Mise en place d'une solution de supervision : déploiement d'un système de supervision sous Check_MK pour les serveurs ateliers de Thales Underwater Systems à Sophia Antipolis

Jean-Philippe Baruteu

SeaTech
Spécialité Télécommunications

MISE EN PLACE D'UNE SOLUTION DE SUPERVISION

Rapport de stage de 3ème année rédigé par
Jean-Philippe BARUTEU

Année 2014-2015

THALES

Lu et approuvé par :

M. BOUGE Patrick
et
M. CREMONI Robert

Thales Underwater Systems

Service : **Technique Intégration et Logiciel**
Site : **Sophia-Antipolis**
Responsable : **M. CREMONI Robert**
E-Mail : **robert.cremoni@fr.thalesgroup.com**

Enseignant tuteur : Mme MINGHELLI Audrey
Statut : Enseignant chercheur

Engagement de non plagiat

Je soussigné, Jean-Philippe BARTUTEU

N° carte d'étudiant : 212034475

Déclare avoir pris connaissance de la charte des examens et notamment du paragraphe spécifique au plagiat.

Je suis pleinement conscient que le plagiat de documents ou d'une partie de document publiés sous quelques formes que ce soit (ouvrages, publications, rapports d'étudiant, internet etc…) constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour produire et écrire ce document.

Fait le 20/08/2015

Signature(s)

Document du chapitre 10 annexe 5, issu de la Charte des examens adoptée en Conseil d'Administration le 11 juillet 2013 après avis du CEVU du 27 juin 2013 - Délibération N°2013-73

Dominante du stage : système/supervision

Université de TOULON
Ecole d'Ingénieurs SeaTech
Spécialité Télécommunications
Avenue de l'Université
83130 LA GARDE
Tel 04 83 16 66 60
info.seatech@univ-tln.fr

MISE EN PLACE D'UNE SOLUTION DE SUPERVISION

Déploiement d'un système de supervision sous Check_MK pour les serveurs ateliers de Thales Underwater Systems à Sophia Antipolis

Volume 1/1

Rapport rédigé par Jean-Philippe BARUTEU
Étudiant à l'école d'ingénieurs de SeaTech
Spécialité Télécommunications
3ème année

Copies du présent document :

SeaTech : 1

Thales Underwater Systems : 1

Responsable de stage

Mme Audrey MINGHELLI
Enseignant chercheur
SeaTech
Avenue de l'Université
83130 LA GARDE

Tuteur en entreprise

M. Robert CREMONI
Responsable Activité Logiciel Support
Thales Underwater Systems
525 Route des Dolines
06903 Sophia Antipolis

REMERCIEMENTS

Je tiens, avant tout, à remercier mon tuteur en entreprise, M. Robert CREMONI ainsi que Philippe et Patricia. Ils ont été, tout au long du stage, mes principaux interlocuteurs, toujours présents pour me conseiller et m'accompagner en cas de problème. Ce sont aussi des personnes présentant d'importantes qualités humaines. Je suis heureux d'avoir passé ces six mois de travail avec eux et je garderai un excellent souvenir de ce stage.

Je tiens de plus à exprimer toute ma reconnaissance à M. Cyril BEAUMONT, responsable de l'équipe Software Architecture & Technologies au sein de Thales Underwater Systems, pour m'avoir accueilli dans son service.

Pour finir, je souhaite remercier tous les membres de l'équipe pour leur soutien, leur gentillesse et la bonne humeur qu'ils ont apportée tout au long de ce stage.

TABLE DES MATIERES

REMERCIEMENTS 5
PRESENTATION DE L'ENTREPRISE 7
PRESENTATION DU PROJET 9
INTRODUCTION 10
I. CHOIX DE LA SOLUTION 11
1. Cahier des charges 11
2. Etude des solutions 12
3. Présentation de Check_MK 14
II. INSTALLATION DE CHECK_MK 16
1. Installation de Check_MK 16
2. Supervision du serveur de supervision 19
3. Sécurisation des connexions à l'interface web 19
III. DEVELOPPEMENT ET VALIDATION 22
1. Scripts locaux 22
2. Développement de scripts locaux 23
3. Validation des exigences du cahier des charges 26
IV. DEPLOIEMENT DE LA SOLUTION 28
1. Supervision d'une machine Linux 28
2. Supervision d'une machine Windows 30
3. Déploiement 31
CONCLUSION 34
ANNEXE 1 36
ANNEXE 2 38
1. Script qui récupère l'utilisation des montages NFS 38
2. Script qui récupère l'utilisation des licences pour une application donnée 41
TABLE DES ILLUSTRATIONS 44

PRESENTATION DE L'ENTREPRISE

Thales est un grand groupe, au chiffre d'affaire de 13 milliards d'euros en 2014, spécialisé dans le développement de solutions et d'équipements. Thales emploie à ce jour près de 67000 personnes dans 56 pays. Ce groupe opère dans deux principaux domaines, celui de l'aérospatial et du transport ainsi que celui de la défense et la sécurité.

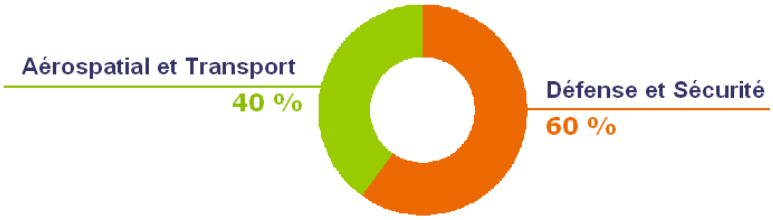


Figure 1 : Partage des deux secteurs d'activité

Le site Thales Underwater Systems de Sophia-Antipolis emploie près de 700 personnes et est spécialisé dans le développement d'équipements propres à la défense sous-marine appelés systèmes de lutte sous la mer. Thales Underwater Systems équipe près de 500 bâtiments de combat pour plus de 50 marines à travers le monde. Le groupe fournit principalement des systèmes de détection de mines sous-marines, des drones navals, des têtes acoustiques de torpilles, des systèmes pour sous-marins.

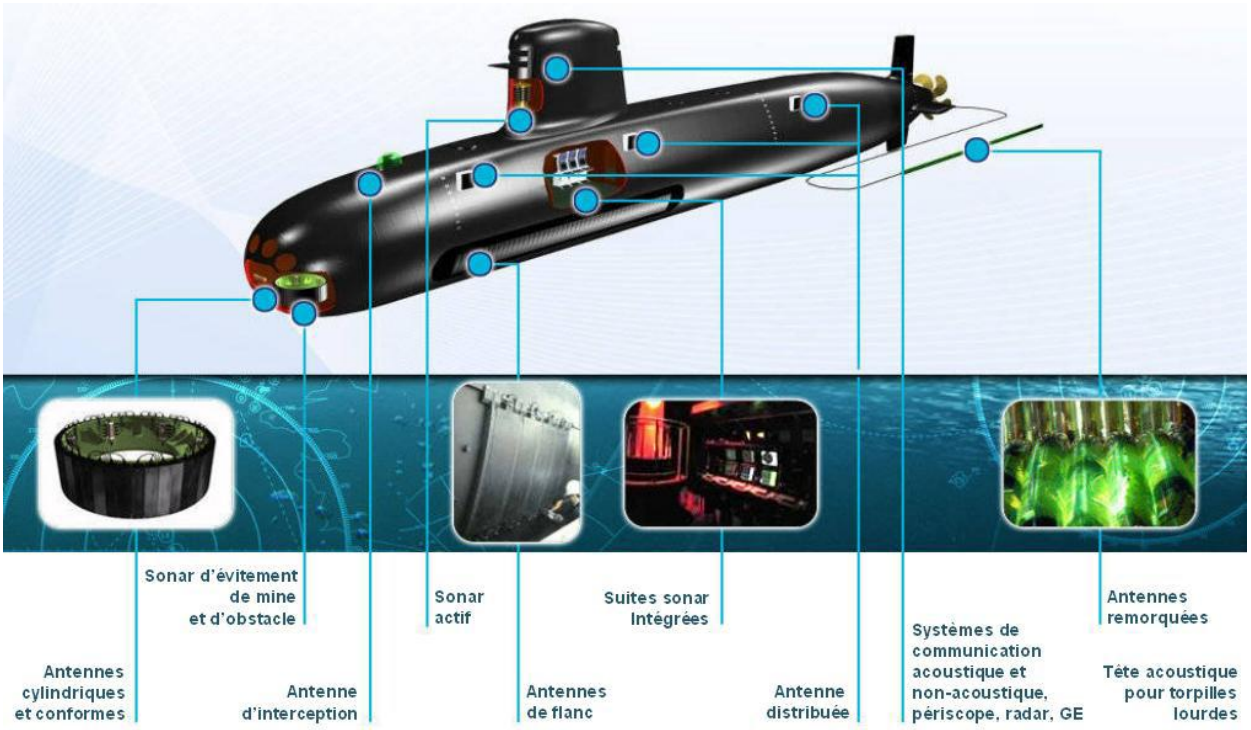


Figure 2 : Exemple de systèmes développés pour les sous-marins de combat

J'ai effectué mon stage sur le site de Thales Underwater Systems (TUS), au sein de l'équipe SAT (Software Architecture & Technologies). Cette équipe est constituée de dix personnes dont la mission principale est de répondre aux besoins des utilisateurs de TUS en terme de logiciels et machines atelier.

Cette équipe gère ainsi la partie installation et maintenance des logiciels de développement ainsi que les serveurs et machines ateliers qui supportent ces outils de développement. Ces outils sont des logiciels tels que MatLab, Eclipse ou encore Maple.

PRESENTATION DU PROJET

Au cours de ces six mois de stage chez Thales Underwater Systems, sur le site de Sophia Antipolis, j'ai été chargé de mettre en place une solution permettant à l'équipe SAT de superviser, en temps réel, l'ensemble des serveurs atelier. Ces serveurs atelier sont des machines physiques et virtuelles, qui intègrent des outils comme MatLab, Eclipse ou encore Orchestra. Ces machines permettent de mettre à disposition des employés une suite logiciels et de la puissance de calcul.

Pour répondre aux besoins de l'équipe, j'ai tout d'abord commencé par étudier différentes solutions parmi les plus utilisées dans le domaine. Le cahier des charges imposé m'a permis d'en retenir trois. J'ai ensuite présenté à l'équipe les avantages et inconvénients de chacune de ces solutions. A l'issue de cette présentation, nous avons retenu la solution de supervision Check_MK, car elle répondait le mieux aux exigences imposées.

Par la suite, j'ai pu commencer à mettre en place un prototype de la solution en installant Check_MK sur un serveur Linux dédié et en testant le fonctionnement de l'outil. Cette étape m'a permis de me familiariser avec Check_MK. J'ai poursuivi la construction de mon prototype en développant des scripts et en configurant la solution selon le cahier des charges.

