Vers une architecture d'auto-réparation sur le Cloud Computing

Faten Khemakhem*, Riadh Ben Halima*, Ahmed Hadj Kacem*

*Laboratoire ReDCAD, Université de Sfax, Tunisia
khemakhem.faten@gmail.com, riadh.benhalima@enis.rnu.tn, ahmed.hadjkacem@fsegs.rnu.tn

Résumé. Avec le Cloud Computing les organisations, institutions et entreprises n'ont plus besoin d'investir lourdement dans des ressources informatiques, nécessairement limitées, et exigeant une gestion lourde et coûteuse. Aujourd'hui, elles ont le choix de migrer vers un modèle Cloud Computing où elles peuvent louer des ressources en ligne. Ce modèle leur épargne les coûts de gestion interne, puisque les ressources informatiques sont administrées au niveau du fournisseur du Cloud. Le monitoring devient primordial vu qu'il jouera un rôle important dans les opérations des entreprises fournissant des services tels que les infrastructures (IaaS), plateformes (PaaS) et logiciels (SaaS). Le choix d'une solution stable et performante nécessite une étude approfondie des différents outils de monitoring existants pour guider le fournisseur Cloud afin de choisir l'outil le plus adéquat. Dans ce travail, nous procédons à une enquête sur des différentes solutions de monitoring existantes, à savoir l'open sources et les payantes afin de guider l'administrateur à choisir la solution la plus adéquate à son besoin.

Ces outils garantissent les phases de monitoring et d'analyse pour détecter la dégradation de performance d'une infrastructure de Cloud. Mais malheureusement, ces deux phases ne suffisent pas pour que le système réagit et résout les problèmes liés à cette dégradation. Pour celà, nous envisageons d'appliquer une architecture d'auto-réparation sur l'OpenStack comme étant notre Cloud privé.

1 Introduction

Vu que les services offerts par le Cloud doivent être disponibles, en terme de volume et de temps, selon les besoins du client et à sa demande, leur maîtrise devient primordiale, puisqu'ils doivent fonctionner pleinement et en permanence pour garantir la fiabilité et l'efficacité exigées (Shao et al. (2010)), d'une part. D'autre part, les problèmes liés à l'infrastructure du Cloud tels que les défaillances, les pannes, les coupures et les différents problèmes techniques doivent être réduits, du fait qu'une dégradation du système ou du réseau peut causer des pertes considérables. Afin de minimiser le nombre de ces pertes, une sorte de contrôle s'avère obligatoire ; c'est ainsi que la notion de 'monitoring' est devenue une tâche vitale pour les services de Cloud (Aceto et al. (2012)).

Le monitoring, étant la première étape de la boucle d'auto-réparation MAPE, est une étape clé vu qu'elle mesure les paramétres nécessaires et permet d'établir des tableaux de bord pour

observer l'état du Cloud. Vu la multitude d'outils de monitoring existants sur le marché, l'utilisateur ait l'embarras du choix car chaque solution a ses propres caractéristiques. Des travaux précédents (N'tchirifou (2010) et Jean-Louis et Olivier (0708)) se concentrent sur la comparaison des solutions open sources en se basant sur le monitoring du réseau et des infrastructures informatique. Mais l'évaluation des solutions de monitoring Cloud n'a pas été encore étudiée.

Notre travail consiste, en premier lieu, à étudier les différentes solutions existantes de monitoring du Cloud, à savoir Ganglia, Nagios, Zabbix, Zenoss, Cacti, Munin, OpenNMS, Centreon, LogicMonitor, HP OpenView et Amazon CloudWatch. En second lieu, nous classifions ces outils suivant des critères spécifiques (l'architecture, le niveau de monitoring, alertes, portage, etc.) et nous enchaînons par une synthèse. Notre objectif est de guider l'utilisateur à choisir l'outil de monitoring le plus adéquat à ses besoins afin de contrôler l'état de ses services sur le Cloud en cours d'exécution et de garantir la qualité de service (QdS) (Alhamazani et al. (2012)) de son Cloud.

En se basant sur cette étude comparative des outils de monitoring sur le Cloud, nous enchaînons par une expérimentation qui consiste à mettre en évidence un processus complet d'auto-réparation assurant la reconfiguration environnementale en cours d'exécution sur l'OpenStack.

Le reste de ce papier est structuré comme suit :

Dans la section 2, nous introduisons, en premier lieu, le contexte de monitoring qui vise à surveiller l'état des ressources et déclencher les notifications appropriées en cas de dégradation de performance d'une part. En deuxième lieu, nous présentons le mécanisme de monitoring en précisant la façon de mesure des paramètres de QdS (Aversa et al. (2012)). La section 3 expose et classifie les outils de monitoring les plus populaires dans le domaine de Cloud. Une synthèse de cette classification est établie dans la section 4. La section suivante est consacrée à la description des expérimentations d'auto-réparation effectuées sur l'OpenStack. La dernière section conclut le papier et donne quelques orientations pour les futurs travaux.

2 Contexte

2.1 Monitoring des services de Cloud Computing

La gestion d'un nuage est une tâche difficile qui exige le déploiement et la coopération des mécanismes de prise de décision différente. Par conséquent, le déploiement d'un système fiable de monitoring est une importance primordiale dans le Cloud. Le monitoring joue un rôle important dans l'amélioration de la qualité des services dans le cloud computing. Il aide les développeurs à identifier les défauts dans les services de Cloud pour garantir la qualité associée à la grande échelle (Shao et al. (2010)).

Le monitoring est une activité de surveillance et de mesure d'une activité informatique.

2.2 Mécanismes de monitoring

Il existe deux mécanismes pour la collecte des données de monitoring, soit à l'aide d'un agent installé sur chaque équipement à contrôler soit à distance et sans agent spécifique. L'agent est une application légère en termes de CPU et de mémoire, qui remplit deux fonctions :

– l'utilisation des diverses moyens de collecte de données ;

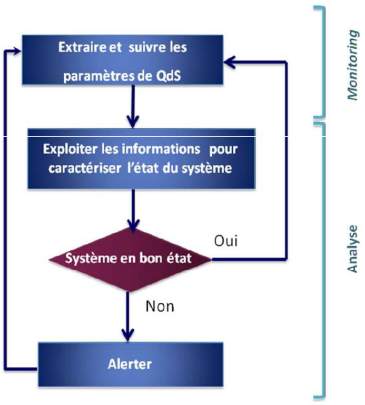


FIG. 1 – *Monitoring de Cloud*

– l'envoi des données recueillies aux serveurs pour le traitement, le stockage et l'alerte.

• ***Monitoring avec agent***

Les outils de monitoring facilitent le contrôle des infrastructures par l'intermédiaire des agents qui détectent automatiquement tout ce qui concerne un équipement informatique (les attributs et les performances).

Le monitoring avec agent nécessite l'installation de ce dernier sur chaque machine à contrôler.

• ***Monitoring sans agent***

Diverses méthodes existent pour le monitoring, mais la plus simple consiste à superviser les hôtes distants directement par réseau et sans agent comme l'utilisation du 'Simple Network Management Protocol' (SNMP).

2.3 Mode de contrôle des données de monitoring

Les logiciels dédiés pour le monitoring permettent d'obtenir toutes les informations concernant un appareil informatique de deux modes différentes :

• ***Passif***

La vérification est planifiée et exécutée localement dans l'hôte à superviser, puis le résultat est envoyé au serveur du moniteur.

• ***Actif***

Le serveur du moniteur est l'initiateur de la vérification : l'hôte reçoit la demande de vérification de la part du serveur , il exécute la vérification, puis il renvoie le résultat.

La différence majeure entre le contrôle actif et celui passif est que dans le premier cas le serveur du moniteur lance et exécute la demande des données de monitoring, alors que dans le cas passif, cet acheminement est géré par une application externe.

3 Comparaison des outils de monitoring sur le Cloud

3.1 Outils de monitoring open sources

La plupart des outils de monitoring open sources offrent des services au niveau de l'infrastructure IaaS et PaaS de Cloud. Ganglia, Nagios, Zabbix, Cacti, Munin, Zenoss, Open NMS et Centreon sont les solutions les plus répondues dans le monde de monitoring open sources.

