

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Remerciements

Ce travail de fin d'études éonclue-conclut mes trois années d'études en tant qu'analyste programmeur. Il va de soi qu'il n'aurait sûrement pas eu le même visage sans l'aide de plusieurs personnes.

J'aimerais donc remercier mes professeurs pour le savoir qu'ils m'ont apporté durant ces trois années pour me permettre de me lancer dans le monde professionnel. J'aimerais remercier plus particulièrement Monsieur De Fooz pour l'attention qu'il m'a portée ainsi que son soutien pour ce travail.

Je remercie Monsieur Granados et Monsieur Robin de m'avoir permis de travailler à leur côté et de m'avoir laissé la chance de faire mes preuves. Ils ont sans nul doute contribué à mon insertion dans le monde professionnel.

Je remercie aussi évidemment l'entreprise qui m'a accueilli, Limelogic. Je remercie surtout toutes les personnes au sein de l'entreprise qui m'ont appris énormément de choses et qui m'ont permis de m'épanouir dans mon futur métier. Leur patience et leur sympathie sont sans égales et je me félicite d'avoir pu travailler à leur côté pendant ces quatre mois.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Table des matières

1	INTRODUCTION	8
1.1	L'ENTREPRISE.....	8
1.2	LE CAHIER DES CHARGES.	8
1.2.1	<i>Le contexte.....</i>	8
1.2.2	<i>Les objectifs.....</i>	10
2	DESCRIPTION DES OUTILS.....	11
2.1	GIT : UN OUTIL DE GESTION DE VERSIONS	11
2.1.1	<i>Commencer à travailler avec GIT.....</i>	12
2.1.2	<i>Travailler avec GIT.....</i>	15
2.1.3	<i>Les GUI.....</i>	17
2.2	UN LANGAGE DE SCRIPT.....	18
2.3	WINDOWS POWERSHELL.....	19
2.4	LA SYNTAXE D'UNE CMDLET POWERSHELL.....	21
2.5	POWERSHELL, PERL, PYTHON ET BASH.....	21
2.5.1	<i>Perl</i>	21
2.5.2	<i>Python.....</i>	22
2.5.3	<i>Bash.....</i>	23
2.5.4	<i>Les différences et les points communs avec PowerShell.....</i>	23
3	ETUDE DE L'EXISTANT.....	25
3.1	LA PROBLÉMATIQUE	25
3.2	CHEF	26
3.3	PUPPET.....	27
3.3.1	<i>Architecture</i>	27
3.4	RUDER	30
3.5	POURQUOI UTILISER YASC ?	32
4	YASC	34
4.1	DIAGRAMME DE FONCTIONNEMENT	35
4.2	LES FICHIERS XML DE CONFIGURATION	36
4.2.1	<i>Le fichier de configuration de YASC : YascConfig.xml</i>	36
4.2.2	<i>Le fichier des métadonnées : YascMeta.XML</i>	36
4.2.3	<i>Le fichier de définition des modules PowerShell : YascEngine.xml</i>	3737
4.2.4	<i>Le fichier de déploiement d'application : AppData.xml</i>	3939
4.2.5	<i>En résumé</i>	4040
4.3	LES MODULES POWERSHELL	4141
4.3.1	<i>Règles de bonne pratique pour la programmation d'un module PowerShell.....</i>	4141
4.3.2	<i>Déclarer une cmdlet dans le fichier YascEngine.xml.....</i>	4444
4.3.3	<i>Utilisation de la cmdlet dans le fichier de déploiement de l'application.....</i>	4747
4.3.4	<i>En résumé</i>	4848

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.4	EN SORTIE DU PROCESSUS YASC.....	4949
4.4.1	<i>Schéma de fonctionnement d'un script de déploiement.....</i>	4949
4.4.2	<i>Le script de déploiement.....</i>	5050
4.4.3	<i>La documentation et la copie du contexte d'exécution</i>	5353
4.5	EN RÉSUMÉ.....	5455
5	DÉVELOPPEMENT DE MODULES DE GESTION D'UN SWITCH HP.....	5656
5.1	HP PROCURVE ET SWITCH CISCO.....	5656
5.2	CONFIGURER UN SWITCH HP PROCURVE GRÂCE À YASC.....	5757
5.2.1	<i>L'accès par SSH.....</i>	5757
5.2.2	<i>Le mode « Create ».....</i>	5858
5.2.3	<i>Le mode « Audit ».....</i>	6060
5.3	LES CMDLETS DE CONFIGURATION DU SWITCH HP PROCURVE	6262
5.3.1	<i>La configuration de 802.1X.....</i>	6262
5.3.2	<i>La configuration d'un VLAN</i>	6666
5.4	EN RÉSUMÉ.....	6868
6	DÉVELOPPEMENT DE MODULES DE GESTION D'UN FORTIGATE.....	6969
6.1	FIREWALL FORTIGATE DE CHEZ FORTINET	6969
6.2	L'ACCÈS PAR SSH	6969
6.3	LES CMDLETS DE CONFIGURATION DU FIREWALL FORTIGATE.....	7271
6.3.1	<i>Le mode de sauvegarde de la configuration</i>	7271
6.3.2	<i>La configuration d'une interface</i>	7372
6.4	LA CONFIGURATION D'UNE « POLICY »	7776
6.4.1	<i>Les objets</i>	7877
6.5	LA CONFIGURATION D'UN FILTRE WEB	7978
6.5.1	<i>La déclaration de la collection</i>	7978
6.5.2	<i>Utilisation dans le mode « Audit ».....</i>	8079
6.5.3	<i>Utilisation dans le mode « Create »</i>	8180
6.6	LA CONFIGURATION D'IPSEC	8281
6.6.1	<i>IPSec et IKE.....</i>	8281
6.6.2	<i>IPSec sur le firewall Fortigate</i>	8382
6.7	DIVERS.....	8584
6.7.1	<i>S'authentifier par LDAP.....</i>	8584
6.7.2	<i>SSL Web portal</i>	8988
6.7.3	<i>Les « traffic shapers »</i>	9089
6.8	EN RÉSUMÉ.....	9190
7	CONCLUSIONS.....	9291
7.1	CONCLUSION TECHNIQUE.....	9291
7.2	CONCLUSION PERSONNELLE.....	9292
8	ANNEXE.....	9493

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

8.1	LES PARAMÈTRES DYNAMIQUES	9493
8.1.1	<i>Structure d'une cmdlet à paramètres dynamiques.....</i>	9493
8.1.2	<i>Création de paramètres dynamiques.....</i>	9594
8.2	DÉCLARER UNE COLLECTION DE DONNÉES DANS LE FICHIER DES MÉTADONNÉES.....	9695
9	BIBLIOGRAPHIE.....	9796