Pour finir, j'ai procédé à la validation de chaque exigence du cahier des charges et à la rédaction de la documentation technique de la solution. Enfin, j'ai déployé la solution sur tous les serveurs atelier du site et j'ai formé l'équipe à l'utilisation de Check_MK.

INTRODUCTION

Le système informatique est un outil essentiel pour une entreprise, c'est pourquoi il est aujourd'hui indispensable de veiller constamment à son bon fonctionnement.
Pour cela, il existe de nombreuses solutions, appelées solutions de supervision, qui permettent de surveiller l'état des serveurs en temps réel et de remonter des alarmes en cas de problème. Ces outils permettent de faciliter le travail des administrateurs et de les rendre plus performants dans la résolution des pannes.

L'équipe Software Architecture & Technologies de Thales Underwater Systems est responsable d'une cinquantaine de serveurs. Pour superviser plus efficacement ces serveurs, des solutions de supervision avaient été mises en place, à deux reprises, par des stagiaires. Cependant, la complexité de ces solutions n'a pas permis à l'équipe de les maintenir à jour. Il n'était donc plus possible, pour les administrateurs, d'avoir un état en temps réel de tous les serveurs.

Il m'a donc été demandé, durant ces six mois de stage, de mettre en place un outil complet, simple d'utilisation et de maintenance, documenté, performant, et, permettant de superviser tous les serveurs atelier, sans altérer les performances des serveurs et la charge du réseau.

I. CHOIX DE LA SOLUTION

La première partie de mon stage a été consacrée à l'étude de différentes solutions de supervision afin de choisir la mieux adaptée aux besoins des administrateurs. Pour cela, il fallait d'abord définir un cahier des charges et vérifier la faisabilité de chacune des exigences du cahier des charges.
Une fois le cahier des charges correctement définit, je devais étudier les différentes solutions de supervision, parmi les plus utilisées, afin de définir celle répondant le mieux aux besoins de l'équipe.

1. Cahier des charges

Pour définir les besoins de l'équipe en termes de supervision des serveurs, la première étape a été d'étudier le cahier des charges et d'éclaircir chacune des exigences. Le cahier des charges se présente sous la forme suivante :

N°	Titre Exigence	Validation	Commentaires	Validateur
1	**Dashboard**			
1.1	**Les dashboard doivent remonter sur une interface Web d'une machine du VLan Xxxx**			
1.2	**Interface Web protégée par mot de passe**			
1.3	**Un bandeau (toutes pages) nombre de machines vert/orange/rouge + nombre alarmes orange/rouge + case "Recherche" (par mot clé)**			

Figure 3 : Extrait du cahier des charges

La colonne « Titre Exigence » définit l'exigence. La colonne « Validation » me permettra d'indiquer les tests que j'effectuerai pour valider l'exigence. La colonne « Validateur » me permettra d'écrire le compte rendu du test effectué pour valider l'exigence.

Le cahier des charges complet comporte 31 exigences. Il a donc été nécessaire, au début du stage, de discuter de chacune des exigences avec les administrateurs. Cette étape a permis de correctement définir chaque exigence et de parfaitement comprendre les besoins des administrateurs.
Durant cette phase, nous avons retiré deux exigences car aucun outil de supervision ne permet d'y répondre.

L'étape suivante a été de renseigner la colonne « Validation » du cahier des charges. Voici un exemple pour les trois premières exigences :

N°	Titre Exigence	Validation	Commentaires	Validateur
1	**Dashboard**			
1.1	**Les dashboard doivent remonter sur une interface Web d'une machine du VLan Xxxx**	Depuis une machine appartenant au VLan Xxxx : Ouverture d'un navigateur web (firefox 17.0.6). Connexion à l'interface web via l'URL de la page. Affichage de la page d'authentification à l'interface web.		
1.2	**Interface Web protégée par mot de passe**	Authentification à l'interface Web via utilisateur local (username/password) autorisé à s'authentifier. Test d'authentification avec comptes inexistants (vérification failles de sécurité). Vérifier, à l'aide de wireshark, si username/password passent en clair sur le réseau.		
1.3	**Un bandeau (toutes pages) nombre de machines vert/orange/rouge + nombre alarmes orange/rouge + case "Recherche" (par mot clé)**	Vérification de la présence du bandeau sur toutes les pages (avec état des machines et services). Vérifier que le bandeau affiche le bon nombre de machines et alarmes. Vérifier que le bandeau se met à jour en temps réel. Vérifier la présence de la barre de recherche et test de recherche avec mot clé valide et non valide. Vérification des résultats renvoyés.		

Figure 4 : Extrait de la colonne "Validation" du cahier des charges

Après avoir présenté, lors d'une réunion, la démarche que je suivrai pour valider chacune des exigences, j'ai pu commencer à étudier les différentes solutions parmi les plus utilisées dans le domaine de la supervision, afin de trouver celle qui répondra le mieux au cahier des charges.

2. *Etude des solutions*

Après de nombreuses recherches, j'ai pu sélectionner trois solutions de supervisions open source qui pourraient répondre aux besoins définis :

- Centreon
- Check_MK
- Zabbix

Chacune de ces solutions présente des avantages et inconvénients. J'ai donc établi un DAR (Design Analysis Report) en reprenant des points importants du cahier des charges. J'ai ensuite attribué une note de 0 à 3, associée à un coefficient, pour chaque critère :

	Utilisation/ présentation interface Web	Supervision d'une nouvelle machine	Gestions des alarmes	Affichage de graphes	Configuration des ressources à superviser	Performances	Besoins en ressources	Solution préconisée par Thales	Total
Check_MK	*Intuitive, facile d'utilisation*	*Possibilité d'utiliser un template*		*Intègre PNP4Nagios*	*Liste de checks définie par défaut en fonction du type de machine*	*Récupération de toutes les infos d'une machine en une seule requête*	*Intel Atom Z530 1.6 GH, 2Go de RAM, 21 Go d'espace disque*	*Solution utilisée et préconisée pas TGS*	
	3	2	2	1	2	1	2	1	29
Centreon	*Design ancien, très dense*	*Possibilité d'utiliser un template*		*Oui, personnalisation possible*			*2 processeurs, 2Go de RAM, 28 Go d'espace disque*		
	2	2	2	2	1	0	1	0	22
Zabbix	*Complexe d'utilisation, moins lisible*		*Alarmes complexes à définir*	*Oui*			*AMD Athlon 3200+, 2Go de RAM, 10 Go d'espace disque*		
	1	1	1	1	1	0	3	0	15
Coeff	*3*	*3*	*1*	*2*	*3*	*1*	*1*	*1*	

Figure 5 : Tableau comparatif des solutions

D'après mes recherches, c'est la solution Check_MK qui semble la plus appropriée. Néanmoins, j'ai préféré tester chacune des trois solutions.

Sur les sites dédiés à chacune des solutions, il est possible d'essayer l'outil grâce à une « live demo ». Ces « live demo » ne permettent pas de tester chacune des fonctionnalités, mais seulement d'avoir un aperçu de l'interface Web. L'interface Web est l'ensemble des pages, accessibles depuis un navigateur, qui vont permettre d'administrer l'outil et de présenter les informations récupérées sur chaque machine.

Comme je l'ai dit dans l'introduction, deux solutions de supervision avaient été mises en place par le passé. Leur complexité d'utilisation n'avait pas permis à l'équipe de maintenir à jour ces deux outils. Il me semblait donc indispensable de choisir l'outil le plus simple d'utilisation.

Après avoir testé chacune des « live demo », c'est Check_MK qui m'a paru la solution la plus appropriée. En effet, son interface Web présente un design beaucoup plus intuitif que Centreon ou Zabbix. Sa maintenance en est donc grandement facilitée.

Lors d'une réunion, j'ai ainsi présenté les avantages et inconvénients de chaque solution. J'ai également pu définir, grâce à mes recherches, de quelle manière chaque solution peut répondre à chaque exigence du cahier des charges. Pour simplifier la maintenance de la future solution, il fallait trouver l'outil permettant de répondre, nativement, au maximum d'exigences. La solution retenue a ainsi été Check_MK. J'ai alors commencé à me documenter sur la procédure d'installation de cet outil.
Durant toute la suite de mon stage, j'ai présenté l'avancement du projet, à l'équipe, lors de différentes réunions mais aussi en rédigeant un planning d'avancement hebdomadaire.

3. *Présentation de Check_MK*

Check_MK est une solution de supervision open source développée par Mathias KETTNER en 2008. Check_MK est en réalité une extension de Nagios, l'outil de supervision le plus connu et le plus utilisé en supervision système.

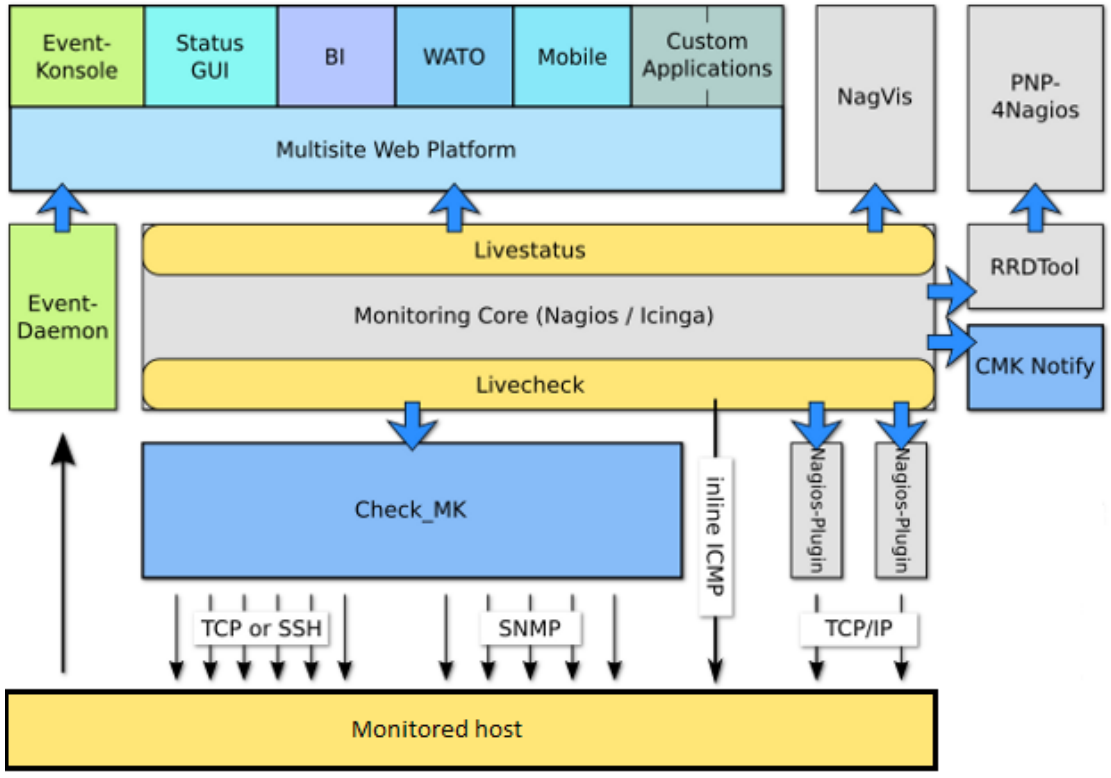


Figure 6 : Schéma de l'architecture de Check_MK

Check_MK collecte les informations sur les machines, mais utilise le cœur de Nagios pour les traiter. Cette solution intègre une interface Web beaucoup plus intuitive et des outils, comme PNP4Nagios et RRDTtool, qui permettent de réaliser des graphiques. Contrairement à Nagios,

qui se configure à partir de fichiers texte, l'interface de Check_MK permet une configuration entièrement graphique.
Une dernière particularité de Check_MK est que cette solution intègre son propre agent, installé sur chaque machine à superviser, qui récupère toutes les informations d'une machine en une seule requête. La supervision est de ce fait plus performante et nécessite moins de ressources CPU et RAM pour le serveur qui héberge Check_MK. Il est ainsi possible d'utiliser un serveur virtuel plutôt qu'un serveur physique.