3.1.1 Ganglia

Ganglia (art) a été développé initialement à l'université de Berkeley sous licence BSD [1]. Il fournit une solution robuste et peu consommatrice en ressources, pour le monitoring des performances de clusters qui peuvent comporter des milliers de nœuds.

Cet outil est basé sur une conception hiérarchique qui est constituée d'une fédération des clusters de grandes dimensions. Il s'appuie aussi sur une connexion multicast pour contrôler les états des nœuds.

3.1.2 Nagios

Le développement de Nagios (Nag) a commencé en 1999 sous le nom de NetSaint avec la licence [2], par 'Ethan Galstad'. En 2002, à cause d'un problème de propriété intellectuelle sur le nom, NetSaint version 0.7 devient Nagios 1.0 pour éviter la confusion avec le produit 'Saint'. Il impose le mode actif et passif pour collecter les données de monitoring de façon hiérarchisée.

Cet outil est accessible via une interface web développée en PHP, qui offre une carte de statut en 2D et en 3D pour consulter l'état du réseau.

3.1.3 Zabbix

Zabbix (Zab) a été créé en 2002 par 'Alexei Vladishev', il est actuellement soutenu par 'ZABBIX SIA'. Cet outil est écrit et distribué sous la licence GNU et il possède une version entreprise (payante) qui apporte des fonctionnalités supplémentaires comme les accords de soutien annuels.

Zabbix offre un panel d'outils graphiques multilingue pour un meilleur contrôle de différents services (IaaS et PaaS) de Cloud.

3.1.4 Cacti

Cacti (Cac) est un outil de monitoring, purement open source, dédiée aux fournisseurs d'IaaS et PaaS sur le Cloud. Il est créé en 2001 par 'Ian Berry', 'Larry Adams', 'Tony Roman', 'JP Pasnak', 'Jimmy Conner' et 'Reinhard Scheck' et actuellement développé et soutenu par 'The Cacti Group' sous licence GNU.

Cette solution est utilisée par plusieurs sociétés et laboratoires comme l'Université Nancy2 et INRA [3] et elle s'adapte à des environnements comportant jusqu'aux 160 000 nœuds qui sont

1. 'Berkeley software distribution license' est une licence libre utilisée pour la distribution de logiciels
2. Licence publique générale
3. Institut National de Recherche Agronomique

un nombre réduit par rapport à d'autres solutions de monitoring telles que Nagios, Zabbix, Zenoss, OpenNMS et Centreon.

3.1.5 Munin

Munin (Mun) est un outil basé sur RRDTool [4], écrit en Perl [5]. Il dispose d'une interface de fonctionnalités équivalentes à Ganglia. Munin est plus axé pour le monitoring des services d'IaaS et de PaaS en Cloud Computing. Il peut observer les performances des ordinateurs, des réseaux et des applications via le protocole SNMP ou un agent (Munin-node) installé sur chaque nœud à contrôler.

Cette solution bénéficie aussi d'une présentation graphique monolingue (anglais) et d'une architecture centralisée pour le déploiement des agents et la récupération des données de monitoring.

3.1.6 Zenoss

Zenoss (Zen) représente une alternative à des plateformes de monitoring payantes comme HP OpenView. Il assure le suivi des services de Cloud en utilisant les fonctions intégrées dans les systèmes d'exploitation et les applications. Il est basé sur un modèle temps-réel pour fournir une analyse au niveau des couches IaaS et PaaS de Cloud.

3.1.7 OpenNMS

Le projet OpenNMS (Ope) a été créé en Juillet 1999 par 'Day One' et a été enregistré sur SourceForge [6] en Mars 2000. OpenNMS est une plateforme de monitoring soutenue par la communauté 'OpenNMS Group'. Tout le code associé au projet est disponible sous la licence GNU. Il apporte la puissance, l'évolutivité que demandent les entreprises et les opérateurs de Cloud. Mais il est globalement moins avancé que Nagios. Sa configuration est très lourde et difficile à gérer, même lorsque le nombre d'éléments supervisés est réduit.

3.1.8 Centreon

Centreon (Cen) est un outil de monitoring, sous licence GPL, basé sur le moteur de monitoring libre Nagios. Il est édité par 'Romain le Merlus' et 'Julien Mathis' puis il est soutenu par la société 'Merethis' depuis 2003 sous le nom d'Oreon. En 2007, il devient Centreon. En tant qu'éditeur, 'Merethis' présente des extensions payantes mais soumet le cœur de Centreon sous licence GPL aux utilisateurs en accès.

Centreon simplifie l'administration de Nagios grâce à la surcouche applicative qui se positionne au dessus de ce dernier.

4. Un outil de gestion de base de données RRD (Round-Robin database) créé par Tobi Oetiker
5. Un langage de programmation créé par Larry Wall en 1987 et reprenant des fonctionnalités du langage C et des langages de scripts sed, awk et shell (sh)
6. Plateforme d'hébergement de projets

3.2 Outils de monitoring propriétaires

En fait, les solutions de monitoring les plus connus incluent Amazon CloudWatch, HP OpenView et LogicMonitor. Nous les décrivons brièvement dans la suite.

3.2.1 LogicMonitor

LogicMonitor (Log) est un logiciel de monitoring qui se classe parmi les leaders du domaine, dédié aux services (IaaS et PaaS) de Cloud Computing et créé à Santa Barbara, en Californie. Il propose un puissant outil Web qui utilise une architecture distribuée pour contrôler d'une manière proactive les infrastructures technologiques et mesurer la performance opérationnelle des entreprises.

3.2.2 Amazon CloudWatch

Amazon CloudWatch (Clo) est un outil de monitoring dédié pour contrôler les ressources d'Amazon Web Services (AWS) et les applications qui tournent sur AWS en temps réel.

CloudWatch prend en charge le provisionnement dynamique (appelé auto-scaling), le suivi et l'équilibrage de charge d'une façon centralisée. Son plus grand avantage est qu'il ne nécessite aucun logiciel supplémentaire à installer et aucun site supplémentaire pour accéder aux applications à travers.

3.2.3 HP Open View

HP Open View (HP) est un outil classé parmi les meilleurs outils pour la gestion des performances systèmes et réseaux. Il est destiné aux moyennes et grandes entreprises qui souhaitent avoir une vue global de leur réseau et de son état. HP OpenView assure une gestion centralisée des identités des utilisateurs et leurs droits d'accès via l'interface d'administration.

Cette solution offre deux méthodes pour visualiser l'état de l'infrastructure de Cloud, soit à travers un client Java installé chez le client soit une interface accessible via un navigateur web.

4 Synthèse

Le monde du logiciel de monitoring est vaste. Comment s'y retrouver parmi les nombreuses solutions qui existent et trouver celle qui répond au mieux à nos besoins ?

Pour avoir une réponse convaincante, nous avons réalisé une étude comparative de ces onze outils en se basant sur différents critères de classification.

4.1 Critères de classification

Tableau 1 et 2 montrent les études comparatives entre ces différentes solutions de monitoring (open sources et propriétaires). Parmi les critères de classification, nous citons :

1. ***Niveau de monitoring*** : définir les services (IaaS, PaaS ou SaaS) de Cloud Computing à contrôler.

2. ***Interface d'administration personnalisée*** : la personnalisation de l'interface d'administration et aussi la manière de monitoring comme l'établissement des propres seuils de performances et l'identification des causes d'une alerte.
3. ***Création de graphes complexes*** : c'est l'utilisation de plus qu'une métrique dans un graphe (Mise en relation des métriques).
4. ***Possibilité de communication avec d'autres outils de monitoring*** : il existe des outils de monitoring qui travaillent en coopération pour offrir un meilleur contrôle comme Centreon qui présente une surcouche à l'interface graphique de Nagios.
5. ***Cartographie*** : qui désigne la réalisation et l'étude des cartes géographiques pour afficher le statut de l'infrastructure réel de l'environnement à contrôler.
6. ***Architecture*** : spécifier l'architecture (hiérarchique, centralisée ou distribuée) utilisée pour modéliser l'emplacement de nœuds à contrôler.
7. ***Stockage centralisé des données de monitoring (SGBD)*** : les outils de SGBD utilisés pour stocker les informations de monitoring générées par tous les nœuds.
8. ***Stockage local des données de monitoring*** : les moyens de SGBD utilisés par chaque nœud pour stocker les données de monitoring localement.