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Table des figures

FIGURE 1 SCHÉMA DE FONCTIONNEMENT DÉCENTRALISÉ (GÉRER VOS CODES SOURCES AVEC GIT, 2017)	11
FIGURE 2 PAGE D'ACCUEIL DE GITHUB (GITHUB, S.D.)	13
FIGURE 3 CRÉATION DU DÉPÔT "MEMOGIT"	13
FIGURE 4 ADRESSE HTTP DU DÉPÔT GIT.	14
FIGURE 5 OUVERTURE DU BASH GIT DIRECTEMENT DANS LE DOSSIER CIBLE.	14
FIGURE 6 CLONER LE DÉPÔT GRÂCE À L'ADRESSE HTTP GitHub.....	14
FIGURE 7 FICHIER ".GIT" DANS LE DOSSIER "MEMOGIT".....	15
FIGURE 8 AJOUTER LE "TRACKING" D'UN FICHIER OU D'UN DOSSIER PAR GIT.	15
FIGURE 9 COMMIT DU FICHIER Memo02-05-2017.TXT.....	16
FIGURE 10 "PUSH" LES MODIFICATIONS VERS LE "REPOSITORY" GIT.....	16
FIGURE 11 CRÉATION DE LA BRANCHE "1STTEST".....	16
FIGURE 12 CHANGER DE BRANCHE DANS GIT.	17
FIGURE 13 AJOUT D'UN FICHIER TEXTE DANS LA NOUVELLE BRANCHE.....	17
FIGURE 14 TORTOISEGIT.....	17
FIGURE 15 CHEF ARCHITECTURE (AN OVERVIEW OF CHEF, S.D.)	27
FIGURE 16 PUPPET ARCHITECTURE (PUPPET ENTERPRISE USER'S GUIDE, 2017).....	29
FIGURE 17 RUDDER ARCHITECTURE (RUDDER 4.0 - USER'S MANUAL, 2016).....	31
FIGURE 18 YASC ARCHITECTURE (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	35
FIGURE 19 SCHÉMA DE FONCTIONNEMENT D'UN SCRIPT DE DÉPLOIEMENT (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	4949
FIGURE 20 EN-TÊTE "PARAM" D'UN SCRIPT DE DÉPLOIEMENT GÉNÉRÉ PAR YASC DANS LE CADRE DE LA CONFIGURATION D'UN FIREWALL.	5050
FIGURE 21 DOCUMENTATION D'UN DÉPLOIEMENT À UNE ÉTAPE.....	5454
FIGURE 22 ENTRÉE POUR UN UTILISATEUR "MANAGER" DANS LE KEEPASS.	5959
FIGURE 23 SCHÉMA D'AUTHENTIFICATION EAP SUR LAN (EAP - WIKIPEDIA, 2017)	6262
FIGURE 24 CONFIGURATION D'UN MODÈLE AAA. (CONFIGURING 802.1X PORT-BASE AUTHENTICATION, S.D.)	6565
FIGURE 25 CONFIGURER LES INFORMATIONS DU SERVEUR RADIUS SUR UN SWICH CISCO. (CONFIGURING 802.1X PORT-BASE AUTHENTICATION, S.D.).....	6565
FIGURE 26 ARBORESCENCE D'UN SERVICE D'ANNUAIRE.	8584
FIGURE 27 CONFIGURATION D'UN SERVEUR LDAP SUR LE FORTIGATE.	8685
FIGURE 28 CRÉER UN GROUPE D'UTILISATEUR LIÉ À UN GROUPE SUR LE SERVEUR LDAP.	8786
FIGURE 29 CONFIGURATION DES PARAMÈTRES DU VPN SSL.....	8887
FIGURE 30 CONFIGURATION DU MODE TUNNEL ET DU MODE WEB AVEC DU "SPLIT-TUNNELING".....	8887
FIGURE 31 CRÉER DES BOOKMARKS, TÉLÉCHARGER FORTICLIENT, ETC ... (SSL VPN USING WEB AND TUNNEL MODE - FORTINET COOKBOOK, 2015)	8988
FIGURE 32 LE PORTAIL WEB PERMET D'ACCÉDER AUX RESSOURCES DU RÉSEAU VIA PLUSIEURS PROTOCOLES DIFFÉRENTS. IL PERMET AUSSI L'ACCÈS À DES PAGES WEB OU DES SERVEURS FTP ET TOUT CELA DE FAÇON TOTALEMENT SÉCURISÉE (SSL VPN USING WEB AND TUNNEL MODE - FORTINET COOKBOOK, 2015)	8988
FIGURE 33 SHAPERS CRÉÉS POUR LIMITER LA BANDE PASSANTE SELON TROIS NIVEAUX. "ACCÈS LIMITÉ", "ACCÈS MEDIUM" ET "ACCÈS PRIVILÉGIÉ"	9089

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

EXEMPLE 1 CRÉATION D'UN TYPE POWERSHELL	19
EXEMPLE 2 CRÉATION D'UNE FONCTION PERMETTANT DE CHANGER LA VALEUR D'UNE VARIABLE DE L'OBJET CRÉER PRÉCÉDEMMENT	20
EXEMPLE 3 INSTANCIATION DU TYPE ET PIPE VERS LA FONCTION	20
EXEMPLE 4 EXEMPLE SYNTAXE DU PERL (PERL (LANGAGE) - WIKIPEDIA, 2017)	22
EXEMPLE 5 EXEMPLE DE PYTHON (PYTHON (LANGAGE) - WIKIPEDIA, 2017)	22
EXEMPLE 6 CMDLET PERMETTANT DE CHANGER LE MOT DE PASSE ADMIN D'UN A.D.	3838
EXEMPLE 7 YASC IRA CHERCHER LA VALEUR DANS LE FICHIER DES MÉTADONNÉES	3939
EXEMPLE 8 TAG "GENERAL"	3939
EXEMPLE 9 LES ÉTAPES DE DÉPLOIEMENT D'UN FIREWALL	4040
EXEMPLE 10 EXEMPLE D'EN-TÊTE DE DESCRIPTION	4141
EXEMPLE 11 EXEMPLE DE NOMINATION DES PARAMÈTRES (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	4242
EXEMPLE 12 SYNTAXE DU PARAMÈTRE "Mode" DE YASC (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	4242
EXEMPLE 13 CODE DE DÉMARRAGE D'UNE CMDLET (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	4242
EXEMPLE 14 OBJET "Status" (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)	4343
EXEMPLE 15 LOGGER UNE INFO DANS UNE CMDLET (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016).....	4444
EXEMPLE 16 CMDLET DÉCLARÉE DANS LE FICHIER YASCEngine.XML.....	4444
EXEMPLE 17 DÉCLARATION D'UNE COLLECTION DANS YASCEngine.XML	4545
EXEMPLE 18 DÉCLARATION D'UNE COLLECTION DANS LE FICHIER YascMeta.XML.....	4646
EXEMPLE 19 DÉCLARATION DE LA COLLECTION DANS LE FICHIER DE DÉPLOIEMENT "AppData.XML"	4646
EXEMPLE 20 CONFIGURATION D'UN VLAN SUR UN SWITCH HP PROCURVE.....	4747
EXEMPLE 21 CHARGEMENT DU FICHIER YascConfig.XML ET CRÉATION DE L'OBJET YRT	5252
EXEMPLE 22 ENVOI D'UN EMAIL À LA FIN DU SCRIPT DE DÉPLOIEMENT	5252
EXEMPLE 23 OBJET CONTENANT SOUS FORME TEXTUELLE LA DOCUMENTATION.....	5353
EXEMPLE 24 CRÉATION D'UNE SESSION SSH ET ENVOI D'UNE COMMANDE.....	5757
EXEMPLE 25 CRÉATION D'UNE SESSION SSH AVEC UN FLUX SHELLSTREAM	5858
EXEMPLE 26 PREMIÈRE ÉTAPE : APPELER LE MODE "AUDIT" DE LA CMDLET POUR VÉRIFIER SI LA CONFIGURATION N'EST PAS DÉJÀ APPLIQUÉE.....	5959
EXEMPLE 27 MODE "CREATE" DE LA CMDLET PERMETTANT DE CRÉER UN VLAN ET DE CONFIGURER SON NOM.....	6060
EXEMPLE 28 MODE "AUDIT" DE LA CMDLET PERMETTANT DE CRÉER UN VLAN ET DE CONFIGURER SON NOM.	6161
EXEMPLE 29 VÉRIFIER SI LE NOM DU VLAN "X" A BIEN ÉTÉ CONFIGURÉ AVEC LE NOM SOUHAITÉ.....	6161
EXEMPLE 30 CONFIGURER LES INFORMATIONS SUR LE SERVEUR RADIUS	6363
EXEMPLE 31 CONFIGURER LES INFORMATIONS D'AUTHENTIFICATION AAA.....	6464
EXEMPLE 32 CONFIGURATION DES "AUTHENTICATOR PORTS" ET DES VLANS POUR LES CLIENTS AUTORISÉS ET NON AUTORISÉS. ...	6464
EXEMPLE 33 COMMANDE PERMETTANT DE CONFIGURER UN PORT DANS UN MODE QUELCONQUE POUR UN VLAN DONNÉ.....	6767
EXEMPLE 34 CRÉER UNE SESSION SSH VERS LE FORTIGATE ET RÉCUPÉRER LA CONFIGURATION POUR UNE ADRESSE.	7070
EXEMPLE 35 CONFIGURER UNE ADRESSE SUR LE FORTIGATE VIA LA CMDLET "Invoke-SSHCommand".	7170
EXEMPLE 36 VÉRIFIE LE MODE DE SAUVEGARDE D'UN FIREWALL.	7271
EXEMPLE 37 VÉRIFIER LES MÉTHODES D'ACCÈS D'UNE INTERFACE.	7473

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

EXEMPLE 38 CRÉER UNE COMMANDE CLI SUR BASE D'UNE COLLECTION DE DONNÉES	7473
EXEMPLE 39 DÉCLARATION DE L'ÉTAPE DANS LE FICHIER DE L'ENGINE	7473
EXEMPLE 40 AJOUTER À LA DÉCLARATION UN PARAMÈTRE (DANS CE CAS-CI, UNE COLLECTION DE DONNÉES)	7574
EXEMPLE 41 DONNER LES VALEURS ATTENDUES À LA COLLECTION DE DONNÉES	7574
EXEMPLE 42 VÉRIFIER UN ÉVENTUEL CHEVAUCHEMENT D'ADRESSES IP VIA LA CMDLET "TEST-LDPYASCFORIGIPPEXISTS"	7675
EXEMPLE 43 DANS LE CAS OÙ UN CHEVAUCHEMENT A ÉTÉ DÉTECTER, ON N'AJOUTE PAS LA COMMANDE DE CONFIGURATION DE L'ADRESSE IP À LA COMMANDE GLOBALE	7675
EXEMPLE 44 CONSTRUCTION DU TABLEAU CONTENANT LES CATÉGORIES ET LEUR POLITIQUE D'ACCÈS	8079
EXEMPLE 45 VÉRIFIER SI UNE CATÉGORIE N'A PAS ÉTÉ SUPPRIMÉE PAR RAPPORT À LA CONFIGURATION SOUHAITÉE	8079
EXEMPLE 46 VÉRIFIER SI UNE CATÉGORIE N'A PAS ÉTÉ AJOUTER PAR RAPPORT À LA CONFIGURATION SOUHAITÉE	8180
EXEMPLE 47 AJOUTER À LA COMMANDE CONSOLE À ENVOYER AU FIREWALL, LA CONFIGURATION DES CATÉGORIES ET LEUR POLITIQUE D'ACCÈS	8180
EXEMPLE 48 STRUCTURE D'UNE CMDLET À UNE FONCTION	9493
EXEMPLE 49 CMDLET UTILISANT DES PARAMÈTRES DYNAMIQUES	9493
EXEMPLE 50 CRÉATION D'UN PARAMÈTRE DYNAMIQUE	9594
EXEMPLE 51 UTILISATION DU PARAMÈTRE DYNAMIQUE DANS LA PORTÉE "PROCESS"	9594
EXEMPLE 52 APPEL D'UNE CMDLET AVEC DES PARAMÈTRES DYNAMIQUES	9594
EXEMPLE 53 DÉCLARATION D'UNE COLLECTION DE DONNÉES DANS LE FICHIER XML DES MÉTADONNÉES : YASCMETA.XML	9695