Au terme de ce premier mois de stage, j'ai pris connaissance des attentes de l'équipe pour le système de supervision que je devrai déployer.

Après plusieurs réunions, toutes les exigences du cahier des charges étaient correctement définies. J'ai ainsi pu commencer mes recherches afin de sélectionner des outils de supervision répondant aux besoins. Après avoir présenté les avantages et inconvénients de chaque solution, le choix s'est porté sur Check_MK.

Je pouvais ainsi passer à l'étape suivante, l'installation de Check_MK.

II. INSTALLATION DE CHECK_MK

Dans cette deuxième partie de mon stage, je pouvais commencer à mettre en place un prototype de la future solution de supervision.
Pour cela, je devais procéder à l'installation de Check_MK et de tous les éléments qui permettraient son fonctionnement. Il fallait ensuite que je me familiarise avec l'environnement Check_MK et que je paramètre le serveur.

1. Installation de Check_MK

Check_MK se trouve sous la forme d'un package, appelé OMD (Open Monitoring Distribution). La version que je vais installer est la « omd-1.20.rhel6.x86_64.rpm », sortie en fin d'année 2014. Pour héberger Check_MK, j'ai à ma disposition un serveur virtuel, sous CentOS 6.6, qui dispose de 4 Go de RAM et d'un CPU deux cœurs.

Pour installer le package OMD, il me suffit de passer la commande « `rpm -ivh omd-1.20.rhel6.x86_64.rpm` ». Cependant, ce package a besoin d'autres paquets Linux « rmp » pour fonctionner. L'installation de ces dépendances se ferait de façon automatique en utilisant la commande « `yum omd-1.20.rhel6.x86_64.rpm` ». La commande « yum » se charge de récupérer les dépendances dans un dépôt, sur internet, et les installe. Cependant, la politique de sécurité du groupe Thales m'impose de télécharger et installer manuellement chaque dépendance.

```
[] Desktop $ rpm -ivh omd-1.20.rhel6.x86_64.rpm
error: Failed dependencies:
        fping is needed by omd-1.20-rh61-33.x86_64
        graphviz-gd is needed by omd-1.20-rh61-33.x86_64
        libmcrypt is needed by omd-1.20-rh61-33.x86_64
        mod_fcgid is needed by omd-1.20-rh61-33.x86_64
        perl-Net-SNMP is needed by omd-1.20-rh61-33.x86_64
        php-mbstring is needed by omd-1.20-rh61-33.x86_64
        xorg-x11-server-Xvfb is needed by omd-1.20-rh61-33.x86_64
        radiusclient-ng is needed by omd-1.20-rh61-33.x86_64
```

Figure 7 : Liste des dépendances à installer

Il est à noter que les dépendances installées peuvent nécessiter d'autres dépendances avant de pouvoir être installées. Il a ainsi fallu installer, au total, une trentaine de dépendances.

Le serveur est maintenant prêt pour l'installation de Check_MK.

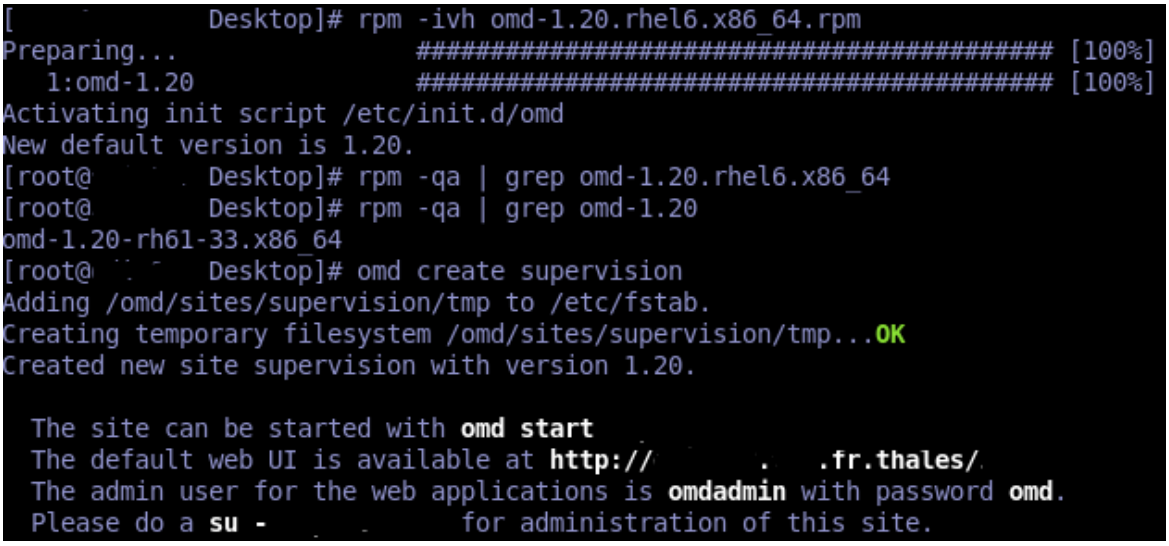


Figure 8 : Installation de Check_MK

L'installation de Check_MK s'est correctement déroulée. Nous voyons ici encore un avantage de Check_MK, son installation est très simple par rapport à Nagios ou d'autres solutions. Il faut compter une heure pour la recherche et l'installation de toutes les dépendances et du package OMD.

L'interface Web est maintenant accessible à l'adresse (masquée pour des raisons de sécurité) indiquée dans le déroulement de l'installation. Je me connecte donc à la page indiquée :

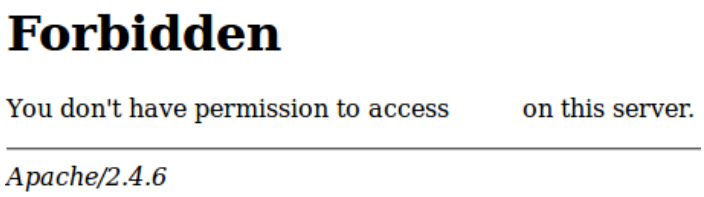


Figure 9 : Accès à l'interface Web impossible

L'accès à la page est impossible, le serveur Apache me refuse l'accès au répertoire qui contient les pages de l'interface Web. Après plusieurs heures d'investigation, il s'est révélé que le problème venait de SELinux (Security Enhanced Linux). SELinux est un système qui permet de définir une politique d'accès à chaque dossier et répertoire de Linux.

Après en avoir parlé à l'administrateur, il m'a conseillé de désactiver cette sécurité. Elle se trouve en effet inutile car les serveurs sont protégés par des pare-feu. L'interface Web était maintenant accessible :

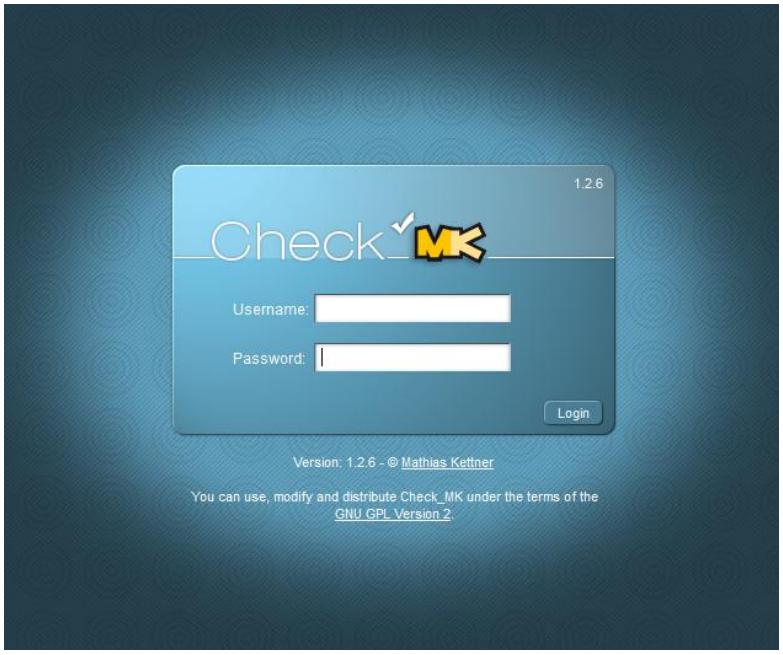


Figure 10 : Page d'authentification de Check_MK

Je pouvais ainsi commencer à me familiariser avec l'environnement de Check_MK.