	Ganglia	NAGIOS	Zabbix	Cacti	Munin	Zenoss	OpenNMS	Centreon
Niveau de monitoring	IaaS	IaaS, PaaS	IaaS, PaaS	IaaS, PaaS	IaaS, PaaS	IaaS, PaaS	IaaS	IaaS, PaaS
Architecture	Hiérarchique	Hiérarchique, Centralisée ou Distribuée	Distribuée	Hiérarchique	Centralisée	Distribuée	Distribuée	Centralisée ou Distribuée
Visualisation des données de monitoring	-Tableau de bord -Graphes	-Tableau de bord -Rapports -Graphes	-Tableau de bord -Rapports -Graphes	-Tableau de bord -Rapports -Graphes	-Graphes -Tableau de bord	-Tableau de bord -Rapport -Graphes	-Tableau de bord -Rapport -Graphes	-Tableau de bord -Rapport -Graphes
Interface d'administration	-Graphique	-Graphique -Ligne de commande	-Graphique	-Graphique -Ligne de commande	-Graphique	-Graphique	-Graphique	-Graphique -Ligne de commande
Langage de programmation de l'interface d'administration	PHP	PHP	PHP	PHP	Perl	Python	JSP, HTML, JAVA, PERL et C	PHP
Traduction de l'interface d'administration	Monolingue (Anglais)	Monolingue (Anglais)	Multilingue	Monolingue (Anglais)	Monolingue (Anglais)	Monolingue (Anglais)	Monolingue (Anglais)	Multilingue (anglais, portugais et allemand)
Interface d'administration personnalisée	Non	Non	Oui	Non	Non	Non	Oui	Oui
Création de graphes complexes	Non	Oui	Oui	Oui	Oui	Oui	Non	Non
Zoom sur les graphes et défilement sur une période sélectionnée	Oui	Non	Oui	Oui	Oui	Oui	Oui	Oui
Interface d'administration dédiée au mobile	Avec	Avec	Avec	Sans	Oui	Oui	Oui	Non
Accès mobile	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Alertes	Sans	Email, SMS, Pager, Audio alerts, Méthode définie par l'utilisateur, etc.	Email, SMS, Jabber ou Exécution de commandes	Email	E-mail, Alertes via un lancement des scripts	E-mail, Pager ou Exécution de commandes	Asterisk call origination, E-mail, Growl, IRC (Internet Relay Chat)/ Pager, etc.	Email, Pager, Jabber
Portage	Distributions Linux (Fedora / RedHat / CentOS / Debian)	Distributions Linux, AIX, HP-UX, Solaris, Irix, Compaq Tru64, UNIX, SCO Unixware, FreeBSD, NetBSD, OpenBSD ou FREESCO.	Distributions Linux,IBM AIX, FreeBSD, NetBSD, OpenBSD, HP-UX, Mac OS X, Solaris ou Windows	Windows, Distributions Linux, OpenBSD, NetBSD, FreeBSD.	Distributions Linux, FreeBSD, NetBSD, Solaris, Mac AIX, OS X, HP-UX.	Distributions Linux ou Windows	Distributions Linux, Mac OS X, NetBSD, Solaris, AIX, HP-UX ou Windows.	Distributions Windows, Ubuntu, Fedora, Centos, Debian, etc.
Moniteurs	-Gmond au niveau des nœuds -Gmetad au niveau des clusters	NRPE, SNMP, NSCA, SSH, NSClient++ ou WMI	SNMP, IPMI, SSH ou Agent Zabbix	SNMP, SSH ou WMI	Munin-node	SNMP, SSH, WMI ou JMX	SNMP, JMX, HTTP, JDBC ou NSClient++	SNMP
Les métriques observées au niveau IaaS réel (serveur physique)	CPU_user, CPU_num, cpu_wait ,sys_cpu_queue_len, CPU_ speed, CPU_nice, CPU_system, CPU_idle, CPU_aidle, Swap_free, Swap_total, Disk_total, etc.	CpuLoad, CpuUser, CpuSystem, CpuNice, Cpulowait, Cpuidle, CpuSteal, DiskUsage, CurrentUserCount, ProcessInformation, LoggedInUsers, etc.	Hostname, uname , localtime, cpu.intr, cpu.load , cpu.num , cpu.util , cpu.user, cpu. Nice, cpu. Ralenti, cpu.iowait, mem.size, mem. commun, mem.tampons, etc.	CPU_Used, Memory Usage, Load Average, Logged in Users, Processes, Memory Buffers, Real Memory, Swap Space, etc.	disk i/o stat, Cpusystem, cpuuser, cpunice, cpuidle, cpuiowait, load , etc.	cpureservedcapacity, cpuusage, cpuusagemhz, diskusage, memactive, memgranted, memswapused, sysuptime, etc.	swap in, swap out, io buff ram, shared mem, total swap, total real mem, cpu user, cpu nice, cpu wait, cpu system, disk space used, uptime, etc.	memory usage, cpu load, disk usage, service states, running processes, etc.
Les métriques observées au niveau IaaS virtuel (VMs)	CPU_ speed, CPU_nice, CPU_system, CPU_idle, CPU_aidle, swap_free, Disk_total, etc.	mem_overhead, mem_overall, cpu_wait, mem_active, io_usage, runtime_con, DiskFreeSpace, DiskSize, etc.	vm.memory.buffers, vm.memory.cached, vm.memory.free, vm.memory. shared,vm.memory.total.	memUtil, cpuUtil, disk read, disk write, etc.	Cpusystem, cpuuser, cpunice, cpuidle, cpuiowait, etc.	cpuusage, cpuusageavg, cpuusagemax, cpuusagemin, diskusage, memconsumed, memoverhead, etc.	virtual memory size, etc.	mem_overhead, mem_overall, cpu_wait, mem_active, io_usage, runtime_con, runtime_cpu, etc.
Les métriques observées au niveau PaaS	Sans	-MySQL: Uptime, Threads, Questions, Slow queries, Opens, Flush Tables, etc.	-SQLServer: Utilisation CPU, Taille, Taille des logs, Deadlocks, Transactions, etc.	MailReceived, MailSent, MailReject, TotalReceived, TotalSent, TotalReject, etc.	-PostgreSQL : postgres_bgwriter, postgres_cache, etc.	Apache Tomcat Application Server: Tomcat Cache, Tomcat Core, Tomcat Thread, etc.	Sans	-MySQL: Uptime, Threads, Questions, tables, Queries per second avg, etc.
Les métriques observées sur l'infrastructure réseau	Bytes_in, Bytes_out, MTU, location, pkts_in, pkts_out, etc.	in_Bps_max, in_Bps_avg , in_Bps _min, out_Bps_max, out_Bps_avg, TrafficTotal, etc.	net.tcp.listen, net.tcp.service, net.tcp.service.perf, net.tcp.port, swap.out, etc.	InBound, OutBound, TotalIn, TotalOut, PacketsErrors, ProcessKeepAlive, etc.	eth0 traffic, eth0 errors, netstat, etc.	Network in, Network out, Traffic send, Traffic received, etc.	coDot11PktsRate1, coDot11PktsRate2, coDot11InPkts, etc.	in_Bps_max, in_Bps_avg , in_Bps _min, out_Bps_max, In bytes, Out bytes,
Format d'exportation les rapports de monitoring	Non	CSV	CSV	XML	Sans	PDF, CSV, Excel, Word, HTML ou JPG.	CSV, HTML ou PDF	CSV ou XML
Possibilité de communication avec d'autres outils de monitoring	Nagios	Centreon, Munin, Ganglia, MRTG, OpenNMS	Cacti,OpenNMS	Zabbix	Nagios via NSCA	Non	Nagios, Zabbix	Nagios
Scalabilité	S'adapte à des milliers de nœuds	S'adapte à des milliers de nœuds	S'adapte à des dizaines de milliers de noeuds	S'adapte à 160 000 nœuds	Non spécifié	S'adapte à des milliers de nœuds	S'adapte à des dizaines de milliers de noeuds	S'adapte à des milliers de nœuds
Elastique	Non	Oui	Oui	Non	Non	Oui	Non	Non
Licence	BSD	GPL ou Commercial	GPL	GPL	GPL	GPL ou Commercial	GPL	GPL ou Commercial
Propriétaire	Université Berkeley en Californie	Créé par Ethan Galstad et actuellement activement développé et soutenu par "Nagios Enterprises"	Créé par Alexei Vladishev et actuellement activement développé et soutenu par "ZABBIX SIA"	Créé par Ian Berry, Larry Adams, Tony Roman, JP Pasnak, Jimmy Conner et Reinhard Scheck et soutenu par 'The Cacti Group'	Soutenu par 'Software Edgewall'	Créé par Erik Dahl et actuellement développé et soutenu par Communauté de "Zenoss Inc"	Créé par 'day one'et soutenue par la communauté 'OpenNMS Group .Inc' depuis 2002	Créé par Romain le Merlus et Julien Mathis et soutenu par la société 'Merethis'
Cartographie	Non	Oui	Oui	Oui	Non	Oui	Oui	Oui