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

1 Introduction.

1.1 L'entreprise.

Limelogic est une société d'infogérance créée en [2006-2004](#) par Jean-Christophe Robin et [Éric Eric](#) Granados. Son siège social se situe à Liège, Rue Ernest Solvay.

Limelogic offre une solution de gestion informatique assez innovante à ses clients. En effet, la création de Limelogic elle-même se base sur le constat qu'un dirigeant de PME ne dispose pas toujours du temps ou des compétences nécessaires pour gérer son infrastructure réseau. Le plus souvent, ce même patron décide de déléguer la gestion informatique à une entreprise spécialisée. Cependant, ces entreprises ne tiennent pas toujours leurs promesses ou ne prennent pas leurs responsabilités et chaque intervention est facturée.

Limelogic offre une solution innovante et élégante qui consiste en un forfait invariable, peu importe la quantité d'interventions à réaliser pour régler une panne matérielle ou même logicielle. La société s'impose des objectifs de résultats, fournit le matériel nécessaire pour la gestion du réseau, et tout ça, à des prix fixes et [maîtrisésmaîtrisés](#).

La plupart des clients de Limelogic sont des PME qui ne possèdent pas une équipe informatique pour gérer leur réseau : travailler avec Limelogic revient à engager une équipe d'informaticiens fournissant un service de support informatique 24 heures sur 24 et 7 jours sur 7.

En cas de panne, qu'elle arrive le week-end ou en semaine, une notification plus ou moins grave est envoyée aux techniciens qui prennent alors en charge le problème, peu importe le moment ou l'heure.

L'équipe compte actuellement une bonne dizaine de membres et souhaite encore s'agrandir à l'avenir.

1.2 Le cahier des charges.

1.2.1 Le contexte

Limelogic a développé un outil de déploiement et configuration automatisés, entièrement basé sur PowerShell. Ce système peut de façon résumée se définir en trois composants :

- I) Un ensemble de « cmdlets » (ou fonctions) PowerShell, chacune effectuant une opération atomique [sus-sur](#) le système
- II) Un « engine » qui est alimenté par trois inputs :
 - Un fichier XML de métadonnées, décrivant les constantes à appliquer à un environnement ;

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

- Un fichier XML « engine », décrivant le mode d'utilisation de chaque cmdlet ;
- Des fichiers XML « déploiement », chacun décrivant un déploiement particulier que l'on souhaite effectuer.

III) Les résultats d'une exécution de l'engine, qui sont :

- Un script PowerShell, dont l'exécution réalisera le déploiement réel ;
- Une documentation technique HTML, générée automatiquement, et décrivant ainsi sous une forme lisible le déploiement.

On pourrait assimiler en quelque sorte cet outil à des produits existants tels que Chef, Puppet, Rudder, voire le « Desired State Configuration » initié par Microsoft depuis Windows Server 2012 R2, avec cependant trois différences notables :

- La génération automatique d'une documentation,
- La génération d'un script de déploiement **à usage unique**, ne contenant aucun paramètre mais les valeurs de données qui seront réellement utilisées pour exécuter le déploiement, ce qui rend leur lecture relativement aisée,
- La présence d'un « audit mode », qui ne réalise pas de déploiement mais compare un déploiement déjà réalisé à ce qui était prévu, ceci afin de déceler une dérive entre le déploiement souhaité et la situation réelle (interventions manuelles par exemple).

Au départ, cet outil a été conçu dans le cadre d'un projet de migration et sécurisation d'applications Web IIS de Windows 2003 vers Windows 2008R2. De ce fait, les cmdlets qui ont été implémentées concernent quelques domaines bien précis, tels que : création de users Active Directory ; création de groupes de sécurité ; octroi de droits NTFS via ACL ; octroi de droits sur des bases de données Sql Server ; déploiement d'applications ; configuration de IIS (application pools, applications, ...).

Le besoin de Limelogic est d'étendre ce système pour implémenter un ensemble de cmdlets qui à terme pourront gérer tout le cycle de déploiement d'une infrastructure. Cela inclut notamment :

- Le déploiement d'un OS et sa configuration de base ;
- Le déploiement de machines virtuelles ;
- La configuration complète d'un Active Directory (création d'un domaine ; mise en place de services Dns, Dhcp, File, Print ; création de GPOs ; etc) ;
- La configuration d'équipements réseau (switches ; firewalls ; points d'accès Wifi ; etc).

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

A ce jour, Limelogic a également développé tout un ensemble de scripts PowerShell qui gèrent complètement ou partiellement ces problématiques ; mais ces scripts manquent de structure et de cohérence, ils n'implémentent pas de fonction « audit », etc.

1.2.2 Les objectifs

Les objectifs sont :

- **Principalement** : développer un ensemble de cmdlets conformes à l'outil de déploiement, soit à partir de zéro, soit en s'inspirant des scripts existants ;
- **En second lieu** : participer à la réflexion sur des améliorations possibles de « l'engine » en lui-même.

Ce travail sera composé de multiples objectifs tiers qui dépendront des étapes traversées durant le développement. Il sera par exemple demandé d'appréhender les différentes technologies et les différents matériels présents dans une infrastructure et d'apprendre à les utiliser. Infrastructure sur laquelle l'outil de déploiement devra justement travailler.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2 Description des outils.

YASC est un engine totalement écrit en PowerShell tout comme les modules qu'il utilise pour fonctionner. Le PowerShell est un langage de script natif dans Windows depuis Windows 7. C'est donc pour cela qu'une introduction au PowerShell, de son historique et des langages de script en général semble évidente.

De plus, ce travail de fin d'études concernant un projet de développement, il est relativement important de débuter avec un chapitre décrivant un outil indispensable dans ce domaine : GIT.

2.1 GIT : un outil de gestion de versions

GIT est un logiciel de gestion de versions décentralisées très utilisé en développement informatique. Il permet de gérer les différentes versions d'un programme/d'un projet et de les sauvegarder de manière indépendante les unes par rapport aux autres. GIT est un logiciel libre développé par Linus Tavarlids, le créateur du noyau Linux.¹

GIT est donc un logiciel décentralisé. C'est-à-dire qu'il n'y a pas de serveur source, chaque développeur possède un historique de l'évolution des fichiers et partage les modifications du projet en « peer-to-peer ». Cependant, pour des raisons pratiques on utilise tout de même des serveurs.

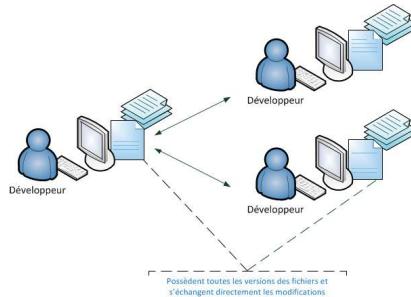


Figure 1 Schéma de fonctionnement décentralisé (Gérer vos codes sources avec GIT, 2017).

¹ Les premières prises en mains de GIT sont peu évidentes. Pour bien comprendre et appliquer les commandes GIT : <https://try.github.io/levels/1/challenges/1>. Ce site permet de simuler une invite de commande GIT. Malgré son aspect basique, elle permet de s'essayer aux branches et autres fonctionnalités GIT déjà bien pratique pour débuter.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Il existe plusieurs logiciels de gestion de versions, cependant GIT se démarque des autres par sa rapidité et sa gestion des branches permettant d'implémenter en parallèle de nouvelles fonctionnalités pour un programme en développement.

2.1.1 Commencer à travailler avec GIT

Lorsque l'on commence à travailler avec GIT il est possible de créer un projet ou de reprendre un projet en cours (par exemple lorsque l'on souhaite travailler sur une nouvelle machine). Dans les deux cas, il faudra utiliser ce que l'on appelle un « repository » ou un dépôt, en français.