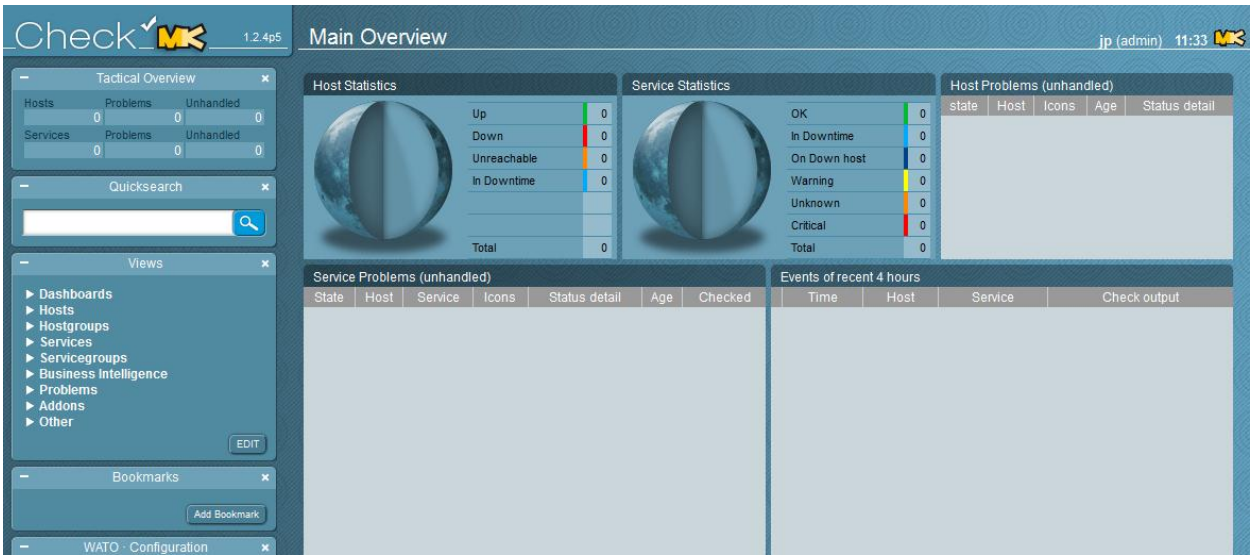


Figure 11 : Page principale de l'interface Web Check_MK

2. Supervision du serveur de supervision

State	Service	Icons	Status detail	Age	Checked	Perf-O-Meter
OK	Check_MK		OK - Agent version 1.2.4p3, execution time 1.5 sec	2015-04-14 09:46:19	2 min	1.5s
OK	Connected_User		OK - 1 utilisateur(s) actuellement connecté(s)	2015-05-04 09:36:41	2 min	
OK	CPU load		OK - 15min load 0.00 at 2 CPUs	20 hrs	2 min	0.0
OK	fs_/		OK - 51.5% used (10.59 of 20.5 GB), (levels at 80.00/90.00%), trend: +3.72MB / 24 hours	2015-04-14 09:46:20	2 min	51.55%
OK	fs_/opt		OK - 6.7% used (2.63 of 39.2 GB), (levels at 80.00/90.00%), trend: +13.47MB / 24 hours	2015-04-14 09:46:20	2 min	6.71%
OK	Interface 2		OK - [eth0] (up) MAC: 1GBit/s, in: 3.50kB/s, out: 6.08kB/s	2015-04-14 09:46:20	2 min	0.0% 0.0%
OK	Linux_Installed_Programmes		OK - job de récupération de la liste des programmes installés.	2015-05-04 09:36:41	2 min	
OK	LOG /var/log/fichier_test.log		OK - no error messages	2015-04-16 11:36:08	2 min	
OK	LOG /var/log/messages		OK - no error messages	2015-04-16 11:36:08	2 min	
OK	Memory used		OK - 0.96 GB used (0.90 GB RAM + 0.02 GB SWAP + 0.04 GB Pagetables, this is 24.8% of 3.87 GB RAM)	2015-04-14 09:46:20	2 min	23%

Figure 12 : Extrait des ressources supervisées en natif sur une machine Linux

Pour tester les fonctionnalités de configuration de Check_MK, j'ai commencé par superviser mon propre serveur de supervision. Pour cela, il suffit de définir la nouvelle machine, en lui attribuant un nom et une adresse IP, puis de lancer la découverte des services. Cette étape va recenser toutes les ressources que Check_MK sait superviser en natif.

Mon serveur de supervision est maintenant supervisé. Je récupère des informations comme l'utilisation des CPU, de la RAM, des espaces disque, etc… On remarque que Check_MK sait déjà, par défaut, récupérer beaucoup d'informations utiles.

Même si Check_MK peut être entièrement configuré en graphique, son interface Web nécessite, du fait de sa richesse, un petit temps d'adaptation. J'ai ainsi passé quelques jours pour découvrir et me familiariser avec toutes les fonctionnalités de cette interface.

3. Sécurisation des connexions à l'interface web

L'interface Web de Check_MK utilise Apache, le serveur web de Linux. C'est mon serveur de supervision qui joue le rôle de serveur web.

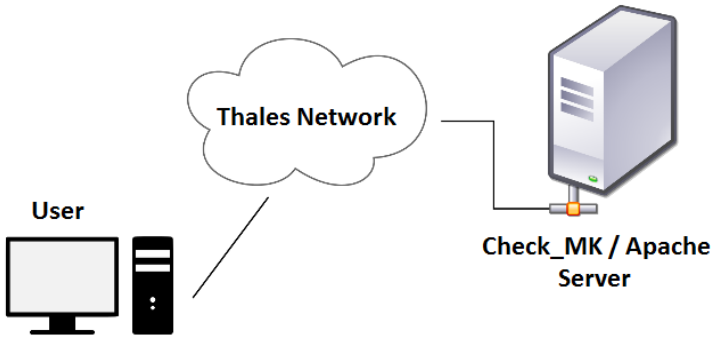


Figure 13 : Schéma de l'architecture client/serveur

Lorsque l'on souhaite se connecter à l'interface de Check_MK, via un navigateur, le serveur Apache va dialoguer en http avec le poste utilisateur. Malheureusement, en http, le dialogue client/serveur n'est pas chiffré. Toutes les requêtes passent en clair sur le réseau. Après avoir fait une analyse de trames avec Wireshark, je me suis aperçu que chaque requête, initiée par le poste utilisateur, contient les identifiants de connexions à l'interface (omdadmin : omd).

Figure 14 : Capture de trames avec identifiants en clair

Si une personne mal intentionnée venait à capturer ces identifiants, elle pourrait se connecter sur l'interface et mettre à mal tout le système de supervision.

Il fallait donc remédier à ce problème, en sécurisant les connexions via https (http secure). Pour cela, il faut activer le mode SSL du serveur Apache. Le protocole SSL utilise un système de chiffrement par clé qui va permettre de chiffrer toutes les données. Pour activer le mode SSL du serveur Apache, il suffit de créer un certificat, de modifier le fichier « /etc/httpd/conf.d/ssl.conf» et de redémarrer le service Apache. Une fois ces opérations effectuées, les données des trames transitent en https. Les identifiants ne passent donc plus en clair sur le réseau.

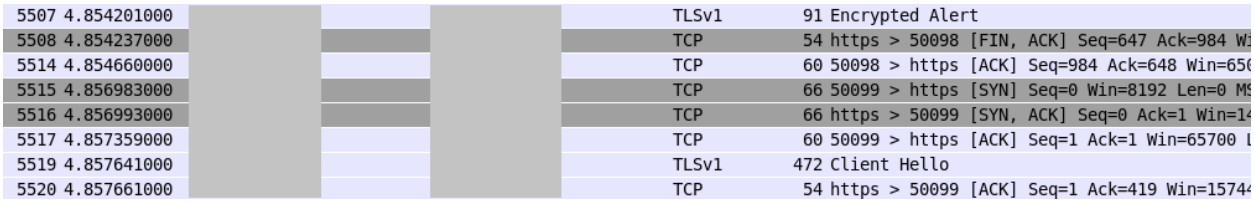


Figure 15 : Capture de trames après sécurisation

Check_MK est maintenant correctement installé et entièrement fonctionnel. Il répond aux normes de sécurité du groupe Thales.

Au terme de cette deuxième étape de mon stage, j'ai installé l'outil de supervision. J'ai été confronté à quelques problèmes, mais j'ai su les résoudre.
J'ai pu me familiariser avec toutes les fonctionnalités de Check_MK, en supervisant mon serveur et en jouant avec tous les paramètres de configuration. J'avais maintenant parfaitement l'outil en main.
J'ai pu par la suite m'apercevoir d'une faille de sécurité de Check_MK, mais j'ai pu corriger le problème en sécurisant les connexions au serveur de supervision.
Je pouvais alors passer à l'étape suivante, le développement de scripts et la validation du cahier des charges.

III. DEVELOPPEMENT ET VALIDATION

Dans cette troisième partie du stage, je pouvais commencer à répondre aux exigences plus techniques du cahier des charges. Pour cela, je devais développer des scripts permettant d'ajouter des fonctionnalités à Check_MK.
Enfin, je devais procéder aux tests et à la validation de chacune des exigences et présenter un prototype fonctionnel de la future solution de supervision.

1. Scripts locaux

Tout comme Nagios, pour récupérer des informations sur des machines, Check_MK utilise des plugins. Ces plugins sont des scripts, qui doivent se trouver sur chaque machine supervisée et qui sont exécutés en local, à intervalle de temps régulier. Check_MK se connecte ensuite sur la machine et récupère les résultats retournés par chaque plugin.

Par défaut, Check_MK propose près de 400 plugins génériques. Malheureusement, pour des applications particulières, si aucun plugin n'existe, il est nécessaire de développer ses propres plugins, appelés scripts locaux.

Ces scripts locaux, du fait qu'ils sont exécutés par la machine supervisée elle-même, peuvent être écrits dans n'importe quel langage exécutable par cette machine. Cependant, les données issues de ces scripts doivent respecter un certain format de sortie afin d'être compatibles avec Check_MK. Les arguments de sortie du script doivent ainsi se présenter sous la forme :

```
Status Script_Name Data=Perfdata;Warning_seuil;Critical_seuil; Statustxt
```

La variable « Status » indique, sur l'interface web, l'état de la ressource monitorée par le script. Si l'état est « Ok », « Status » vaut 0. Si l'état est « Warning », « Status » vaut 1. Enfin, si « Status » vaut 2, l'état de la ressource sera « Critical ». « Ok », « Warning » et « Critical » sont les trois niveaux d'alerte par ordre de criticité croissant.

Par exemple, si la charge CPU est inférieure à 90%, la ressource se trouve à l'état « Ok ». Il n'y a alors aucun problème.
Si la charge dépasse les 90%, l'état passe à « Warning ». Les administrateurs sont alors avertis d'un potentiel problème.

Enfin, si la charge dépasse les 95%, l'état devient « Critical ». Il faut intervenir tout de suite car le serveur est en surcharge anormale.

C'est donc la variable « Status » qui déclenche les alarmes sur l'interface Check_MK.

La variable « Script_Name » contient le nom du script qui sera affiché sur l'interface web.

La variable « Data » n'est pas obligatoire. Elle contient les données à afficher sur les graphiques. Ainsi, « Perfdata » contient la valeur mesurée à l'instant t par le script, « Warning_seuil » et « Critical_seuil » servent à tracer les seuils limites sur les graphiques.

Pour finir, la variable « Statustxt » contient le texte d'information qui sera affiché, sur l'interface web, en fonction de l'état de la ressource supervisée.

2. *Développement de scripts locaux*

Parmi toutes les exigences du cahier des charges, Check_MK ne peut pas, nativement, toutes les satisfaire. Ces exigences sont plus techniques et propres aux applications de Thales. Elles requièrent donc le développement de scripts locaux adaptés. J'ai développé au total une quinzaine de scripts permettant de récupérer des informations comme la vérification des mises à jour des antivirus Windows, l'état des sauvegardes des serveurs, la liste des applications installées sur chaque serveur, le nombre de licences utilisées pour certaines applications, etc…

Le développement des scripts a été une étape assez longue. J'ai en effet dû, pour chaque script, me documenter sur les différentes méthodes qui permettent de récupérer l'information souhaitée. J'ai ensuite testé plusieurs méthodes afin d'aboutir à un script performant, c'est-à-dire le plus léger possible en terme d'exécution et compatible avec toutes les versions de Linux (RedHat et CentOS), Solaris et Windows.