TAB. 1 – *Solutions de monitoring open sources*

	LogicMonitor	HP OpenView	Amazon CloudWatch
Niveau de monitoring	IaaS, PaaS	IaaS, PaaS	IaaS, PaaS
Architecture	Distribuée	Centralisée	Centralisée
Visualisation des données de monitoring	-Tableau de bord -Rapport -Graphes	-Tableau de bord -Rapport -Graphes	-Tableau de bord -Rapport -Graphes
Interface d'administration	Graphique	Graphique	-Graphique -Ligne de commande
Langage de programmation de l'interface d'administration	PHP	Java, HTML	Non spécifié
Traduction de l'interface d'administration	Multilingue	Multilingue (français, anglais et allemands)	Multilingue
Interface d'administration personnalisée	Oui	Oui	Oui
Création des graphes complexes	Oui	Non	Oui
Zoom sur les graphes et défilement sur une période sélectionnée	Oui	Oui	Oui
Interface d'administration dédiée au mobile	Oui	Non	Oui
Accès mobile	Oui	Oui	Oui
Alertes	E-mail, SMS, appel téléphonique, Texte vocale, rapport périodique, etc.	Email ou Pager	Email, SMS, etc.
Portage	Multi-plate-formes	HP-UX, Windows, Solaris, Linux, AIX, Tru64, NetWare, MPE/iX, Sinix, Dynix/ptx, Irix, AS/400, OS/390	Windows, Linux
Scalability	S'adapte à des milliers de nœuds	S'adapte à 1500 noeuds	S'adapte à la plate-forme Amazon seulement
Elastique	Oui	Oui	Oui
Moniteurs	SNMP, HTTP, ICMP, WMI ou JDBC	SNMP ou WMI	Non spécifié
Format d'exportation les rapports de monitoring	HTML, PDF ou CSV	XML, HTML ou CSV	Non spécifié
Stockage centralisé des données de monitoring (SGBD)	-MySQL -SQL Server	SQL Server	Amazon RDS
Stockage local des données de monitoring	Buffer	MIB ou SGBDR Oracle	Non spécifié
Dédié seulement pour le Cloud	Oui	Oui	Oui
Les métriques observées au niveau IaaS réel	System type, CPU used, Disks data rate read, Disks data rate write, Disk latency read, Disk latency write, Disk operation read, Disk operation write, Status, Mem swap rate, Mem swap Percent, Mem used percent etc.	Non spécifié	CPU usage, disk reads, disk writes, Memory Utilization, Memory Used, Memory Available, Page File Utilization, Page File used, Page File available, Disk Space Utilization, Disk Space Used, Disk Space Available, etc.
Les métriques observées au niveau IaaS virtuel (VMs)	Mem used percent, CPU used, Disks data rate read, Disks data rate write, Mem Granted, Mem Active, Mem Shared, Swap in, Swap out, VM Disk operations, VM Memory used etc.	Non spécifié	Non spécifié
Les métriques observées au niveau PaaS	"Mailserver" : received mail by SMTPD, received mail from local, rejected by local SMTPD, local SMTPD_errored_reject etc.	Non spécifié	Non spécifié
Les métriques observées sur l'infrastructure réseau	Bandwidh saved, VPN Packets In, VPN Packets Out, VPN traffic In, VPN traffic Out etc.	Non spécifié	Non spécifié
Propriétaire	Créé et soutenu par la communauté de 'LogicMonitor'	Créé et soutenu par la société 'HP Software'	Créé et soutenu par 'Amazon'
Cartographie	Avec	Avec	Avec
Page d'accueil	http://www.logicmonitor.com	http://www8.hp.com/us/en/software/enterprise-software.html	http://aws.amazon.com/fr/cloudwatch/

TAB. 2 – *Solutions de monitoring propriètaires*

4.2 Discussion

Au vu des tableaux 1 et 2, il est clair que chaque solution possède son propre architecture. Certaines solutions sont basées sur une architecture hiérarchique (Ganglia, Cacti). D'autres s'appuient sur une architecture centralisée (Amazon CloudWatch, HP OpenView, Munin), alors que d'autres outils utilisent une architecture distribuée (LogicMonitor, Zabbix, Zenoss, OpenNMS). Mais, il existe aussi d'autres solutions qui s'adaptent au environnement à contrôler en utilisant différentes architectures (Nagios, Centreon).

Le tableau 3 presente une comparaison des outils de monitoring open sources selon la complexité d'installation et d'utilisation.

Ganglia, Zabbix, Cacti, Munin et OpenNMS sont des outils réellement libres. Ils n'offrent pas une 'version entreprise' comme le font certaines solutions de monitoring (Nagios, Zenoss et Centreon). Le mode d'installation varie énormément selon le système d'exploitation utilisé et les dépendances qui doivent êtres installées.

	Installation			*Documentation*	*Facilité d'emploi*
	Type	*Facilité*	*Configuration*		
Ganglia	- Manuelle - Package RPM	++	+++	Bien documenté	+++
Nagios	- Manuelle - Package RPM	+	+	Bien documenté	++
Zabbix	- Manuelle - Package RPM	++	+	Très bien documenté	++
Cacti	- Manuelle - Package RPM	+++	+++	Très bien documenté	++
Munin	- Manuelle - Package RPM	++	+	Assez bien documenté	++
Zenoss	- Manuelle - Exécutable - Package RPM	++	++	Bien documenté	+++
OpenNMS	- Manuelle - Package RPM	++	++	Bien documenté	+++
Centreon	- Manuelle - Package RPM	+	+	Assez bien documenté	++

+ Difficile ++ Moyenne +++ Facile

TAB. 3 – *Comparaison des outils de monitoring open sources selon la complexité d'installation et d'utilisation.*

Le tableau 3 montre que Nagios s'avère la solution la plus complexe de point de vue d'installation et de configuration, mais la plus utilisée dans le Cloud. Il joue le rôle d'un complément pour d'autres outils de monitoring, tels que Centreon, Munin, Ganglia et OpenNMS qui ne se suffisent pour contrôler de manière exhaustive une infrastructure. Ses fonctionnalités avancées permettent de gérer pratiquement la majorité des cas.

Centreon est le plus utilisé avec Nagios, il agit comme un intermédiaire entre l'administrateur et les fichiers de configuration de Nagios. Il enregistre les configurations effectuées par l'administrateur dans une base de données, puis il modifie les fichiers de configuration de Nagios en fonction du contenu de la base de données. Ceci a permis de simplifier le travail de l'administrateur, contrairement à l'utilisation de Nagios seul.

Munin propose une configuration très simple sous forme de fichiers et de liens symboliques. Il offre des fonctionnalités équivalentes à Ganglia. Le point faible de cette solution de monitoring est le fait qu'elle n'assure pas des techniques de sécurité au niveau administration.

Certains outils de monitoring se basent sur la création des templates de configuration qui peuvent s'avérer un peu complexe au premier abord avec les différentes couches (Data Templates, Graph Templates, Host Templates, etc.) comme le cas de Cacti.