Un dépôt représente une copie du projet à un instant donné. Chaque ordinateur travaillant sur ce projet possède une copie du dépôt. Le dépôt contient, en plus des fichiers du projet, l'historique de celui-ci.

On peut donc soit créer un nouveau dépôt soit « cloner » un dépôt. Lorsque l'on clone un dépôt on va télécharger une copie du dépôt sur son ordinateur, c'est-à-dire une copie du projet et de tout son historique. Lorsque l'on crée un nouveau dépôt celui-ci est vide et est prêt à être utilisé pour un nouveau projet.

Voici les commandes, sous un bash GIT, à taper pour créer un dépôt ou pour le cloner :

Créer un dépôt	<code>git init</code>
Cloner un dépôt	<code>git clone adresse web du dépôt</code>

Pour créer un dépôt c'est très simple, il suffit de créer un dossier, de se placer à l'intérieur, et de taper la commande « git init ». Cependant, cloner un dépôt nécessite d'avoir... un dépôt ! Pour cela, il faut l'adresse de celui-ci. Plusieurs plateformes permettent de créer des dépôts pour des projets, comme GitHub ou GitLab. Depuis ces interfaces web, il est possible de récupérer l'adresse web du dépôt, elles sont généralement affichées sur la page d'accueil du projet. Il suffit alors de taper la commande « git clone » suivie de l'adresse http.

Pour les curieux, il est bon de savoir qu'il est possible de récupérer l'adresse de dépôt git de certains programmes connus et libres pour analyser le code source.

Pour illustrer les manipulations faites avec GIT prenons un exemple : nous allons créer sur GitHub un dépôt qui sera censé suivre un ensemble de mémos écrits sous forme de fichier texte. Rien d'exceptionnel, mais ce sera suffisant pour comprendre les concepts de GIT.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Premièrement, il faut se créer un compte sur GitHub. Une fois cela fait, on peut trouver sur la page d'accueil directement de quoi créer un dépôt :

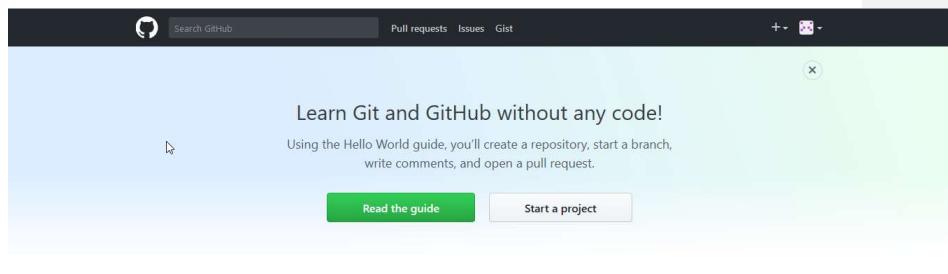


Figure 2 Page d'accueil de GitHub (GitHub, s.d.).

Il suffit alors de cliquer sur « Start a project » et de lui donner un nom, par exemple, « MemoGIT ».

Create a new repository

A repository contains all the files for your project, including the revision history.

A screenshot of the GitHub 'Create a new repository' form. The form includes fields for 'Owner' (set to 'CerfontaineMathieu'), 'Repository name' (set to 'MemoGIT'), and a 'Description (optional)' field containing 'MemoGIT'. There are two radio button options for visibility: 'Public' (selected) and 'Private'. A checkbox for 'Initialize this repository with a README' is checked, with a note below stating 'This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.' At the bottom, there are buttons for 'Add .gitignore: None ▾', 'Add a license: None ▾', and a large green 'Create repository' button.

Figure 3 Création du dépôt "MemoGIT".

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Une fois le dépôt créé, on est redirigé vers la page d'accueil de celui-ci. C'est là que l'on trouve l'adresse du dépôt à utiliser pour le cloner.



Figure 4 Adresse HTTP du dépôt GIT.

Il suffit alors de copier-coller l'adresse et de créer un dossier quelconque sur l'ordinateur. Prenons par exemple le dossier « MemoGIT » que nous placerons à la racine de la partition Windows « C:\ ». Si GIT est installé, lorsque l'on réalise un clic droit au sein d'un dossier. On a alors la possibilité d'ouvrir un bash GIT. Cela va nous permettre de taper les commandes pour cloner notre dépôt (il est aussi possible d'utiliser le GUI mais il est conseillé de réaliser au moins une fois les opérations via les commandes).

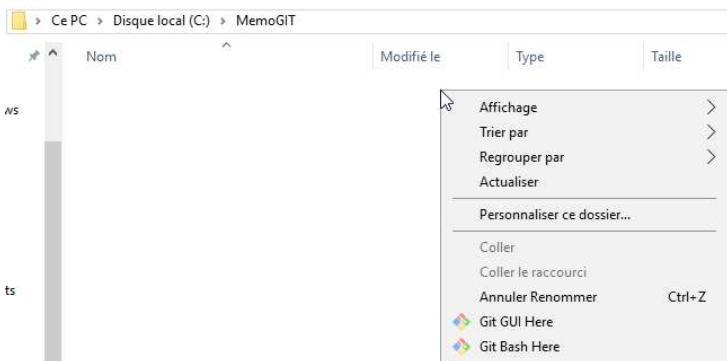


Figure 5 Ouverture du bash GIT directement dans le dossier cible.

```
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT
$ git clone https://github.com/CerfontaineMathieu/MemoGIT.git
Cloning into 'MemoGIT'...
warning: You appear to have cloned an empty repository.

Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT
$ |
```

Figure 6 Cloner le dépôt grâce à l'adresse HTTP GitHub.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2.1.2 Travailler avec GIT

Maintenant que l'on a cloné ou créé un nouveau dépôt dans un dossier donné, celui-ci contient un dossier « .git ». Grossièrement, c'est ce fichier qui va gérer le tracking de votre projet et des modifications que vous lui avez apportées. Il ne faut donc surtout pas le supprimer. Sinon il faudra cloner à nouveau le dépôt dans le dossier.

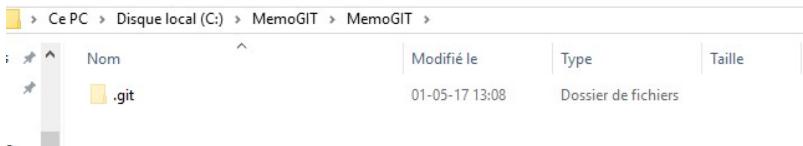


Figure 7 Fichier ".git" dans le dossier "MemoGIT".

Globalement, pour débuter il est important de comprendre le principe de « commit », « pull », « push », « branches » et « merge ». Dans notre dossier presque vide, à l'exception du dossier « .git », nous voulons rajouter un mémo sous forme de fichier texte. Dans un premier temps, ce fichier texte ne sera pas tout de suite synchronisé avec le dépôt GitHub/GitLab. Il faudra d'abord réaliser ce que l'on appelle un « commit ». Réaliser un « commit » consiste à signaler que les modifications sur un fichier donné, ou un dossier complet, sont terminées. Un commit s'accompagne toujours d'un commentaire pour désigner ce qui a été réalisé sur ce fichier. Attention, pour « commit » un fichier qui n'a encore jamais été ajouté au dépôt, il faut d'abord le signaler comme étant un fichier à suivre et à intégrer à l'historique. La plupart des GUI utilisés pour GIT réalisent cette étape automatiquement avant un commit, comme TortoiseGit par exemple.

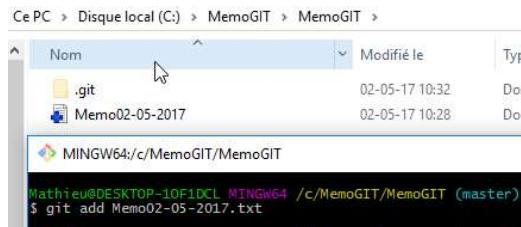


Figure 8 Ajouter le "tracking" d'un fichier ou d'un dossier par GIT.

Lorsque l'on ajoute le fichier, ou le dossier, au suivi de fichier de GIT, sous TortoiseGit une petite croix bleue apparaît. Une fois le fichier « commit », un symbole vert incrusté d'un « V » de confirmation nous montre que toutes les modifications apportées au fichier sont enregistrées. Dans ce cas-ci le fichier est encore vide, il ne s'agissait que d'un premier « commit » d'ajout.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (master)
$ git commit Memo02-05-2017.txt
[master (root-commit) 57d4996] Commit test
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Memo02-05-2017.txt
```

Figure 9 Commit du fichier Memo02-05-2017.txt

Cependant, réalisé-réaliser ce « commit » ne suffit pas pour l'envoyer dans le dépôt. C'est là qu'intervient le concept de « push ». « Pusher », un fichier revient à dire que l'on synchronise ce fichier avec le dépôt. Si ce fichier n'existe pas dans le dépôt il est créé sinon les modifications sont appliquées à sa version définitive.