Les scripts que je vais devoir écrire nécessitent l'utilisation de commandes système. J'ai donc choisi de les développer en Shell sous Linux et en Bat sous Windows. De plus, ces langages restent assez simples de compréhension pour les administrateurs système. Il leur sera donc plus facile, dans l'avenir, de les maintenir à jour ou les modifier.

Pour exemple, voici un script qui permet de vérifier si l'antivirus d'une machine Windows est à jour :

```
@echo off
set datfile=C:\xxxxxxx\xxxxxxxx\file.dat
set YESTERDAY=Yesterday_Date_Calculation()

If not exist "%datfile%" (goto :NOTFILEEXIST) else (goto :FILEEXIST)
:FILEEXIST
dir /TW "%datfile%"  | find "/" > tmp.txt
call :DATE
```

```
call :HOUR
del tmp.txt
goto :COMPARAISON



:DATE
SETLOCAL
FOR /F "tokens=1-3 delims=/ " %%i in (tmp.txt) do (
      set datetmp=%%i/%%j/%%k
      set a=%%i
      set b=%%j
      set c=%%k
)
ENDLOCAL&(
set filedate=%datetmp%
set df=%a%
set mf=%b%
set yf=%c%
)

:HOUR
SETLOCAL
FOR /F "tokens=2,3 delims=: " %%i in (tmp.txt) do (
      set hourtmp=%%i:%%j
      set a=%%i
      set b=%%j
)
ENDLOCAL&(
set filehour=%hourtmp%
set hh=%a%
set min=%b%
)

:COMPARAISON
if %yf% GTR %yyyy% (goto :OKSTATUS)
if %mf% GTR %mm% (goto :OKSTATUS)
if %df% GTR %dd% (goto :OKSTATUS)

if %yf% GEQ %yyyy% (if %mf% GEQ %mm% (if %df% GEQ %dd% (goto :OKSTATUS) else goto
:WARNINGSTATUS) else goto :WARNINGSTATUS) else goto :WARNINGSTATUS

:WARNINGSTATUS
set status=1
set statustxt=L'antivirus n'a pas été mis à jour depuis le %filedate% a %filehour%
!
goto EXIT

:OKSTATUS
set status=0
set statustxt=L'antivirus est à jour. Date de dernière mise à jour : %filedate% a
%filehour%
goto EXIT

:NOTFILEEXIST
set status=2
set statustxt=Le fichier "%datfile%" est introuvable. Il est possible qu'aucun
Antivirus ne soit installé ! Si l'antivirus a récemment été mis à jour ou
remplacé, merci de modifier le chemin du fichier ".dat" dans le script
"check_antivirus_update"
```

```
goto EXIT

:EXIT
echo %status% Check_Antivirus_Update var=;;;; %statustxt%
Exit
```

Pour alléger le script, j'ai retiré les commentaires et le corps de la fonction « Yesterday_Date_ Calculation() ».

Le fonctionnement de ce script repose sur le fichier « file.dat ». Ce fichier contient la base des signatures de virus. Il est mis à jour quotidiennement.

Ce script est ainsi chargé de récupérer la date de dernière modification du fichier « file.dat » afin de la comparer avec la date du jour précédant (« YESTERDAY »).
Si le fichier « file.dat » n'a pas été mis à jour depuis la veille au moins, le script retourne une alerte « Warning » qui sera affichée sur l'interface Web de Check_MK.

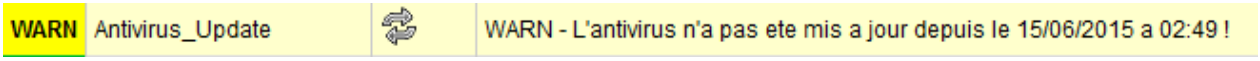


Figure 16 : L'antivirus n'est pas à jour

Si le fichier « file.dat » n'existe pas, c'est qu'aucun antivirus n'est installé. Une alerte « Critical » est retournée.

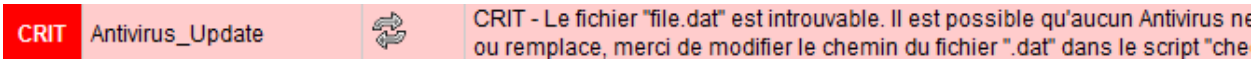


Figure 17 : L'antivirus n'est pas installé

Enfin, si l'antivirus a été mis à jour la veille ou le jour même, aucune alerte n'est remontée. La ressource apparaît en « OK » sur l'interface.

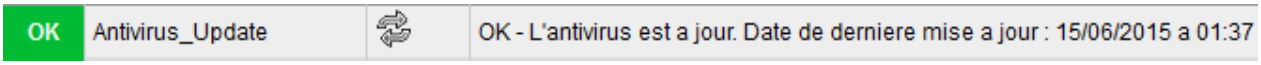


Figure 18 : L'antivirus est à jour

Le script a été testé dans tous les cas de figures. Il est parfaitement fonctionnel et s'exécute en 106*ms*. Il pourra donc être déployé sur tous les serveurs Windows.

Afin de répondre au cahier des charges, j'ai développé au total une quinzaine de scripts pour des machines Linux, Solaris et Windows ainsi que des pages php que j'ai intégrées à l'interface web de Check_MK.

3. Validation des exigences du cahier des charges

Tout au long de la phase de développement des scripts, je devais également, en parallèle, tester et valider chaque exigence du cahier des charges.

Il fallait ainsi que je valide la compatibilité de la solution de supervision pour chaque système d'exploitation présent dans le parc de serveurs. Dans ce parc, on compte quatre systèmes d'exploitation déclinés en dix-huit versions :

Windows	*XP* *Server 2003* *Server 2008* *Seven* *Eigth*
Linux RedHat	*3.3* *4.4* *4.5* *5.1* *5.4* *5.8* *5.9* *6.2*
Linux CentOS	*5.8* *6.2* *6.6*
Solaris	*9* *10*

Figure 19 : Liste des différents OS à superviser

Il était donc indispensable, avant de déployer la solution, de vérifier que l'agent de supervision installé sur les machines était bien compatible. Pour cela, j'ai installé l'agent de supervision sur des machines de test, puis j'ai testé et vérifié une quinzaine de points pour chaque version. Il fallait vérifier, pour ces quinze points, que les informations collectées par l'agent de supervision étaient bien celles que l'on pouvait observer directement sur la machine.

Les informations à vérifier étaient par exemple la quantité de mémoire totale et utilisée, la charge processeur, les espaces disque, les montages réseau, les paramètres de l'interface réseau ou encore les informations remontées par chaque script local, etc…

Je devais ensuite renseigner, dans la colonne « Validateur » du cahier des charges, la procédure que j'avais suivie pour valider l'exigence. Voici, pour exemple, un aperçu de colonne « Validateur » pour les espaces disque :

N°	Titre Exigence	Validation	Validateur
2.1.6	Espace disque (interne et externe (montages réseau))	Récupération de la ressource à l'aide du plugin dédié, pour chaque partition et montage réseau. Vérifier l'affichage de chaque ressource dans la liste des ressources par machine. Comparer les valeurs récupérées avec celles affichées sur la machine, pour chaque OS.	Nativement, Check_MK permet de récupérer l'utilisation des espaces disque et créer une ressource pour chaque espace. Pour les machines Windows, Check_MK récupère le pourcentage d'utilisation du disque, la capacité utilisée, sa capacité totale et la variation de l'espace sur les 24 dernières heures. Pour les machines linux, Check_MK récupère le pourcentage d'utilisation du disque, la capacité utilisée, sa capacité totale et la variation de l'espace sur les 24 dernières heures. Les montages réseau sont détectés, mais Check_MK n'affiche que le pourcentage d'espace utilisé, la capacité utilisée et la capacité totale. Il n'y a pas d'alertes sur les montages réseau des machines (toujours en status "OK"). Les ressources sont consultables en cliquant sur le nom de la machine. Les informations récupérées par Check_MK sont en accord avec celles observables directement sur les machines (commande df -h sous linux et gestionnaire des disques sous windows).

Figure 20 : Validation d'une exigence

Après avoir vérifié chaque exigence sur chaque système d'exploitation, j'ai pu constater quelques problèmes de compatibilité avec la version 3.3 de RedHat et Solaris 7. Sur ces machines, l'interface réseau n'est pas détectée et la quantité de mémoire vive collectée par le serveur de supervision est fausse. Néanmoins, ces OS étant assez anciens, les serveurs qui embarquent ces versions migreront bientôt vers une version plus récente de Linux. Il n'est donc pas indispensable de corriger ces problèmes de compatibilité.

Pour toutes les autres versions de Windows, Linux et Solaris, l'agent de supervision Check_MK est parfaitement compatible.

Mon prototype de la future solution de supervision était donc complet. J'ai ainsi pu le présenter à l'équipe lors d'une réunion et nous avons décidé de déployer l'outil.

Au terme de cette étape, j'ai travaillé sur le développement de scripts afin de répondre aux exigences que Check_MK ne peut pas satisfaire nativement. J'ai ainsi pu modeler Check_MK et son interface web aux besoins des administrateurs.
En parallèle, j'ai pu valider toutes les exigences du cahier des charges, pour tous les systèmes d'exploitation utilisés, hormis RedHat 3.3 et Solaris 7.
Après avoir présenté le prototype de la solution, l'équipe a décidé de mettre l'outil en production. Je pouvais ainsi passer à la dernière étape de mon stage : le déploiement de la solution.

IV. DEPLOIEMENT DE LA SOLUTION

Dans cette dernière partie de mon stage, la solution de supervision fonctionnait correctement sur les machines de test. Je devais maintenant automatiser l'installation et la configuration de l'agent afin de faciliter le déploiement sur les serveurs en production.
Pour finir, je devais procéder au déploiement de la solution.

1. Supervision d'une machine Linux

Sous Linux, l'agent Check_MK se compose de deux fichiers exécutables :

- Check_mk-agent-version.noarch.rpm
- Check_mk-agent-logwatch-version.noarch.rpm

Le premier fichier permet d'installer les plugins Check_MK. Ce sont ces plugins qui permettront de récupérer des informations génériques comme l'utilisation du CPU, de la mémoire vive, des disques durs, etc…

Le deuxième fichier permet de superviser les fichiers de logs afin de remonter des alertes au niveau du système ou des applications qui tournent sur le serveur.

Sous Linux, l'installation de l'agent se fait en ligne de commande. Etant donnée l'architecture de la solution de supervision, installer l'agent sur une machine Linux prend environ dix minutes et il est facile de faire des erreurs lors de la configuration.