Zenoss représente une alternative à des plateformes de monitoring payantes comme HP OpenView. Il a l'avantage de se suffire à lui même au contraire de Nagios où il faut l'associer à quelques produits pour avoir une solution réellement exploitable.

L'avantage du logiciel open source est justement que personne ne peut se l'approprier pour en faire une grosse boîte noire. Mais il reste accessible à tout le monde. L'autre avantage est que toute amélioration du logiciel profite à tout le monde, y compris à ceux qui gagnent de l'argent avec le logiciel. Inversement avec les logiciels éditeurs, dont la duplication, la modification ou l'usage sont limités. Les limitations légales, liées aux choix des ayants droit, sont souvent encadrées par un contrat de licence. Par exemple, pour avoir des services supplémentaires, il faut acheter des modules à part comme le cas de HP OpenView et LogicMonitor.

L'inconvénient majeur d'Amazon CloudWatch, comme étant une solution payante, est qu'elle ne supporte que la plate-forme Amazon seulement, ce qui limite son utilité globale.

4.3 Aide à la prise de décision

On définit la décision comme étant un acte par lequel le fournisseur de Cloud opère un choix entre plusieurs options permettant de sélectionner une solution de monitoring satisfaisante à son besoin.

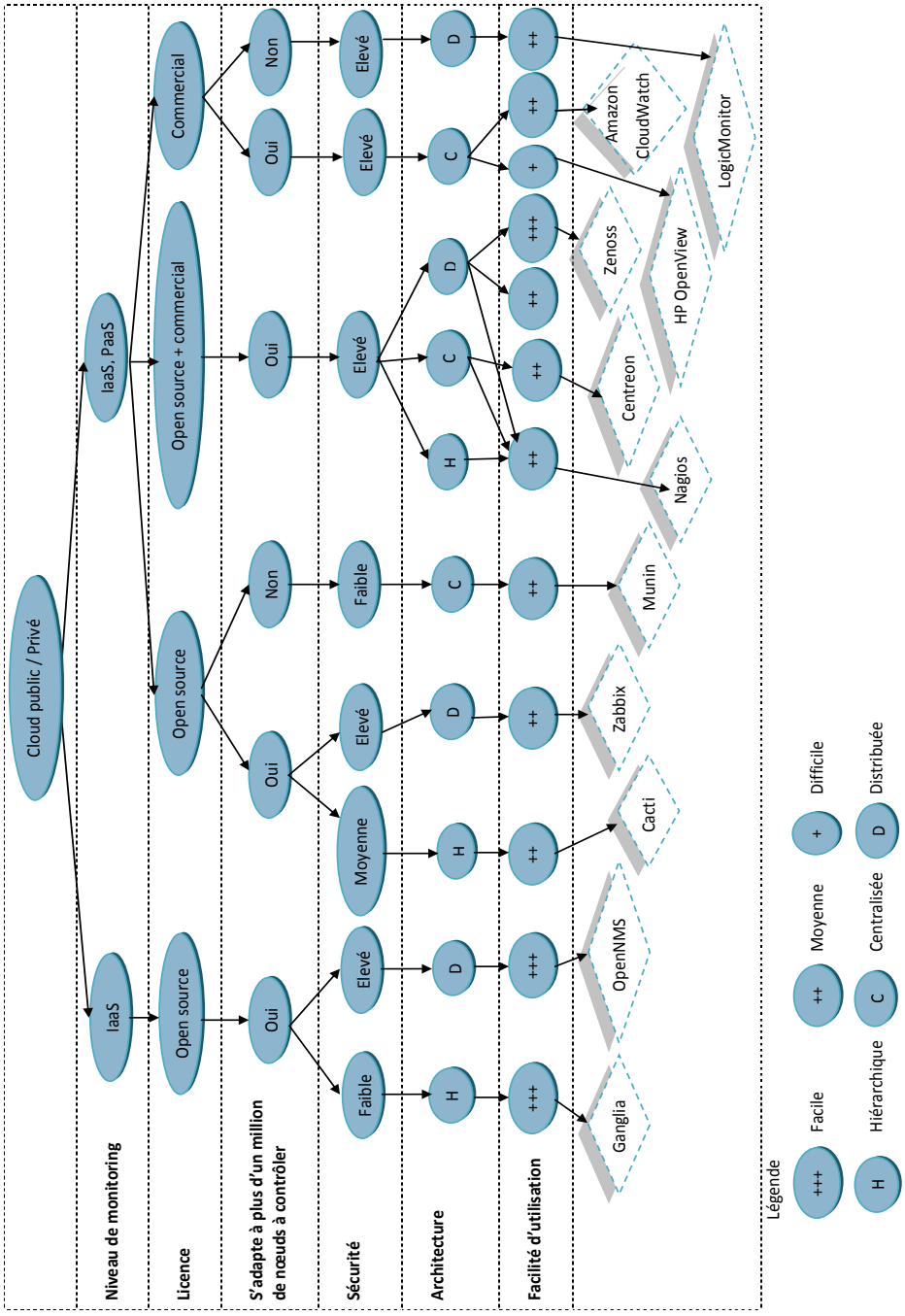
Le fournisseur de Cloud en tant qu'une structure bien organisée pose une question relative à la manière de la prise de décision en adéquation avec les objectifs poursuivis par l'entreprise : Comment prenons-nous nos décisions face à ces nombreuses solutions ?

Pour avoir une réponse claire, la figure 4 présente une arborescence qui aide à la prise de décision et montre la différence entre les outils de monitoring déjà étudiés en utilisant des critères généraux et communs entre eux. Elle donne ainsi une vue d'ensemble d'une situation donnée pour chaque solution.

Tout d'abord le type de Cloud influence le processus décisionnel vue que le Cloud public est plus large que le Cloud privé en terme de nombre de nœuds et de puissance. Ainsi, le fournisseur de Cloud doit prendre en compte la possibilité d'une extention de son Cloud qui, par conséquent, nécessite un outil de monitoring plus performant et scalable.

Pour que les décideurs puissent choisir un tel outil de monitoring, nous proposons ce processus de prise de décision :

1. Définir le niveau de Cloud à contrôler ;
   - IaaS : le fournisseur de Cloud demande le contôle de son infrastructure réelle seulement.
   - IaaS et PaaS : le fournisseur de Cloud demande le contôle de son infrastructure réelle et des services offerts au niveau de la couche PaaS.
2. Définir les conditions dans lesquelles cet outil peut être utilisé ;
   - Open source : l'outil de monitoring est gratuit.



TAB. 4 – *Arborescence d'aide à la prise de décision pour choisir l'outil de monitoring le plus adéquat*

  - Open source & commercial : le noyau de la solution est gratuit, mais pour avoir des services supplémentaires, il faut acheter des modules en plus.
  - Commercial : l'outil de monitoring est payant.

3. Spécifier le nombre de nœuds maximal à contrôler : le fournisseur de Cloud doit prévoir la dimension de son Cloud en terme de nombre de nœuds en prenant compte la possibilité d'une extension.
4. Définir le niveau de sécurité souhaité pour éviter le risque de perte d'information : utiliser un ensemble de moyens de sécurisation nécessaires pour garantir la sécurité d'accès à l'interface graphique de la solution de monitoring et le contrôle de transport des données collectées entre les nœuds à contrôler. La qualité de sécurisation (Faible, moyenne, élevée) est mésurée par les moyens de sécurisation utilisés à savoir la connecxion SSH, la connecxion SSL, le cryptage, l'authentification, etc.
5. Spécifier l'architecture utilisée dans la modélisation de Cloud : les nœuds du Cloud sont déployés dans le réseau d'une façon centralisée, hiérarchique ou distribuée.
6. Définir le degré de la facilité d'utilisation souhaitée (facile, moyenne ou difficile).

5 Vers l'auto-réparation

Dans la section précédente, nous avons présenté une étude comparative des outils de monitoring qui offrent une vue globale du fonctionnement de l'infrastructure du Cloud. Mais à l'apparition d'une dégradation de performance au niveau du service de Cloud, ils génèrent des alertes sans réagir.

Pour montrer la faisabilité de notre expérimentation, nous visons enchainer la phase de monitoring par d'autres phases (analyse, planification et exécution) pour que notre système réagit. Ce processus est appelé 'auto-réparation' (Daubert et al. (2011)).