```
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (master)
$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 228 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/CerfontaineMathieu/MemoGIT.git
 * [new branch]      master -> master
```

Figure 10 "Push" les modifications vers le "repository" GIT.

Imaginons maintenant qu'autre développeur modifie le fichier Memo.txt. Si l'on souhaite récupérer la nouvelle version, on réalise alors ce que l'on appelle un « pull ». « Pull » permet de récupérer un dossier ou un fichier dans la dernière version que possède le dépôt.

Dans le cas où l'on réalise une manœuvre risquée dans un projet, comme réaliser un patch d'un bug ou que l'on crée une nouvelle fonctionnalité, il est toujours conseillé de créer, ce que l'on appelle, une branche. Une branche dans GIT est une sorte de copie parallèle du dépôt permettant à un développeur de réaliser sa nouvelle fonctionnalité ou son patch sans que cela impacte le dépôt principal. Cela permet de ne pas impacter le travail des autres développeurs sur ce projet dans le cas où la nouvelle fonctionnalité nécessiterait des modifications majeures et moins localisées. Créons donc une branche « 1stTest » dans laquelle nous mettrons un nouveau fichier texte.

```
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (master)
$ git branch 1stTest
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (master)
$ git branch
  1stTest
* master
```

Figure 11 Création de la branche "1stTest".

Nous constatons que nous sommes dans la branche « Master », qui est la branche principale du dépôt. Déplaçons-nous dans la nouvelle branche créée pour y ajouter un fichier.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (master)
$ git checkout 1stTest
Switched to branch '1stTest'
Mathieu@DESKTOP-10F1DCL MINGW64 /c/MemoGIT/MemoGIT (1stTest)
$
```

Figure 12 Changer de branche dans GIT.

.git	02-05-17 10:58	Dossier de fichiers
Memo02-05-2017	02-05-17 10:48	Document texte
MonNouveauMemo	02-05-17 10:59	Document texte

Figure 13 Ajout d'un fichier texte dans la nouvelle branche.

Si l'on retourne maintenant dans la branche « master », on peut constater la disparition du fichier texte « MonNouveauMemo » (il faut évidemment au préalable ajouter ce fichier au suivi GIT).

Une fois les modifications faites, le développeur à-a l'occasion alors de faire un « merge » de la branche avec le dépôt. Cela permet d'ajouter les nouvelles fonctionnalités et de les rendre disponibles aux autres développeurs dans la branche principale.

2.1.3 Les GUI

Il est évidemment plus aisés d'utiliser un GUI pour travailler avec GIT. Bien que débuter en tapant les commandes GIT soit recommandé pour améliorer sa compréhension du gestionnaire de versions, il est plus productif de se tourner vers un GUI par la suite. GIT embarque un GUI avec lui lors de son installation sur un PC, cependant l'utilisateur peut souhaiter quelque chose de plus avancé.

On peut citer, par exemple, TortoiseGit qui permet d'ajouter des options lors d'un clic droit, permettant d'outrepasser l'utilisation d'une fenêtre.

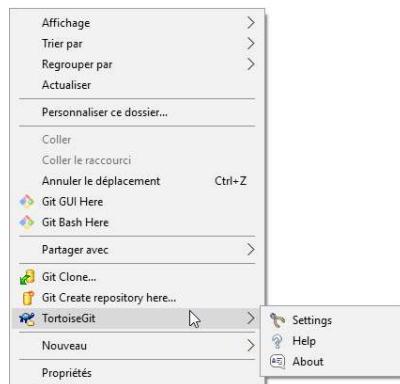


Figure 14 TortoiseGit.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2.2 Un langage de script

Pour commencer, il est sage de décrire ce qu'est un langage de script et ce qu'il a de différent par rapport au langage de programmation que l'on a l'habitude de manipuler. Cependant, beaucoup de professionnels dans le milieu de l'informatique, et surtout du développement, s'accordent à dire que la définition d'un langage de script est constamment sujette au changement. Ainsi, nous pouvons citer l'inventeur du Perl en personne:

« basically, scripting is not a technical term. When we call something a scripting language, we're primarily making a linguistic and cultural judgment, not a technical judgment » - Larry Wall (créateur de perl et linguiste).

Nous allons tout de même énoncer plusieurs points qui reviennent assez souvent dans les différentes définitions d'un langage de script et les utiliser comme base par la suite pour comparer le PowerShell à ses concurrents directs.

Tout d'abord, un langage de script est un langage de programmation particulier qui permet de manipuler des fonctionnalités du système d'exploitation. Pourtant la différence fondamentale entre un langage de script et un langage de programmation vient du fait que le langage de script est interprété et non compilé. Cela signifie que le code source est stocké dans un fichier (par exemple .ps1,.psm1, ... pour PowerShell) et que ce code source sera interprété par après lors de l'exécution du script par, vous l'aurez deviné, l'interpréteur du langage dans lequel il est écrit.

En somme, le script peut être exécuté immédiatement après son écriture sans phase de compilation, il peut être utilisé sur n'importe quel ordinateur à partir du moment où l'interpréteur du langage est présent sur la machine, et ce pour la modique somme de la taille d'un fichier texte et peu importe l'architecture matérielle ou logicielle. Cependant, on y perd énormément en vitesse par rapport à un langage compilé car le code doit être analysé à chaque exécution ligne par ligne par l'interpréteur et celui-ci doit alors les traduire pour qu'elles soient exécutables.

On peut se poser alors la question : « à quoi peut bien servir un langage de script ? ». Un langage de script peut servir à automatiser des tâches complexes ou rébarbatives sur un système d'exploitation, à générer des pages dynamiques sur un serveur web ou même à créer des interfaces graphiques. Par exemple sous PowerShell, on peut très bien utiliser les librairies graphiques disponibles en « .NET » pour créer une interface GUI. Certains langages comme le Python et le Perl sont d'ailleurs utilisés aussi pour créer des programmes complets.

Mis en forme : Français (Belgique)

Mis en forme : Anglais (États-Unis)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2.3 Windows PowerShell

Windows PowerShell est une suite de logiciels anciennement connue sous le nom de Windows Command Shell. Cette suite contient une interface en ligne de commandes, le langage de script PowerShell et un kit de développement. Cette suite est apparue sur Windows 7, bien que prévue initialement sur Windows Vista, et est une descendante directe de la ligne de commande bien connue, DOS.

PowerShell est donc un langage de script propre à Windows. Le but de Microsoft était de créer un langage de script aussi fourni en possibilités et aussi sécurisé que les langages de script UNIX.

Syntaxiquement parlant les commandes en PowerShell, que l'on appelle des « CmdLets », ressemblent fortement aux commandes que l'on peut trouver sous UNIX Shell. C'est-à-dire que l'on appelle ces fonctions par des alias et que l'on peut leur passer des paramètres. Dans les deux systèmes, la sortie d'une commande peut être « pipée » vers une autre commande. Pour paraphraser, on peut récupérer le résultat de la première commande pour réaliser le traitement des données résultantes au sein de la seconde. Cependant, leurs fonctionnements diffèrent au niveau du transport des informations.

En effet, comme dit précédemment, PowerShell permet d'utiliser des libraires .NET et donc des objets. Cela veut dire que l'on peut renvoyer en sortie d'une commande un objet représentant le résultat. Pour traiter le retour de la fonction, il suffit donc d'utiliser les variables et les méthodes membres de l'objet.

À l'inverse, en UNIX Shell, beaucoup de choses sont basées sur du texte y compris la sortie d'une commande. Cela signifie que si l'on veut transmettre le résultat de cette commande à une autre celle-ci devra traiter le texte sortant pour en interpréter le résultat.