Pour éviter cela, j'ai écrit un script Shell qui installe et configure l'agent de façon automatique :

```
#!/bin/bash

INSTDIR=/xxxx/xxx/xx

if (rpm -qa | grep check_mk-agent-1.2.4p3-1.noarch)
then
      echo
      echo "L'agent de supervision est déjà installé ------- WARNING"
      echo
```

```
else
        echo
        echo "## Installation du RPM check_mk-agent"
        echo
        rpm -ivh $INSTDIR/check_mk-agent-1.2.4p3-1.noarch.rpm
        echo
        echo "## Installation du RPM check_mk-agent-logwatch"
        echo
        rpm -ivh $INSTDIR/check_mk-agent-logwatch-1.2.4p3-1.noarch.rpm
        echo
        echo "## Ajout des scripts locaux"
        echo
        rm -R /usr/lib/check_mk_agent/local/
        ln -s $INSTDIR/xxxx/ /usr/lib/check_mk_agent/local
        echo
        echo "## Ajout du fichier logwatch.cfg"
        echo
        cd /etc/check_mk
        rm logwatch.cfg
        ln -s $INSTDIR/logwatch/Linux/logwatch.cfg
        echo
        echo "## Restriction d'accès à l'agent"
        cp $INSTDIR/check_mk /etc/xinetd.d/check_mk
        echo
        echo
        echo "## Agent de supervision installé -------- OK"
        echo
fi
```

Ce script commence par effectuer un test afin de vérifier si l'agent de supervision n'est pas déjà installé. Pour cela, il vérifie, via la commande « `rpm -qa` », si le paquet « `check_mk-agent-1.2.4p3-1.noarch` » est installé sur la machine.
Si l'agent n'est pas déjà installé, alors le processus d'installation démarre avec l'installation des deux paquets « rmp » via la commande « `rpm -ivh` ».
Ensuite, on fait pointer un répertoire local vers un répertoire réseau, afin d'accéder aux scripts locaux que j'ai développés, puis on ajoute le fichier de configuration pour la supervision des logs.
Pour finir, on restreint l'accès au port dédié à l'agent de supervision, afin que seul le serveur de supervision ait le droit d'interroger la machine supervisée.

Grâce à ce script, pour installer l'agent de supervision sur une machine Linux, il suffit de se placer dans le répertoire qui contient le script d'installation et de passer la commande :

```
./installation.sh
```

L'installation prend maintenant environ dix secondes, et, il ne peut plus y avoir d'erreurs de configuration.

2. Supervision d'une machine Windows

Comme pour Linux, l'installation de l'agent Windows se fait en ligne de commande et nécessite un minimum de configuration. J'ai donc également écrit un script « bat » afin d'automatiser l'installation :

```
@echo off


set DIR=x:\Temp\Check_MK

if exist %DIR% (
      goto :OUTPUT
)


echo.
echo ------------------------------------------------
echo ###### Création du répertoire %DIR%
mkdir %DIR%
echo ------------------------------------------------
echo.
echo.
echo ###### Copie des fichiers Check_MK depuis xxxxx :
echo ------------------------------------------------
xcopy /E Check_MK %DIR%
echo ------------------------------------------------
echo.
echo.
echo ###### Installation de l'agent :
echo ------------------------------------------------
%DIR%\check_mk_agent.exe install
echo ------------------------------------------------
echo.
echo.
echo ###### Démarrage du service check_mk_agent :
echo ------------------------------------------------
net start check_mk_agent
echo ------------------------------------------------
echo.
echo.
echo ------------------------------------------------
echo L'agent de supervision Check_MK est installé !
echo.
echo Pour tester : telnet 127.0.0.1 6556
echo ------------------------------------------------
echo.
:OUTPUT
echo.
echo ------------------------------------------------
echo L'agent de supervision Check_MK est déjà installe !
echo.
echo Pour tester : telnet 127.0.0.1 6556
echo.
echo **** En cas de problème, vérifier que le processus
echo "check_mk_agent" est bien démarré ****
echo ------------------------------------------------
echo.
```

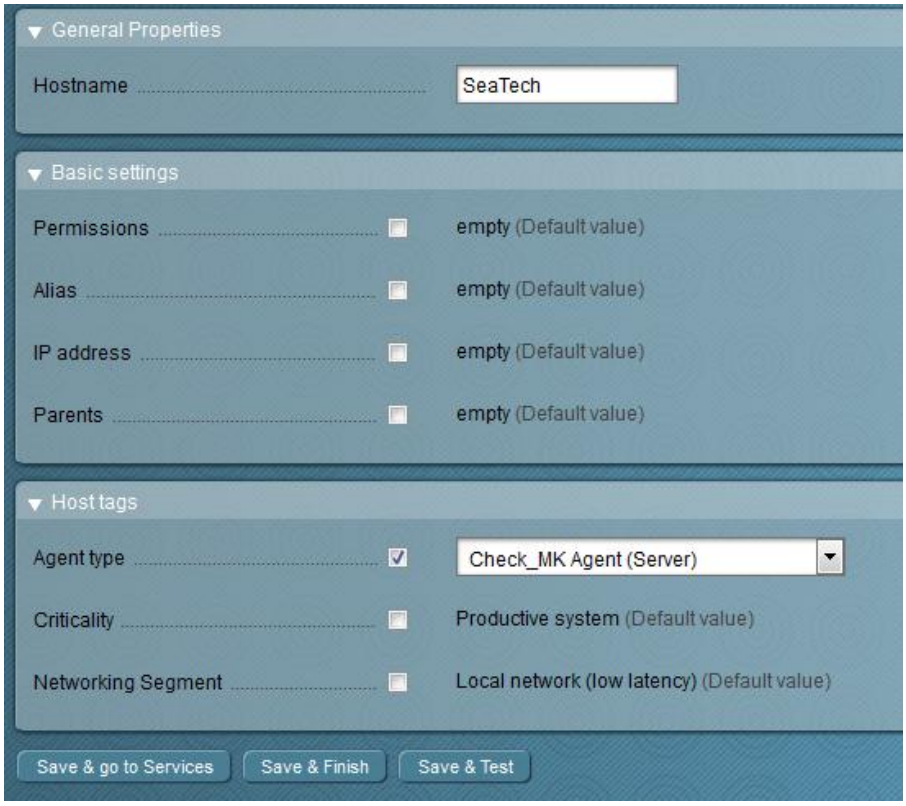
Le déroulement de ce script est semblable à celui pour les machines Linux.

Après avoir appelé ce script en passant la commande « `installation.bat` », l'installation et la configuration de l'agent se déroulent en quelques secondes.

3. *Déploiement*

La solution de supervision doit être déployée sur une cinquantaine de machines Linux, Solaris et Windows. Le parc de serveurs ne contient que quatre serveurs Solaris. Ceux-ci migreront bientôt sous Linux et Solaris sera abandonné. Je n'ai donc pas écrit de script d'installation pour cet OS. Cependant, pour les machines restantes, je vais pouvoir déployer la solution très rapidement.

Après avoir installé l'agent de supervision sur chaque serveur, via les scripts d'installation, il ne reste plus qu'à ajouter les nouvelles machines dans Check_MK. Pour cela, il faut aller sur l'interface web et créer ces nouvelles machines :

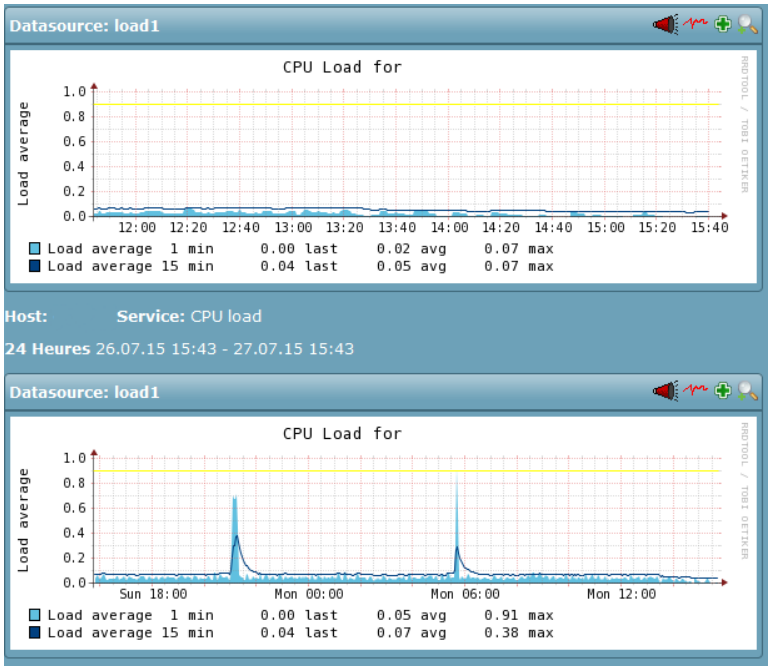


Pour créer une nouvelle machine, il suffit de renseigner son nom, connu du serveur DNS. Ensuite, après avoir cliqué sur le bouton « Save & go to Services », Check_MK découvre automatiquement toutes les ressources qu'il peut superviser sur cette machine :

State	Service	Icons	Status detail
OK	Check_Licenses_Altera		OK - Utilisation des licences Altera
WARN	Check_Licenses_Cadence		WARN - Allegro_performance (2/2), Allegro_studio (2/2), PCB_design_studio (2/2) : tous les jetons sont u pour plus d'infos)
OK	Check_Licenses_PTC-Mathcad		OK - Utilisation des licences PTC-Mathcad
OK	Check_Licenses_QAC		OK - Utilisation des licences QAC
OK	Check_Licenses_Reqtify		OK - Utilisation des licences Reqtify
OK	Check_Licenses_Scope		OK - Utilisation des licences Scope
OK	Check_Licenses_Sybase		OK - Utilisation des licences Sybase
OK	Check_Licenses_Sysnoise		OK - Utilisation des licences Sysnoise
CRIT	Check_Licenses_Telelogic		CRIT - Une ou plusieurs licences sont en erreur. Consultez le fichier de statut pour plus d'informations !
OK	Check_Licenses_TooWorks		OK - Utilisation des licences TooWorks
OK	Check_MK		OK - Agent version 1.2.6p1, execution time 7.4 sec
OK	Connected_User		OK - 2 utilisateur(s) actuellement connecté(s)
OK	CPU load		OK - 15min load 0.03 at 2 CPUs
OK	fs_/		OK - 29.1% used (2.77 of 9.5 GB), (levels at 90.00/95.00%), trend: +226.70kB / 24 hours
OK	fs_/home		OK - 10.5% used (2.36 of 22.6 GB), (levels at 90.00/95.00%), trend: +29.51kB / 24 hours
OK	Number of threads		OK - 77 threads
OK	proc_Altera		OK - 1 processes 3.6 MB virtual, 2.9 MB resident, 0.0% CPU
OK	proc_Cadence		OK - 1 processes 5.2 MB virtual, 3.3 MB resident, 0.1% CPU
OK	proc_Ideas11		OK - 1 processes 2.9 MB virtual, 2.3 MB resident, 0.0% CPU

Sur cette capture d'écran, on observe une partie des scripts que j'ai développés : les « checks » « `Check_Licenses_xxx` », « `Connected_User` » et « `proc_xxx` ».