5.1 Architectures d'auto-réparation

Un système auto-réparable est une entité logicielle apte à réagir par reconfiguration aux changements de son environnement au cours de son exécution. L'auto-réparation d'entités logicielles a été introduite au début des années 2000 (Daubert et al. (2011)). Plusieurs architectures d'auto-réparation sont proposées à savoir MAPE-K (Kephart et Chess (2003)), A-MAPE-K (Maurer et al. (2011)) et KACMM (Martinovic et Zoric (2012)).

5.1.1 MAPE-K

Cette architecture est proposée par Jeffrey O. et David M. en 2003 et fondée par IBM (Kephart et Chess (2003)). Elle introduit un gestionnaire qui est chargé de contrôler une entité à travers quatre étapes de monitoring, analyse, planification et exécution à l'aide d'une base de connaissances.

	MAPE-K [KC03]	A-MAPE-k [MBEB11]	KACMM [MZ12]
Cycle d'auto-réparation	Monitoring, Analyse, Planification et Exécution	**Adaptation**, Monitoring, Analyse, Planification et Exécution	Monitoring, Analyse, Planification, Exécution, **Mettre à jour** et **Configuration**
Corriger les incohérences entre les attributs	×	✓	×
Détecter les simulations entre les attributs	×	✓	×
Intervention humaine	✓	×	×
Auto-configuration	✓	✓	✓
Auto-optimisation	✓	✓	×
Auto-protection	✓	✓	×
Auto-réparation	✓	✓	✓
Gestion de QdS	✓	✓	✓
Gestion de SLA	×	✓	×
Créé par	Jeffrey O, David M	Michael M, Ivan B, Vincent C. E, and Ivona B	Goran M et Bruno Z
Soutenu par	IBM	Institut des Systèmes d'Information, Université Technologique, Vienne	Faculté de génie électrique, Université d'Osijek, Croatie
Date de création	2003	2011	2012

TAB. 5 – *Comparaison des architectures d'auto-réparation*

5.1.2 A-MAPE-k

L'architecture A-MAPE-K (Maurer et al. (2011)) est une extension du modèle proposé par IBM (MAPE-K) en ajoutant une phase supplémentaire d'adaptation. Ce modèle inclut la gestion de SLA et la correction les incohérences et les simulations entre les attributs.

5.1.3 KACMM

KACMM (Martinovic et Zoric (2012)) (**K**nowledge-based **A**utonomic **C**onfiguration **M**anager **M**odel) est basée sur l'architecture MAPE-K en intègrant également la mise à jour et la configuration du gestionnaire pour améliorer la stabilité du système.

Dans le tableau 5, nous détaillons les différentes architectures d'auto-réparation proposées dans le Cloud. Cette comparaison nous mène au choix d'une architecture spécifique pour notre expérimentation.

5.2 Architecture d'auto-réparation adoptée

L'architecture proposée par IBM 'MAPE-K' est la première architecture utilisée dans les systèmes d'auto-réparation. Les autres architectures se sont des extensions de 'MAPE-K' en

integrant d'autres phases pour avoir la robustesse, la proactivité et l'adaptation dans le Cloud.

Après avoir présenté des exemples d'architecture d'auto-réparation, nous passons à détailler les phases de l'architecture adoptée.

La figure 2 montre les quatre phases de l'architecture MAPE-K.

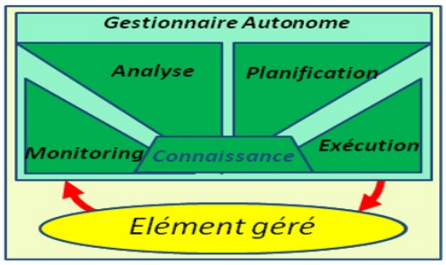


FIG. 2 – *Les étapes de la boucle MAPE (Kephart et Chess (2003))*

5.2.1 Monitoring

Le monitoring est la phase initiale de la boucle MAPE. Il consiste à observer des métriques et à mesurer les paramétres de QdS. Ce contexte inclut les attributs d'hôtes (l'adresse IP, le type de système informations, etc.) et ses performances (la charge CPU, l'utilisation du disque, le temps d'exécution, etc.).

5.2.2 Analyse

Il correspond à la phase d'exploitation des valeurs obtenues par le monitoring. Il permet de s'assurer du bon fonctionnement de l'application et détecter une éventuelle dégradation des valeurs de métriques observées (Ben-Halima (2009)).

5.2.3 Planification

Elle est exécutée suite à la détection d'une dégradation. L'objectif ici est d'identifer et de localiser l'origine des services défaillants. Cette phase est chargée aussi de décider la stratégie d'auto-réparation et la décomposer en un plan d'actions élémentaires.

Daubert et al. (Daubert et al. (2011)) a identifié les différents types d'actions d'auto-réparation selon leur impact sur l'entité à reconfigurer.

- **Action paramétrique** : permet de modifier un ou plusieurs paramètres de l'entité (par exemple modification du niveau de log [7] dans une application).
- **Action fonctionnelle** : permet de remplacer le code d'une fonction de l'entité par une autre sans changer son comportement (par exemple remplacement d'une fonction de tri à bulles par une fonction de tri rapide).

7. Désigne l'enregistrement séquentiel dans un fichier ou une base de données de tous les événements affectant un processus particulier (application, activité d'un réseau informatique, etc.)

- **Action comportementale** : consiste à changer le comportement de l'entité (par exemple ajouter une fonctionnalité à l'entité).
- **Action environnementale** : permet de modifier l'environnement d'exécution du service (par exemple migrer l'entité d'une machine à une autre). C'est l'action que nous exécutons pour réparer les dégradations de notre application.

5.2.4 Éxecution

C'est la dernière phase et elle est chargée d'exécuter les actions de réparation dans l'ordre prévu par la phase de planification.

5.3 Environnement d'expérimentation

L'objectif de cette expérimentation était de montrer la faisabilité de l'auto-réparation dans le Cloud Computing. Pour étudier l'intérêt de cete expérience, nous avons besoin des outils pratiques pour la mise en œuvre d'un Cloud réel.

5.3.1 OpenStack

OpenStack, fondé par Rackspace Hosting et la NASA, propose des éléments logiciels nécessaires à la réalisation d'un Cloud évolutive privé. C'est un projet open-source et libre sous la licence Apache, donc n'importe qui peut de l'exécuter ou soumettre des modifications au projet.

5.3.2 L'hyperviseur Xen

Xen est un logiciel de paravirtualisation de type hyperviseur. Il permet donc de faire tourner plusieurs SE sur une même ressource matérielle (PC, Serveur, etc.).

Le principe de l'hyperviseur est de faire tourner les SE dans le noyau (kernel) même, et non-pas de les émuler, ce qui permet de conserver des performances proches des natives. Les SE invités ont conscience du Xen sous-jacent, ils ont besoin d'être adaptés pour fonctionner sur Xen.

5.4 Configuration

La configuration de notre Cloud privé est composée de trois noeuds fonctionnant et comportant le SE Linux Ubuntu 12.04 LTS. Les deux premiers nœuds comportent XenServer qui servira la location physique des VMs. Le troisième nœud comprend l'OpenStack qui offre l'interface graphique pour gérer la couche IaaS du Cloud.

Pour des contraintes matériels, nous avons utilisé seulement deux VMs qui comprennet XenServer. En effet, nous visons à effectuer notre expérimentation à l'aide de deux serveurs. Pour ce faire, nous utilisons l'outil simple 'Oracle VM VirtualBox' sur le SE Linux afin de fournir les ressources nécessaires à nos trois serveurs.