Voici un exemple de code concret écrit en PowerShell permettant d'illustrer le passage d'informations d'une commande à une autre par le « pipe » :

```
Add-Type @"
    using System;
    public class MyType{
        public string FirstName;
    }
"@
```

Exemple 1 Crédit d'un type PowerShell

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
function newName($myObject)
{
    if ($myObject.GetType() -eq [MyType])
    {
        #print the current name to screen
        $myObject.FirstName
        #change string in the 'name' property
        $myObject.FirstName = "NewName"
    }
    return $myObject
}
```

Exemple 2 Création d'une fonction permettant de changer la valeur d'une variable de l'objet créé précédemment

```
$myObject = New-Object MyType -arg "Luke" | newName
```

Exemple 3 Instanciation du type et pipe vers la fonction

Dans cette suite d'exemples, on voit que l'on peut passer directement l'objet instancié par la cmdlet « New-Object » dans la méthode « newName ». La variable « \$myObject » contiendra en sortie le résultat de la fonction « newName ». À noter qu'il est tout à fait possible d'appeler depuis une invite de commandes PowerShell, les commandes natives de Windows comme « ipconfig », « netstat », « ping », etc.

Pour ne pas trop s'attarder sur la présentation du PowerShell, concluons sur ses différents avantages et inconvénients grâce à ce tableau :

Avantages	Inconvénients
Langage orienté objet	PowerShell ne peut pas afficher les caractères Unicode lorsqu'il est utilisé au travers de Windows console subsystem (applications tournant dans l'environnement de boot Windows).
Utilisation du Framework .NET	
Syntaxe relativement simple à la compréhension	
Simplifier la collecte d'informations du système	
Automatiser des tâches complexes sur l'ensemble d'un parc de machines Windows.	

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2.4 La syntaxe d'une cmdlet PowerShell

La syntaxe d'appel d'une cmdlet PowerShell ressemble fort à celle d'une commande sous UNIX et Windows. Cependant, certaines choses sont différentes et propres au PowerShell.

Le nommage d'une cmdlet respecte certaines règles afin que les utilisateurs puissent les mémoriser ou les retrouver facilement. Le verbe définit l'action que l'on va appliquer sur le nom, celui-ci définit en général le type d'objet manipulé. Les paramètres éventuels sont listés avec un espace entre chaque.

Verbe-Nom [[-parametre0 **arg0**] [-parametre1 **arg1**]]

Le nombre de paramètres est bien évidemment le plus souvent variable. Cependant certains paramètres sont toujours sous-entendus lors de la création d'une cmdlet même s'ils n'ont pas été déclarés. Parmi ceux-ci on peut citer : -ErrorAction, -ErrorVariable, -Debug, -Verbose, ...

2.5 PowerShell, Perl, Python et Bash

2.5.1 Perl

Le Perl est un langage de programmation créé par Larry Wall en 1987. Il est inspiré de langage de script, comme Shell, et du C lui-même. C'est un langage interprété comme le PowerShell qui permet de traiter majoritairement des informations à base de texte.

En effet, le Perl permet de récupérer des données textuelles pour en établir un rapport détaillé. Il inclut donc le système d'expression régulière au sein même de sa syntaxe en plus des librairies standard du C. A la base, il permet de réaliser des scripts dans le cadre de la gestion de systèmes UNIX mais à-a évoluer évolué avec le temps pour devenir un outil polyvalent.

À l'heure d'aujourd'hui-Actuellement le Perl est utilisé pour faire de l'administration sous les systèmes UNIX aussi bien que sous les systèmes Windows. Il peut aussi être utilisé dans des programmes conséquents en tant que langage principal ou tout simplement pour faire le lien entre deux programmes, formant ainsi un projet bien plus gros.

En somme, Perl est un langage très pratique utilisé pour faire un peu près tout. C'est d'ailleurs ce qui lui a valu son surnom de « rouleau de scotch de l'internet ».

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
my $s = 'toto' ;          # variable scalaire à portée
lexical;
local $level += 1;        # variable scalaire avec une
valeur à portée dynamique
our @s = (1, $s, 3.14);  # variable tableau globale au
module courant
```

Exemple 4 Exemple syntaxe du Perl (Perl (langage) - Wikipedia, 2017)

2.5.2 Python

Le Python est un langage de programmation orienté objet créé en 1990 par Guido van Rossum. Il a été conçu pour augmenter l'efficacité des programmeurs en leur fournissant un langage avec une syntaxe de haut niveau, simple à utiliser.

Tout comme Perl, Python est un langage outils permettant de faire un peu près tout. Il est cependant le plus souvent utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses.

Python est aussi utilisé comme langage « prototype ». C'est-à-dire qu'il permet de concevoir un programme fonctionnel servant de structure à une version plus évoluée écrite par après dans un langage de programmation de plus bas niveau comme le C.

Il permet de créer des projets bien plus gros comme des jeux, des logiciels multimédias, etc ... On dit de Python que c'est un langage multiparadigme permettant donc de faire de l'orienté objet, du fonctionnel ou encore de procédural. Enfin le Python est souvent utilisé pour administrer un système informatique sous UNIX ou Windows.

En conclusion, le Python est un langage fort orienté pédagogie et qui facilite l'apprentissage de la programmation, il est multi usage. De ce fait, il est utilisé aussi bien par les administrateurs systèmes que par les développeurs.

```
def factorielle(n):
    if n < 2:
        return 1
    else:
        return n *
            factorielle(n - 1)
```

Exemple 5 Exemple de Python (Python (langage) - Wikipedia, 2017)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

2.5.3 Bash

Un script bash est un fichier texte contenant une suite d'instructions, de commandes qui constituent un scénario d'actions. C'est un fichier texte que l'on peut exécuter c'est-à-dire lancer comme une commande. Ce fichier texte respecte une certaine structure. Ainsi il commence toujours par cette ligne : « # ! /bin/bash » dans le cas d'un script bash. Cette ligne varie selon l'interpréteur utilisé. En effet, le symbole « # ! » est ce que l'on appelle un « Shebang », c'est-à-dire un en-tête de fichier texte permettant de spécifier :

- Le fait que le fichier n'est pas un fichier binaire mais un script,
- Le chemin vers l'interpréteur a utilisé.

Le script bash est le script le plus basique que l'on peut retrouver sous UNIX. Il permet d'exécuter une suite de commandes UNIX l'une après l'autre à des moments donnés comme au démarrage de la machine par exemple, ou à son extinction. Contrairement au Python, au Perl ou encore au PowerShell, le fichier texte est lu et analysé au moment de l'exécution, une erreur au milieu du fichier ne sera détectée qu'au moment du lancement.

On considère souvent le bash simplement comme un outil utilisé pour lancer des commandes. À ce titre, beaucoup font l'erreur de ne pas le considérer comme un langage interprété. Pourtant:

“Bash is the shell, or command language interpreter, for the GNU operating system.”
(What is shell ? - gnu.org, 2016)

2.5.4 Les différences et les points communs avec PowerShell

Comme vous l'aurez sûrement remarqué, ces trois langages de script sont tous issus du monde UNIX. Même si, grâce à leurs bibliothèques et les multiples ajouts de la communauté, Perl et Python permettent maintenant l'administration et la gestion des systèmes Windows, il n'en a pas toujours été ainsi.

Cependant, leur portée de configuration d'un système d'exploitation Windows, déjà énorme, paraît évidemment très limitée par rapport au PowerShell qui, lui, est à la base totalement dédié à cet OS. À savoir que, depuis peu, PowerShell est maintenant open source et a été porté sur Linux.

PowerShell permet donc de manipuler de façon plus étendue les environnements Windows en même temps qu'il utilise des concepts déjà très familiers pour les informaticiens du monde Microsoft comme les WMI, .NET, etc ...

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Bash quant à lui est propre à UNIX et n'est pas du tout reconnu par un système Windows. En effet, comme dit précédemment le script bash peut être résumé comme une suite de commandes UNIX.

Chaque langage possède évidemment sa syntaxe avec ses particularités. Avec Python, par exemple, on devra dire adieu à la parenthèse et à l'accolade, pour les conditions. PowerShell possède quant à lui une syntaxe relativement similaire au C#, mise à part l'utilisation des cmdlets, qui peuvent s'apparenter aux méthodes, et du caractère « \$ », les développeurs .NET ne seront donc pas dépayrés.

L'objectif de ces langages reste évidemment l'automatisation d'une tâche d'administration plus ou moins complexe qui devrait, sans ces langages de script, être réalisée à la main par l'administrateur du parc informatique. On peut citer comme tâche particulièrement fastidieuse et répétitive, la configuration d'un switch ou d'un firewall.

Pourtant, contrairement au PowerShell, Python et Perl ne permettent pas uniquement de faire de l'administration réseaux et systèmes. Ces langages de script peuvent tout aussi bien être considérés comme des langages de programmation à proprement parlé.

En effet, il existe beaucoup d'exemples de programmes fonctionnels utilisés au quotidien écrits en Python ou en Perl. Citons par exemple, « Eclipse », « Netbeans » ou encore des jeux vidéo comme « Civilization IV », « Eve Online » ou encore « World of tanks ».

À l'inverse, on peut classer le PowerShell comme un langage de script dans le sens le plus noble du terme. Sa nature et l'objectif même de sa création sont la configuration et l'administration du système.

Pour finir la liste des comparaisons, parlons du fait que le Python est actuellement supporté, maintenu et développé par une association à but non lucratif qui est la **Python Software Foundation (PSF)** à l'inverse du PowerShell qui, malgré son passage récent en open source, est toujours géré par Microsoft.

En résumé, le Perl et le Python sont des langages très simples à l'utilisation et sont d'ailleurs même recommandés par les pédagogues pour l'apprentissage de la programmation en général. Ils permettent la configuration d'un peu près tout et n'importe quoi sur une machine. Leurs portées ne s'arrêtent pas à Windows cependant, le PowerShell convient mieux dans ce genre de situation. Celui-ci prend maintenant en charge les environnements Linux et n'a pas fini de s'offrir de nouvelles possibilités comme la gestion de base de données SQL. Le PowerShell est un peu moins facile à l'apprentissage que le Perl ou le Python mais à l'avantage de conserver des paradigmes utilisés dans la programmation .NET.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

3 Etude de l'existant.

3.1 La problématique

Le projet YASC a vu le jour dans le cadre d'une migration d'applications d'un client de Limelogic. Ce client avait besoin, pour des raisons de sécurité, de migrer 130 applications web IIS d'un serveur Windows 2003 vers un serveur Windows 2008 et tout cela pour quatre environnements : ~~déploiement, développement, test~~, acceptance, ~~test~~ et production.