Après avoir coché tous les « checks » intéressants, il n'y a plus qu'à valider la configuration. La nouvelle machine est maintenant supervisée et on peut observer l'état de chaque ressource et les graphiques associés :



La solution de supervision est maintenant déployée sur toutes les machines. Près de 50 serveurs et 750 ressources sont vérifiés toutes les trois minutes. Dès qu'un problème survient sur un serveur, l'équipe est maintenant prévenue dans les trois minutes.

Afin de permettre à l'équipe de maintenir à jour l'outil, c'est-à-dire pouvoir ajouter/supprimer des serveurs, ajouter des ressources à superviser, modifier la configuration de l'interface web, etc…, j'ai, tout au long du stage, rédigé une documentation de soixante pages.
Dans cette documentation, j'ai décrit l'architecture du serveur de supervision, rédigé toutes les procédures d'installation de l'outil ainsi que des tutoriels détaillant la configuration de chaque fonctionnalité disponible dans l'interface web.

Au terme de cette dernière étape de mon stage, j'ai déployé la solution de supervision sur les cinquante serveurs atelier de TUS. Afin de faciliter ce déploiement, j'ai développé deux scripts Linux et Windows permettant d'installer et configurer l'agent de supervision très simplement. Ces scripts permettront à l'équipe, par la suite, de superviser facilement de nouveaux serveurs.
Pour finir, j'ai rédigé une documentation technique qui permettra à l'équipe de maintenir à jour la solution de supervision.
Le système de supervision est ainsi parfaitement fonctionnel et intégré aux autres outils de l'équipe.

CONCLUSION

Durant ces six mois de stage, j'ai tout d'abord commencé par étudier le cahier des charges. J'ai ainsi pu sélectionner différentes solutions de supervision répondant le mieux aux exigences imposées. Cette étape m'a permis de retenir trois solutions parmi les plus connues dans le domaine de la supervision. Après avoir présenté chacune des solutions, Check_MK a été retenu.

J'ai ensuite pu débuter la construction d'un prototype de la solution de supervision. Pour cela, j'ai installé Check_MK sur un serveur virtuel dédié et je me suis servi de machines de test Linux, Solaris et Windows sous différentes versions.
Dans un premier temps, j'ai installé l'agent de supervision sur les machines de test afin de vérifier la validité des informations remontées par Check_MK (utilisation CPU, RAM, espace disque, etc…).
Une fois ceci fait pour chaque version de chaque OS, j'ai commencé à travailler sur les fonctionnalités, souhaitées par l'équipe, qui n'étaient pas disponibles en natif. Cette étape a été assez longue et a demandé de nombreuses recherches. J'ai, en effet, développé plusieurs scripts, en Bat et en Shell, permettant de récupérer en temps réel l'utilisation des licences de chaque logiciel, l'utilisation des espaces disques en réseau, le déroulement des sauvegardes, les mises à jour de l'antivirus, etc… La complexité de cette étape a été de développer des scripts Linux entièrement compatibles avec toutes les versions de Linux, de même pour Windows et Solaris. J'ai ainsi dû, pour chaque script, tester différentes méthodes afin d'aboutir à un script fonctionnant pour toutes les versions de chaque système d'exploitation. Ces scripts devaient, de plus, être les plus légers possible afin d'être exécutés rapidement par les machines supervisées, sans altérer les performances.

Enfin, mon prototype terminé, j'ai pu valider chacune des exigences du cahier des charges et déployer la solution de supervision. Pour faciliter l'installation de l'agent de supervision et la supervision de futurs serveurs, j'ai développé des scripts d'installation Linux et Windows. J'ai également rédigé une documentation technique d'une soixantaine de pages détaillant le fonctionnement et la configuration de la solution et des scripts locaux. Cette documentation permettra à l'équipe de maintenir à jour l'outil de supervision.

Au terme de ce stage, les objectifs fixés ont donc été atteints. J'ai en effet su proposer une solution de supervision complète qui répond au cahier des charges imposé.
Comme demandé, cette solution n'altère pas les performances des machines ni la bande passante du réseau.
L'outil de supervision devait être simple à maintenir à jour afin de ne pas être abandonné par l'équipe. Durant la phase de développement et configuration, j'ai toujours gardé cela à l'esprit et je pense avoir livré une solution facile d'utilisation.

ANNEXE 1

Cahier des charges complet

N°	Titre Exigence
1	**Dashboard**
1.1	**Les dashboard doivent remonter sur une interface Web d'une machine du VLan Xxxxx**
1.2	**Interface Web protégée par mot de passe**
1.3	**Un bandeau (toutes pages) nombre de machines vert/orange/rouge + nombre alarmes orange/rouge + case "Recherche" (par mot clé)**
1.4	**Une page liste des machines (Etat Vert/orange/rouge) (filtrage par machine)**
1.5	**Une page liste des alarmes (filtrage machine/état)**
1.6	**Une page détails des ressources par machine (état/valeur/seuil) pour les ressources applicables**
1.7	**Une page liste des ressources montage réseau (quotas)**
1.8	**Une page liste des sauvegardes**
1.11	**Une page consultation des historiques (1mois, 3 mois, 1 an)**
2	**Ressources à surveiller**
2.1	**Ressources à surveiller**
2.1.2	Charge CPU
2.1.3	Nombre d'utilisateurs connectés
2.1.4	Charge mémoire
2.1.5	Charge interface réseau (config port?)
2.1.6	Espace disque (interne et externe (montages réseau))
2.1.7	Etat process et des services (albd, matlab …)
2.1.8	Etat sauvegarde (date dernière)
2.1.9	Etat licences (expirée/valide (date d'expiration) - nb de jetons pris/total)
2.1.10	Alarmes Systèmes (Syslog Unix, Event Windows…)
2.1.11	Version Anti-Virus Windows (à jour?)
2.1.12	Version patchs Windows (WSUS à jour?)
2.1.13	Liste des logiciels installés (date, version)
3	**Performance**
3.1	La supervision ne devra pas entrainer de dégradations en termes de performances
4	**Réseaux**
4.1	**La supervision doit adresser les machines des domaines protégé et non protégé**

5	OS Machines
5.1	**La supervision doit adresser différents OS**
5.1.1	**Windows**
5.1.1.1	Windows Serveur 2008
5.1.1.2	Windows Seven et Eight
5.1.2	**Unix**
5.1.2.1	Redhat, toutes versions (3u3, 4u4, 4u5, 5u1, 5u4, 5u8, 6u2)
5.1.2.2	CentOS (5u8 et 6u6)
5.1.2.3	Solaris (9 et 10)

6	Couleur des Etats
6.1	**Vision globale de l'état**
6.1.1	état "vert" s'il y a absence totale de problèmes sur les ressources et les alarmes
6.1.2	état "orange" s'il y a un "warning" sur les ressources ou les alarmes
6.1.3	état "rouge" s'il y a un "critical" sur les ressources ou les alarmes

7	Configuration
7.1	**Seuils Critical/Warning à configurer par machine, par ressource**
7.2	**Template pour ajouter une nouvelle machine**

8	Historique
8.1	**Stockage des historiques (sans limite de date)**

ANNEXE 2

Quelques autres exemples de scripts locaux

1. *Script qui récupère l'utilisation des montages NFS*

L'équipe utilise un grand nombre d'espaces de stockage en réseau. Aucun outil ne leur permettait d'être prévenus si un espace arrivait à saturation. Il m'a donc été demandé de développer un script, s'adressant à un serveur Solaris, répondant à ce besoin.

```
#----------------------------------------HELP-----------------------------

# Ce script est destiné à superviser les espaces NFS d'une machine SOLARIS
# Il est inutile d'ajouter ce check à toutes les machines mais seulement à
une machine sur laquelle tous les espaces NFS sont montés.

# Pour restreindre la visibilité du check seulement à cette machine, il faut
renseigner son nom dans $HOSTNAME.

# Il est aussi possible de définir les seuils d'alerte Warning et Crticial
avec les variables $WARNING_SEUIL et $CRITICAL_SEUIL.
#-------------------------------------------------------------------------


# ----------------------------------------VARIABLES-------------------------
FILE=tmp/solaris_nfs_tmp_file
WARNING_SEUIL=80
CRITICAL_SEUIL=90
HOSTNAME=XXXXXXXXX
# -------------------------------------------------------------------------

# ----------------------------------------FUNCTIONS-------------------------
# Function to convert disk space to GB
function Conversion
{
num=$1

if [ "${num: -1}" == "T" ] # convert TB in GB
then
      num=${num%?}
      num=$(echo "$num*1024" | bc)

elif [ "${num: -1}" == "M" ] # convert MB in GB
then
      num=${num%?}
      num=$(echo "$num/1024" | bc)

else
      num=${num%?}
fi
echo $num
}
# Function to calcul alert status
function Alerte
{
usage=$1
warning_seuil=$2
critical_seuil=$3

usage=${usage%?}

if [ "$usage" -lt "$WARNING_SEUIL" ] # OK - used space < $WARNING_SEUIL
```

```
then
      status=0

elif [ "$usage" -ge "$CRITICAL_SEUIL" ] # CRITICAL - used space >
$CRITICAL_SEUIL
then
      status=2

else # WARNING - $WARNING_SEUIL <= used space <= $CRITICAL_SEUIL
      status=1
fi
echo $status
}

# Function to remove ":" in NFS mount name when it is too long (ex.
/svolccase/viewstore:)
function Remove_Last_Char
{
nfs_name=$1

if [ "${nfs_name: -1}" == ":" ]
then
      nfs_name=${nfs_name%?}
fi
echo $nfs_name
}
# ------------------------------------------------------------------------


# Check if host is $HOSTNAME
if [ $HOSTNAME != $(hostname) ]
then
      exit
fi


# Commande to retrieve NFS mount names
df -n | grep nfs > $FILE

# Reading $FILE to extract nfs mount names
while read word1 word2 word3
do
      word1=$(Remove_Last_Char $word1)
      nfs_mount="$nfs_mount $word1"

done < $FILE


nfs_mount_tab=( $nfs_mount )

# Commande to retrieve nfs mount spaces (used and total)
df -h > $FILE

# Reading $FILE to extract nfs mount spaces
while read word1 word2 word3 word4 word5 word6
do
      for line in "${nfs_mount_tab[@]}"
      do
            if [  "$word6" == "$line" ]
            then
                  total_space=$word2
                  used_space=$word3
                  percent_used=$word5
                  curent_nfs_mount="$line"

                  # Spaces converted to GB
                  total_space=$(Conversion $total_space)
                  used_space=$(Conversion $used_space)

                  # Determination of alert status
                  status=$(Alerte $percent_used $WARNING_SEUIL
      $CRITICAL_SEUIL)
```

```
                    # Calcul warning and critical levels in GB
                    warning_space=$(echo "$WARNING_SEUIL*$total_space/"100"" |
bc)
                    critical_space=$(echo "$CRITICAL_SEUIL*$total_space/"100""
| bc)

                    # Check_MK return commande
                    echo "$status NFS_$curent_nfs_mount
Usage=$used_space;$warning_space;$critical_space;;$total_space $percent_used
used ($used_space GB of $total_space GB)"
                fi
        done
done < $FILE

# Suppress $FILE
rm $FILE

exit
```

Ce script fonctionne pour Solaris. Il prend en paramètre deux seuils d'alerte, « Warning » et « Critical », fixés à 80 et 90%.
De ce fait, Check_MK émettra une alerte « Warning » si un espace de stockage NFS dépasse 80% d'utilisation :

Figure 21 : Alerte Warning remontée par le script "check_nfs_mount"

Une alerte « Critical » sera émise dépassés les 90% d'espace utilisé :

Figure 22 : Alerte Critical remontée par le script "check_nfs_mount"

Si l'espace utilisé est inférieur à 80%, la ressource sera à l'état « OK :

Figure 23 : Ressource à l'état « OK »

Le système d'exploitation Solaris tend à être remplacé par un système Linux. Ainsi, afin de permettre aux administrateurs, dans l'avenir, de superviser ces mêmes ressources sur un serveur Linux, j'ai également développé une version Linux de ce script.