Composant	*Configuration matérielle*			*Configuration logicielle*
Serveur Xen A	**Processeur**	**RAM**	**Disque**	
(VM Virtual Box)	2.53 GHz	512 Mo	8 Go	Ubuntu Server 12.04 LTS
Serveur Xen B	**Processeur**	**RAM**	**Disque**	
(VM Virtual Box)	2.53 GHz	1025 MO	16 Go	Ubuntu Server 12.04 LTS
	Processeur	**RAM**	**Disque**	
Management	2.53 GHz	1025 MO	50 Go	Ubuntu Server 12.04 LTS

TAB. 6 – *La configuration matérielle et logicielle*

5.5 Mise en place de l'OpenERP sur le Cloud

OpenERP (www.openerp.com) est un logiciel de gestion très modulaire et souple, développé en Python et distribué sous licence GPL. Il permet de coordonner l'ensemble des processus opérationnels d'une entreprise autour dans un même système d'information. Cette solution est caractérisée par deux principes :

- L'utilisation d'un SGBD (PostgreSQL) unique et commune entre les différents modules du logiciel ce qui implique l'élimination de saisies multiples et d'ambiguïté.
- L'utilisation d'un moteur de workflow pour propager des changement de données dans tous les modules à modifier.

L'OpenERP gére l'ensemble des processus métiers d'une entreprise à savoir la gestion des ressources humaines, la gestion comptable, financière, la production, la vente, l'approvisionnement, le commerce électronique, etc.

La mise en œuvre d'une application ERP est un peu complexe et lourde surtout côté installation et elle nécessite des ressources matérielles performantes.

5.6 Expérimentation et résultat

Après avoir présenté l'architecture d'auto-réparation adoptée, nous passons à détailler la procédure d'expérimentation que nous avons adoptée.

5.6.1 Phase de monitoring

Comme déjà mentionné dans le chapitre précédent, Zenoss est un outil de monitoring performant et conccurant dans le domaine de Cloud Computing et il est un partenaire à poids pour l'OpenSatck. Nous l'utilisons pour observer notre Cloud. Dans cette phase, nous présentons les étapes de sa mise en place et les résultats obtenus aucours de notre expérimentation.

Nous nous focalisons, dans ce cadre, sur l'observation et le stockage des paramètres de la qualité de service (QdS) des différents types d'éléments découverts par Zenoss (serveurs, images, etc.).

Zenoss récupère les données de la QdS pour contrôler le bon fonctionnement de notre infrastructure de Cloud.

L'outil Zenoss permet de mesurer la QdS : 'Charge moyenne', en anglais 'load average', qui désigne une mesure de la quantité de travail que fait le système durant une période considérée. C'est la moyenne de la charge système.

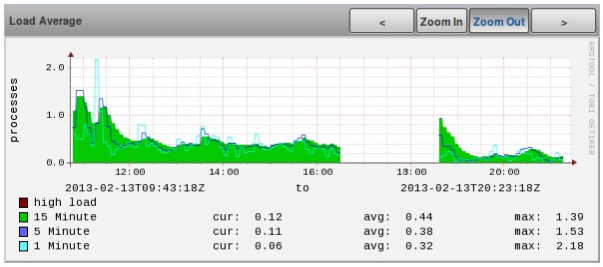


FIG. 3 – *Variation de la charge moyenne pendant 10 heures dans des conditions normales*

La figure 3 montre la variation de la charge moyenne d'un serveur durant une période de dix heures et dans des conditions normales (sans surcharge en terme de volume de traitement sur le serveur physique). Pendant cette période, nous tournons sur la VM une application 'Open ERP' [8] qui déclenche environ 77881 requêtes par 60s.

À 11 h, le système présente une surcharge de plus à 1.0, elle est dûe à l'exécution de plus de 1000 requêtes pas seconde. Entre 16h :30 et 18h :30, la charge moyenne est nulle vu que la machine est arrêtée durant deux heures.

5.6.2 Phase d'analyse

Quand une VM ou un serveur exécute plusieurs tâches à la fois, la dégradation des performances dépend de facteurs tels que le nombre de CPUs qui exécutent ces processus, la disponibilité de la mémoire et de la puissance de traitement.

Zenoss permet de détecter la dégradation des performances sur les serveurs physiques d'OpenStack. Suite à une dégradation d'une métrique de QdS, il declenche une alerte qui sera envoyée à l'administrateur soit par mail soit par pager [9]. Nous gérions l'alerte dégagée dans la phase de planification.

5.6.3 Phase de planification

La troisième fonction est celle de la planification, qui est chargée d'élaborer un plan d'actions élémentaires d'auto-réparation. Notre expérimentation vise à éviter la dégradation de la performance et les temps d'arrêt.

Le facteur déclencheur de cette dégradation est la saturation des ressources du serveur hôte (processeurs, mémoire vive, etc), sachant que le système de virtualisation consomme à lui seul de 5 à 10 % des ressources d'une machine.

8. C'est un progiciel de gestion intégré distribué sous licence libre comprenant les ventes, la gestion de relation client (CRM), la gestion de projet, la gestion d'entrepôt, la comptabilité etc.
9. C'est un petit appareil électronique qui permet de recevoir des messages numériques, texte, ou vocaux.

Pour réagir à une saturation de notre serveur, nous avons planifié une migration d'un domaine entre deux serveurs Xen sans interruption du service.

La migration est le transfert d'un domaine entre deux serveurs physiques différentes. Pour Xen, il faut satisfaire les conditions suivantes :

– La version de Xen doit être la même sur les deux serveurs.
– Les accès entre le serveur source et le destination doivent être autorisés.
– Le serveur destination doit avoir des ressources matérielles suffisantes (mémoire, disque, etc.).

Notre plan est basé sur deux étapes pour mettre en œuvre la migration de VM à savoir le transfert d'état mémoire et la redirection du traffic réseau.

✓Transfert d'état mémoire

Cette action permet de faire deplacer les paramètres et les fichiers de VM2 d'un serveur hôte A vers un emplacement de stockage sur un serveur hôte B comme le montre la figure4.

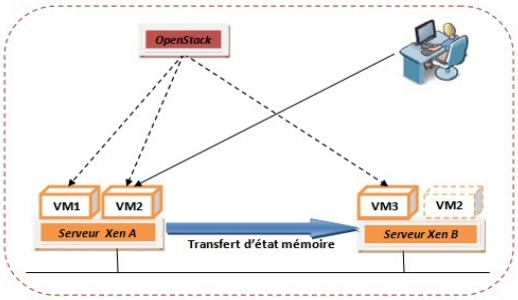


FIG. 4 – *Transfert d'état mémoire de VM2*

Une fois le transfert du fichier est effectué, la VM2 est placée dans l'état enregistré jusqu'au transfert du fichier de configuration (redirection du trafic réseau).

✓Redirection du trafic Réseau

Cette action permet d'affecter directement l'anciènne adresse publique de VM2 au serveur destination (la configuration réseau ne change pas après la migration) comme le montre la figure5.

5.6.4 Phase d'exécution

Il existe différents types de migrations, les migrations qui nécessitent l'arrêt de la machine virtuelle (Regular Migration) et les migrations ne nécessitant pas l'arrêt de la machine virtuelle (Live Migration).

Les deux serveurs doivent avoir accès à l'image disque où est installé le domaine, la version de Xen doit être la même sur les deux machines. En utilisant des scripts shell Linux, nous avons pu exécuter les commandes suivantes pour réaliser notre expérimentation.

'Regular' Migration

Migration du domaine {DomainName} vers la machine Xen {XenHost}. Lors d'une migration 'regular', le domaine est mis en pause.

'Live' Migration

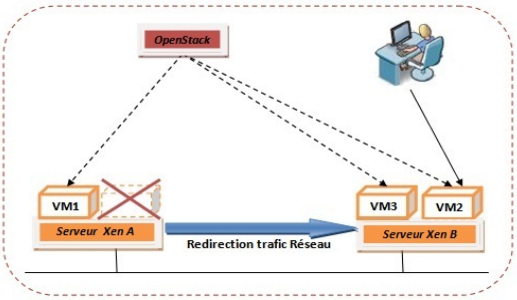


FIG. 5 – *Redirection du trafic Réseau de VM2*

```
# xm migrate \{DomainName\} \{XenHost\}
```

Le DomU n'est pas mis en pause. La durée de l'arrêt effectif est de l'ordre d'une centaine de millisecondes. Aucune connexion réseau n'est perdue lors d'une migration.

```
# xm migrate live \{DomainName\} \{XenHost\}
```

5.6.5 Execution de plan de reconfiguration

Avant de démarrer les étapes de migration de la VM entre les deux serveurs Xen, il est important de vérifier que ces dernières communiquent entre eux. Deux vérifications peuvent être réalisées :

- La première consiste à contrôler que les serveurs Xen sont présentes dans le voisinage réseau.
- L'autre vérification consiste à faire un ping sur le second serveur.