Devant une tâche aussi répétitive et chronophage, le choix de développer un logiciel, ou plutôt un engine permettant de déployer ces applications, a été fait. L'objectif en créant YASC n'était pas seulement d'obtenir un outil permettant de déployer une application. En effet, beaucoup de software existent déjà et permettent de le faire. Le problème principal de la majorité de ces logiciels est un manque de documentation concernant les applications déployées et lisibles aisément par un humain.

Avec YASC, nous avons voulu faire en sorte de pouvoir déployer une application quelconque et de récupérer une documentation claire et précise des paramètres de déploiement. L'objectif est assez évident : permettre un dépannage plus simple et plus efficace.

Dans le cas du client de Limelogic, le choix est assez facile à justifier. En effet, comme beaucoup d'autres, il possède une équipe de développement pour créer des applications, utilisées en interne par les employés ou en externe par les clients, ainsi qu'une équipe d'exploitation pour mettre en place et maintenir ces applications sur les serveurs. Quand un problème arrive, c'est à l'équipe d'exploitation de régler le problème dans les plus brefs délais. En l'absence d'une documentation claire permettant le dépannage de l'application, ou du moins offrant des pistes de dépannage, l'équipe d'exploitation doit se référer à l'équipe de développement. Encore faut-il que nous parlions de problèmes logiciels car une panne réseau peut aussi très bien arriver. Le problème peut alors devenir extrêmement difficile à repérer et la panne très longue à être réparée.

Cependant, après les débuts de YASC, son utilité s'est déployée. Grâce au PowerShell et à l'architecture même du projet, YASC permet de faire bien plus de choses que déployer des sites web ou des applications. On dit maintenant de YASC qu'il déploie des applications dans le sens général du terme. Pour YASC, une application peut très bien être un switch ou un firewall à configurer, comme un utilisateur à rajouter à l'active directory de Windows ou encore une ligne DNS à rajouter à un serveur de nom de domaine.

Les utilités de YASC sont bien diverses et elles ne font que se multiplier avec le temps de par les possibilités qu'offre son architecture.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Cependant, des solutions alternatives existent et il est important d'expliquer pourquoi il a semblé préférable de développer une nouvelle alternative. Parlons donc de trois concurrents directs de YASC : Chef, Puppet et enfin, Rudder.

3.2 Chef

Chef est un programme de gestion de configuration créé par Adam Jacob qui utilise un langage dédié écrit en pure-Ruby et en Erlang. Il est libre et sous licence open source Apache 2.0. Le but de Chef est de permettre la gestion d'un parc de serveurs informatiques grâce à des « recipes » ou des « cookbooks » qui décrivent en fait la configuration du système.

Chef fonctionne comme ceci : l'utilisateur écrit des recettes qui décrivent comment déployer et/ou maintenir un serveur d'applications ainsi que ses services. Cela peut aller d'un serveur web à une base de données MySQL. Comme dit précédemment ces recettes décrivent la configuration du système pour déployer une application. Plus précisément, elles décrivent l'état dans lequel doit être une ressource donnée du système. Elles peuvent donc spécifier que tel ou tel service doit être en cours d'exécution, ou qu'un tel fichier doit être présent sur la machine hôte.

Une fois que la recette est écrite, Chef s'assure que toutes les ressources décrites à l'intérieur sont bien dans l'état demandé. Chef peut fonctionner de deux manières distinctes :

- En mode client/serveur : dans ce mode, le client envoie différents attributs à propos du nœud actuel au serveur Chef. Ce même serveur utilise Apache Solr pour indexer ces attributs et fournit une API au client pour qu'ils puissent questionner ces informations indexées. Concrètement, les recettes réalisent des requêtes sur ses attributs utilisant les données récupérées pour configurer le nœud. Par noeud on entend la machine faisant tourner les clients Chef.
- En mode standalone (Chef solo) : le mode chef-solo/standalone, permet au logiciel de s'affranchir du serveur Chef pour converger les « cookbooks ». Ce mode utilise le client chef en mode local et ne supporte évidemment pas plusieurs fonctionnalités disponibles en mode client/serveur comme, par exemple, la distribution centralisée des « cookbooks », l'API centralisée permettant d'interagir avec les composants infrastructures intégrés, ou encore tout simplement l'authentification et l'autorisation.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.



Figure 15 Chef Architecture (An overview of chef, s.d.)

Chef fournit aussi un Development Kit qui permet de développer et tester le code des recettes pour une infrastructure donné dans une workstation locale et sécurisée.

3.3 Puppet

Tout comme Chef, Puppet est un programme de gestion de configurations. Il permet de gérer et configurer les serveurs d'un réseau ainsi que les applications qui tournent dessus.

Puppet s'attribue plusieurs avantages indéniables :

Réduction des cycles pour optimiser les déploiements d'applications
Effectuer des changements rapides et itératifs.
Définition d'une configuration utilisable et applicable sur un parc de machines
Correction automatique des changements de configuration non souhaitée.
Obtenir des détails poussés sur la configuration logicielle et matérielle des machines du parc.

3.3.1 Architecture

L'architecture de Puppet peut être divisée en plusieurs points décrits ci-dessous :

Le Master of Master (MoM), aussi connu sous le nom de « Puppet Master » est le centre des activités et des processus sous Puppet Enterprise. C'est dans le MoM que le code est compilé pour créer des catalogues d'agents où l'on va vérifier et signer les certificats SSL. Puppet Enterprise peut être installé suivant deux modes :

- Le mode monolithique : le MoM héberge tous ses services actifs comme PE console, PuppetDB, ... sur une seule machine bien définie.
- Le mode « split » : tous ces services sont hébergés sur différentes machines nœud.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Indépendamment de l'architecture d'installation, le MoM contient toujours un « compile master » et un serveur Puppet.

Le « compile master », comme dit plus haut, est un composant du MoM. Il contient un serveur Puppet et peut-être plusieurs au sein d'un MoM. Il contient aussi un compilateur de catalogue et une instance de synchronisation de fichiers. Il est conseillé d'augmenter le nombre de « compile master » à l'intérieur d'un MoM pour augmenter la capacité de traitement.

Le « Puppet Server », qui s'exécute sur une machine virtuelle JAVA qui, elle-même,^b est hébergée sur le MoM. Le serveur Puppet héberge les ressources nécessaires au service de vérification des certificats. Il gère en plus le compilateur de catalogue. Ce compilateur sert à compiler les catalogues de configuration utilisés par les clients Puppet qui utilisent du code Puppet et/ou d'autres sources de données.

Le « Catalog compiler », est utilisé pour gérer les informations contenues dans un catalogue. Un catalogue est un fichier utilisé pour configurer un agent Puppet. Ce fichier est chargé depuis le MoM ou le « compile master », il décrit l'état dans lequel doit se trouver chaque ressource nécessaire à la configuration de l'agent. Il est évident qu'il existe parfois des interdépendances entre certaines ressources et qu'elles doivent être gérées dans l'ordre. C'est à cela que sert le compilateur de catalogue.

Le « File-File sync », ou la synchronisation de fichiers en français, permet de garder le code Puppet synchronisé sur différents « compile masters ». Il repère les changements dans le répertoire de travail principal sur le MoM et déploie le code présent dans un répertoire commun qu'il peut ensuite synchroniser avec tous les « compile masters ».

Le « Certificate Authority », ou l'autorité de certification en français, est un service de CA sous Puppet qui accepte des CSR (« Certificate signing requests ») ainsi que des CRL (« Certificate revocation list ») provenant des nœuds Puppet. Le CA de Puppet utilise des fichiers d'extension « .pem » dans le répertoire « ssldir » pour conserver en mémoire les identifiants des clients.

Et enfin, pour terminer, le « Puppet agent », ou en l'agent Puppet en français. L'agent Puppet est le nœud managé par le serveur Puppet. Le serveur Puppet peut gérer plusieurs agents à la fois sans problèmes. Les nœuds exécutent l'agent Puppet sous forme d'un service en tâche de fond qui envoie périodiquement des requêtes de catalogue au serveur Puppet pour se tenir à jour en cas de changement de la configuration. Quand un nœud reçoit un catalogue en provenance d'un serveur Puppet il vérifie que chaque ressource décrite dans celui-ci est bien dans l'état demandé et, si non, exécute les changements nécessaires. Une fois le changement effectué, le client Puppet fournit pour seul documentation un rapport qui sera stocké dans la

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

base de données Puppet ; PuppetDB, qui est accessible seulement en console grâce à PE Console.

