases: Pros and Cons of Various Database Sharding Schemes,
   http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx, 2009-01-16

10. D. Pritchett, Shard Lessons,
    http://www.addsimplicity.com/adding_simplicity_an_engi/2008/08/shard-lessons.html, 2008-08-24

11. J. Hamilton, On Designing and Deploying Internet-Scale Services, 2007, *21st Large Installation System Administration conference (LISA 'O7)*, http://mvdirona.com/jrh/talksAndPapers/JamesRH_Lisa.pdf

12. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, 2004-12-01, In Proc. of the 6th OSDI,
    http://labs.google.com/papers/mapreduce-osdi04.pdf

13. T. Schlossnagle, *Scalable Internet Architectures*, Sams Publishing , 2006-07-31

14. M. Lurie, The Federation: Database Interoperability,
    http://www.ibm.com/developerworks/data/library/techarticle/0304lurie/0304lurie.html, 2003-04-23

15. E. Hammond, Escaping Restrictive/Untrusted Networks with OpenVPN on EC2,
    http://alestic.com/2009/05/openvpn-ec2, 2009-05-02

16. R. Bragg, The Encrypting File System,
    http://technet.microsoft.com/en-us/library/cc700811.aspx, 2009

17. S. Swidler, How to keep your AWS credentials on an EC2 instance securely,
    http://clouddevelopertips.blogspot.com/2009/08/how-to-keep-your-aws-credentials-on-ec2.html, 2009-08-31

18. **Amazon SQS Team, Building Scalable, Reliable Amazon EC2 Applications with Amazon SQS,**
    http://sqs-public- images.s3.amazonaws.com/Building_Scalabale_EC2_applications_with_SQS2.pdf, 2008

19. Microsoft Support Team, Best Practices For Encrypting File System (Windows),
    http://support.microsoft.com/kb/223316, 2009

20. Solaris Security Team, ZFS Encryption Project (OpenSolaris),
    http://www.opensolaris.org/os/project/zfs-crypto/, 2009-05-01