2. Script qui récupère l'utilisation des licences pour une application donnée

Pour travailler sur le développement de leurs produits, les équipes Thales utilisent une trentaine d'applications. Pour des raisons de budget, ces applications sont disponibles en licences flottantes. Cela permet d'avoir un maximum de disponibilité tout en minimisant le nombre de licences à acheter. En effet, si une application dispose de 50 licences flottantes, alors tous les utilisateurs de Thales pourront potentiellement utiliser l'application, à condition que les 50 licences ne soient pas déjà en cours d'utilisation.

L'équipe SAT est chargée d'ajuster le nombre de licences par application, afin de limiter les coûts, sans impacter les utilisateurs. Cependant, l'équipe ne disposait pas d'outil leur permettant d'avoir un état de l'utilisation de chaque licence au cours du temps. J'ai donc travaillé sur un script leur permettant d'obtenir ces informations :

```
#---------------------------------------HELP-------------------------------

# Script adapté aux fichiers flexlm construits à partir de la ligne suivante :
# Users of MATLAB:  (Total of 17 licenses issued;  Total of 13 licenses in use)

# Ce script récupère l'utilisation des jetons de chaque licence pour une
# application
# "$APPLI_NAME" donnée en appelant le script "$COMMANDE" dans /etc/init.d/
# Le nom "$APPLI_NAME" de l'application n'est pas important.
# Seule la variable "$COMMANDE" doit être correctement définie.


# Le script retourne l'utilisation des licences sous forme de graphs.

# Une alerte WARNING est remontée lorsque tous les jetons d'une ou plusieurs licences
# sont utilisés et à condition que la licence dispose de plus de 1 jeton.

# Des alertes UNKNOWN sont remontées en cas de problèmes (le serveur n'est pas un
# serveur de licences, le fichier de status des licences est manquant, etc...
#--------------------------------------------------------------------------------


#-----------------------------------------CHANGE-----------------------------
# Appli to monitor
# Change this values regarding the monitored appli
APPLI_NAME="MatLab"

# Script to execute to show licenses status
# Change this values regarding the monitored (status line)
COMMANDE="xxxxxxxxxx »
DIR=( $COMMANDE )
#--------------------------------------------------------------------------------


#---------------------------------------DO NOT CHANGE------------------------
# Do not change the folowing values
SCRIPT_NAME="Check_Licenses"_"$APPLI_NAME"
HOST=$(hostname)
FILE="/xxxxxxxxx/"$APPLI_NAME"_"$HOST

# Important lines in $FILE start with "Users"
MATCH="Users"

# Flag to alert if all tokens are in use
```

```
usage_alerte=0
#-------------------------------------------------------------------------------


# Test if this server is a licenses server searching lmgrd processes :
# " ps -edf | grep lmgrd | grep flexlm | wc -l " commande return 0 if it's
not one
if [ "$(ps -edf | grep lmgrd | grep flexlm | wc -l)" -eq 0 ]
then
      exit
fi


# Status script calling to create status file
if [ -f "$DIR" ]
then
      $COMMANDE > $FILE
else
      status=3
      echo   "$status   $SCRIPT_NAME   perfdata=;;;;   Fichier   de   licences
      introuvable !"
      exit
fi


# Loop : read file to extract license name ($word1), number of licenses
($word6),
# Number of used licenses ($word11)
while read word1 word2 word3 word4 word5 word6 word7 word8 word9 word10
word11 string_reste
do
      if [ "$word1" == "$MATCH" ]
      then
            feature=$word3
            feature=${feature%?} #cut the last char of $feature (":")
            total_licenses_number=$word6
            used_licenses_number=$word11

            #detecte if all licenses tokens are in use
            if  [  $total_licenses_number  ==  $used_licenses_number  ]  &&  [
$total_licenses_number -gt 1 ]
            then
                  usage_alerte=1
                  licenses_alerte="$licenses_alerte,$feature
      ($used_licenses_number/$total_licenses_number)"
            fi

            features_list="$features_list, $feature" #list all licenses
            #Check_MK final return commande with perfdata

      return_commande="$return_commande|$feature=$used_licenses_number;;;;$to
tal_licenses_number"
      fi
done < $FILE
# End of loop


return_commande=${return_commande:1} #cut the 1st char of the return commande
(" ")
features_list=${features_list:2} #cut the two first chars of the feature list
(", ")


if [ $usage_alerte == 0 ] # OK status - licenses usage is ok
then
      status=0
      echo  "$status  $SCRIPT_NAME  $return_commande  Utilisation  des  licences
      $APPLI_NAME"
      exit
```

```
else # WARNING satus - one or more licenses are full used
      licenses_alerte=${licenses_alerte:2}
      status=1
      echo "$status $SCRIPT_NAME $return_commande $licenses_alerte : tous les
      jetons sont utilisés ! (graphes pour plus d'infos)"
      exit

fi
```

Ce script se charge d'analyser le fichier de statut correspondant à chaque application. Dans ce fichier, on retrouve le nombre total de licences ainsi que le nombre de licences utilisées, en temps réel.

J'ai donc écrit une procédure qui récupère ces informations et les retourne à Check_MK. A partir de ces informations, des graphiques sont tracés et affichent le nombre de licences utilisées en temps réel et le nombre total de licences. Ces graphiques sont consultables sur une durée allant jusqu'à 4 ans :

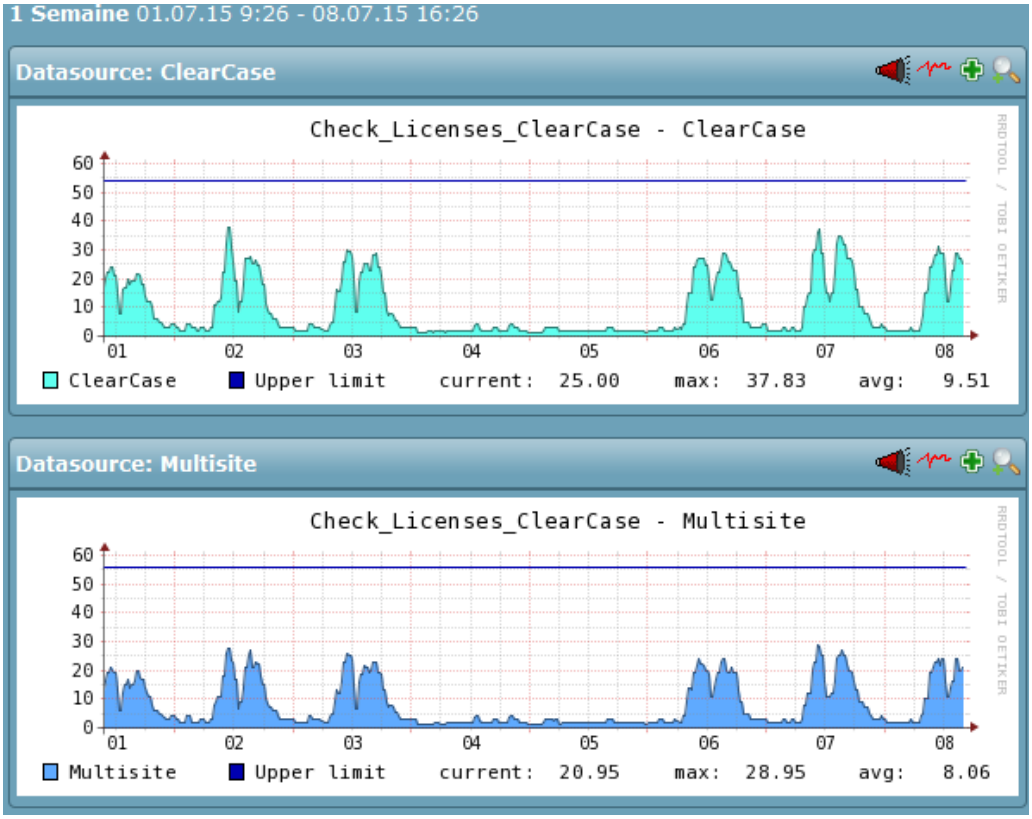


Figure 24 : Graphique d'utilisation des licences de l'application ClearCase sur une semaine

Grâce à ce nouvel outil, il sera maintenant possible pour l'équipe d'optimiser les achats de licences en fonction de l'utilisation.

TABLE DES ILLUSTRATIONS

Figure 1 : Partage des deux secteurs d'activité 7
Figure 2 : Exemple de systèmes développés pour les sous-marins de combat 8
Figure 3 : Extrait du cahier des charges 11
Figure 4 : Extrait de la colonne "Validation" du cahier des charges 12
Figure 5 : Tableau comparatif des solutions 13
Figure 6 : Schéma de l'architecture de Check_MK 14
Figure 7 : Liste des dépendances à installer 16
Figure 8 : Installation de Check_MK 17
Figure 9 : Accès à l'interface Web impossible 17
Figure 10 : Page d'authentification de Check_MK 18
Figure 11 : Page principale de l'interface Web Check_MK 18
Figure 12 : Extrait des ressources supervisées en natif sur une machine Linux 19
Figure 13 : Schéma de l'architecture client/serveur 19
Figure 14 : Capture de trames avec identifiants en clair 20
Figure 15 : Capture de trames après sécurisation 20
Figure 16 : L'antivirus n'est pas à jour 25
Figure 17 : L'antivirus n'est pas installé 25
Figure 18 : L'antivirus est à jour 25
Figure 19 : Liste des différents OS à superviser 26
Figure 20 : Validation d'une exigence 27
Figure 21 : Alerte Warning remontée par le script "check_nfs_mount" 40
Figure 22 : Alerte Critical remontée par le script "check_nfs_mount" 40
Figure 23 : Ressource à l'état « OK » 40
Figure 24 : Graphique d'utilisation des licences de l'application ClearCase sur une semaine 43