Après la vérification, nous lancons un script shell Linux, dans le serveur cible, pour exécuter deux commandes Linux. La première permet d'afficher la liste des domaines existants dans la machine cible et l'identifiant de chacun. La deuxiemme commande établit les étapes de 'Live' Migration.

5.7 *Synthèse*

D'après les données remontées par Zenoss, la charge moyenne du serveur physique cible est augmentée. Cette augmentation est très visible sur la figure 6. En effet la charge sur ce serveur devient lourde et la performance baisse.

La charge moyenne s'exprime habituellement sans unité apparente. Pour faire simple, c'est le nombre de processus entrain d'utiliser le(s) processeur(s) ou entrain d'attendre de pouvoir les utiliser. Une machine qui a une charge de plus de 1.0 par processeur (1 pour mono-processeur, 2 pour un bi-processeur, ...) est considérée comme chargée.

Pour illustrer davantage l'importance de l'auto-réparation, nous représentons l'évolution de la charge moyenne avant l'exécution de la migration dans la figure 6 et après dans la figure 7.

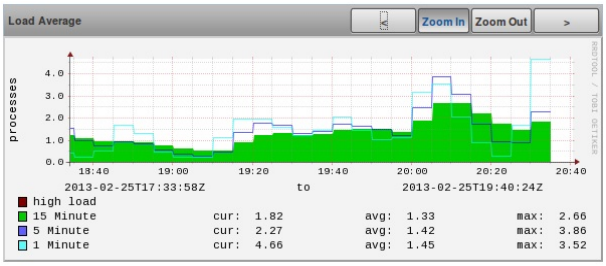


FIG. 6 – *Monitoring de serveur cible sans mise en œuvre de l'auto-réparation*

Nous reportons sur la figure 6 l'évolution de la charge moyenne de notre serveur cible en fonction du temps. La courbe tracée par Zenoss a montré l'augmentation du nombre de processus exécutés entre 19h :20 et 20h :35.

Nous constatons que la charge moyenne a évolué de 1.0 jusqu'à 3.0 pendant une heure et quelque. La raison d'une charge élevée est dûe à un manque de ressources matériels. Il peut par exemple y avoir beaucoup de processus en attente d'entrées/sorties (disque dur, CD, réseau, ...) sans que le processeur ne soit réellement bloqué.

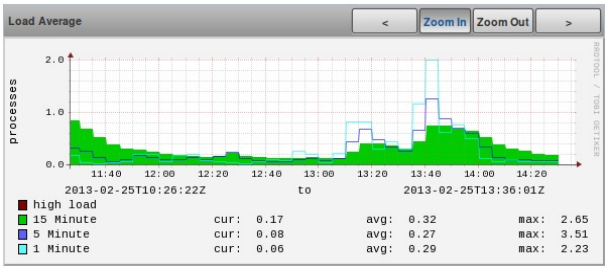


FIG. 7 – *Monitoring de serveur cible avec l'exploitation de la boucle MAPE*

Comme le montre la figure 7, la mise en œuvre d'auto-réparation n'a pas engendré d'une dégradation de la performance. On remarque que la charge moyenne de notre serveur cible ne dépasse jamais 1.0 durant deux heures d'exécution. La migration a été réalisée à 11h :20 suite à la reception d'alerte.

Nous avons remarqué au cours de l'expérimentaion, que la durée effective d'arrêt pour la migration de VM2 est comprise entre 60 et 300 ms et qu'aucune connexion réseau n'a été perdue.

Les résultats expérimentaux déjà présentés montrent bien l'utilisation de monitoring par la suite l'auto-réparation dans le Cloud. En effet, notre but était de montrer la faisabilité de la boucle MAPE en exploitant un des outils de monitoring qui est cité dans la section précédente.

6 Conclusion et perspectives

Vu la multitude des solutions de monitoring existantes, le fournisseur de Cloud se trouve vis-à-vis à des offres concurrentes pour assurer le bon fonctionnement de ses services.

Dans cet article, nous avons présenté une classification des différentes solutions de monitoring selon des critères de classifications bien définis.

Nous croyons qu'une comparaison de ces solutions peut simuler la recherche dans le domaine de monitoring de Cloud et fournit un bon point de départ pour des groupes de recherche et des fournisseurs intéressés de mieux choisir la solution la plus appropriée à leurs besoins pour le suivi des services du Cloud.

Afin de donner une vision plus claire des perspectives de cette étape et pour montrer sa faisabilité, nous avons choisi l'outil de monitoring Zenoss, et puis nous avons enchainé par les étapes restantes de la boucle MAPE sur notre Cloud privé 'OpenStack'. Notre expérimentation a montré que notre architecture d'auto-réparation réagit par des actions de migration de machine virtuelle afin d'éviter la dégradation de la performance du service Cloud.

Dans le futur, nous envisageons construire notre propre solution de monitoring de Cloud en se basant sur cette étude comparative. Ensuite, nous nous concentrons sur une expérimentation à grande échelle pour évaluer la performance de notre solution par rapport aux offres existantes.

Références

Amazon cloudwatch. http ://www.aws.amazon.com/fr/cloudwatch/.

Cacti. http ://www.cacti.net/.

Centreon. http ://www.centreon.com/.

Ganglia. http ://ganglia.sourceforge.net/.

Hp open view. http ://www8.hp.com/us/en/software/enterprise-software.html.

Logicmonitor. http ://www.logicmonitor.com/.

Munin. http ://munin-monitoring.org/.

Nagios. http ://www.nagios.org/.

Opennms. http ://www.opennms.org/.

Zabbix. http ://www.zabbix.com/.

Zenoss. http ://www.Zenoss.com/.

Aceto, G., A. Botta, W. de Donato, et A. Pescape (2012). Cloud monitoring : Definitions, issues and future directions. In *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, pp. 63–67.

Alhamazani, K., R. Ranjan, F. Rabhi, L. Wang, et K. Mitra (2012). Cloud monitoring for optimizing the qos of hosted applications. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 765–770.

Aversa, R., L. Tasquier, et S. Venticinque (2012). Management of cloud infrastructures through agents. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*, pp. 46–53.

Ben-Halima, R. (2009). *Conception, implantation et expérimentation d'une architecture en bus pour l'auto-réparation des applications distribuées à base de services web*. Français

Daubert, E., F. André, et O. Barais (2011). Adaptation multi-niveaux : l'infrastructure au service des applications. In *Conférence Française en Systèmes d'Exploitation (CFSE)*, St Malo, France.

Jean-Louis, A. et G. Olivier (2007/08). Supervision réseau. Master's thesis, Universite Claude Bernard Lyon 1.

Kephart, J. O. et D. M. Chess (2003). The vision of autonomic computing. *Computer 36*(1), 41–50.

Martinovic, G. et B. Zoric (2012). E-health framework based on autonomic cloud computing. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pp. 214–218.

Maurer, M., I. Breskovic, V. C. Emeakaroha, et I. Brandic (2011). Revealing the mape loop for the autonomic management of cloud infrastructures. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pp. 147–152.

N'tchirifou, Y. (2010). Monitoring d'une infrastructure informatique sur base d'outils libres. Master's thesis, Institut africain d'administration et d'études commerciales (IAEC) Togo.

Shao, J., H. Wei, Q. Wang, et H. Mei (2010). A runtime model based monitoring approach for cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 313–320.

Summary

Cloud Computing is a new economic approach that enables Organizations, institutions and companies to use computer resources without having to make a heavy investment in IT infrastructure. Today, they can migrate to a cloud computing model where they can rent online resources as they need. This model saves the management costs since IT resources are administered by the cloud provider. Monitoring becomes essential as it plays an important role in companies operations that provide services such as infrastructure (IaaS) , platform (PaaS) and software (SaaS) . Choosing a stable and efficient solution requires an in depth study of the different available monitoring tools in order. In this work, we conduct a survey on different existing monitoring tools, open source and shareware to guide the Cloud manager to choose the most appropriate tool to his needs. These tools perform the phases of monitoring and analysis. They can detect performance degradation.