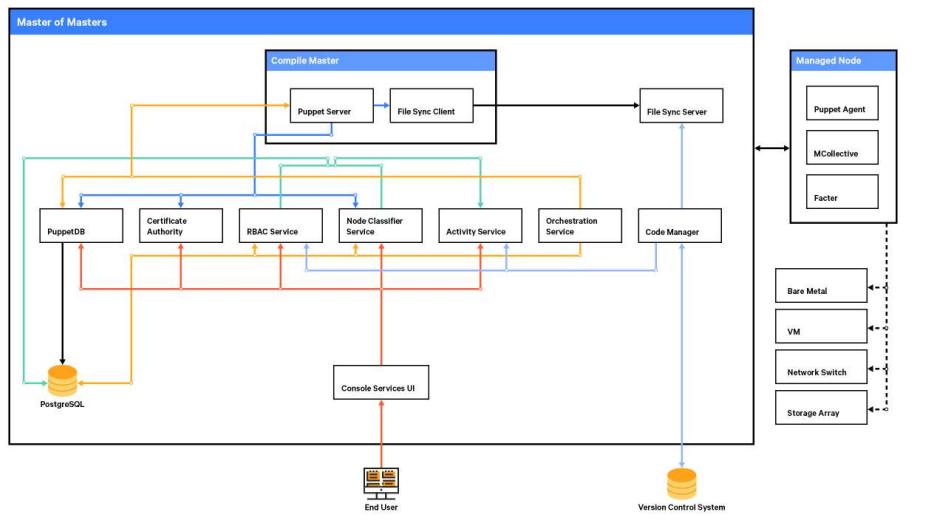


Figure 16 Puppet Architecture (Puppet Enterprise user's guide, 2017)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

3.4 Rudder

Comme ses concurrents présentés ci-dessus, Rudder est un logiciel de configuration automatique d'un parc informatique. C'est un logiciel libre et donc open source orienté web dont la partie serveur (interface web) est écrite en Scala et la partie agent écrite en C. Rudder fonctionne donc avec des agents installés sur les machines du parc à gérer.

Rudder se veut facile d'utilisation et ce pour différents niveaux d'expertise. Ainsi il permet au profane de spécifier les paramètres de déploiement dans le centre de contrôle qui seront ensuite appliqués par Rudder. Un rapport graphique est ensuite affiché sur l'interface utilisateur.

Plus encore, il permet aux experts de spécifier l'implémentation même des paramètres qui seront appliqués sur les machines du parc qui elles-mêmes peuvent tourner sur différents systèmes. Il permet aussi au manager de consulter les rapports d'exécution et les logs qu'il a récoltés au cours de sa configuration.

Au niveau de son fonctionnement Rudder implémente deux fonctionnalités majeures. La première étant évidemment la gestion des diverses configurations des agents via des « règles ». Ces règles sont appliquées sur les nœuds agents. La définition même d'une règle englobe : l'installation d'un logiciel, d'un outil, la configuration d'un service, l'exécution d'un daemon, etc.

La deuxième consiste à gérer les « assets » de chaque nœud agent. En effet, chaque agent est chargé d'envoyer en permanence un inventaire complet au serveur Rudder. L'objectif principal de la gestion des « assets » est surtout de permettre l'identification des nœuds agents ainsi que leurs caractéristiques propres comme le système d'exploitation en présence ou encore le système de fichier. La gestion de configuration dépend principalement de ce principe « d'asset ».

Le fonctionnement de la gestion des « assets » est assez simple, il existe trois concepts à retenir principalement :

- Le concept de « Nouveau nœud » : lors du premier inventaire de ses caractéristiques un nœud est placé dans un état de transit. L'utilisateur a alors accès à un détail plus ou moins complet de ses caractéristiques. Il doit alors accepter dans le serveur Rudder d'ajouter le nœud à la base de données si désiré.
- Le concept de « Recherche de nœuds » : il est possible d'identifier via un moteur de recherche avancé, les machines du réseau local à gérer ou non. On peut ajouter à la base de données des machines selon différents critères comme une plage d'adresses IP, le système d'exploitation, etc ...

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

- Le concept de « Groupe de nœuds » : Rudder offre la possibilité de créer des groupes de nœuds agents. Ces groupes sont en fait créés sur base des résultats de la recherche exécutée via le moteur de recherche. Un groupe peut être statique ou dynamique. Dans le cas d'un groupe statique, la recherche est réalisée une et une seule fois, le résultat est alors stocké. Une fois le groupe déclaré, la liste des agents ne changera plus sauf en cas de modification faite à la main. Pour les groupes dynamiques, la recherche est réalisée à chaque fois que le groupe reçoit une requête. Le groupe contiendra donc la liste des agents correspondants aux critères de recherche ainsi, si un agent ne correspondant plus aux critères de sélection il sera supprimé du groupe.

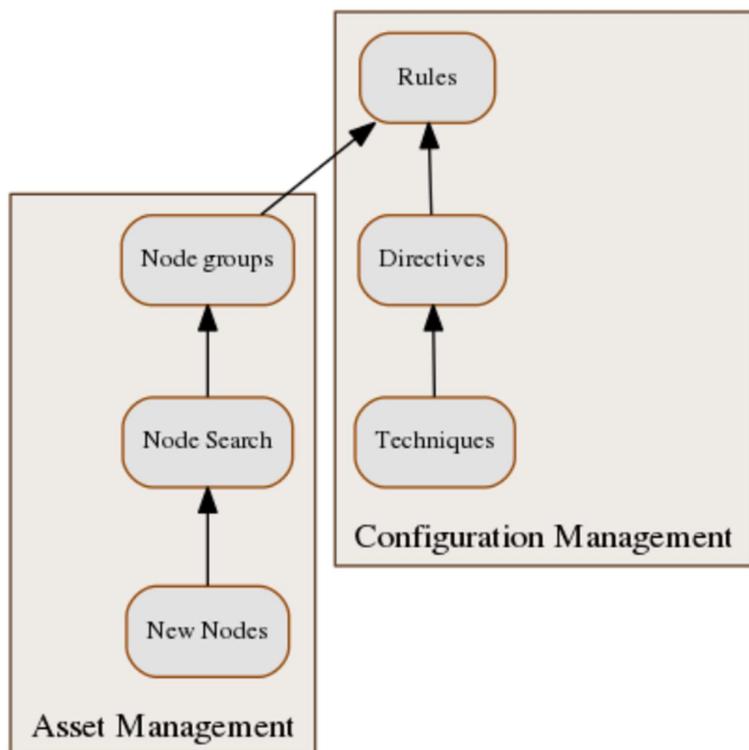


Figure 17Rudder Architecture (Rudder 4.0 - User's manual, 2016)

Rudder est un outil fort complet permettant de gérer la configuration d'une flotte aussi vaste que diverse de systèmes d'exploitation. Cependant il est honnête de souligner que pour gérer de manière très complète un parc de machine Windows avec Rudder il faudra investir dans l'achat de plugin prévu à cet effet.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

3.5 Pourquoi utiliser YASC ?

Après avoir décrit chacun de ces trois logiciels, tous reconnus maintenant comme de grands logiciels de gestion de configuration, on pourrait se demander : « Pourquoi créer un nouveau logiciel de gestion alors qu'il existe déjà des solutions ? »

Il faut savoir que ces trois logiciels sont conçus à la base pour des systèmes UNIX. C'est d'ailleurs la raison pour laquelle ils permettent de gérer un catalogue de système UNIX beaucoup plus complet que celui des systèmes Windows server.

Prenons l'exemple de Rudder. À l'heure où ces lignes sont écrites Rudder permet de gérer seulement jusqu'à la release 2012 de Windows Server. Alors qu'il permet de gérer pas moins d'une vingtaine de systèmes UNIX distincts. De plus comme dit précédemment, pour gérer complètement et correctement un système Windows il faudra investir dans un plugin additionnel de Rudder ...

Puppet quant à lui permet gratuitement la gestion d'un environnement Windows. Cependant son langage dédié (DSL) nécessite une expertise supplémentaire pour gérer un environnement Windows de façon plus approfondie via la programmation de modules indépendants.

Chef, quant à lui, permet d'exécuter directement du code PowerShell mais nous ne nous sommes pas penchés sur cette solution pour une autre raison qui sera expliquée plus bas dans ce chapitre.

Pourquoi faire tout cela alors que Microsoft nous offre un large de programmation permettant de créer des scripts de configuration Windows ? Le PowerShell est un langage maintenant natif à Windows qui permet de gérer l'entièreté des fonctionnalités d'une machine Windows et qui, de plus, est porté depuis peu sur UNIX. Depuis quelques années, la gestion d'un parc informatique de machines Windows ne s'imagine plus avec un autre langage que PowerShell. Il serait fou de s'obstiner à configurer des machines via des outils à l'origine non prévus à cet effet, alors que le concepteur même de Windows, Microsoft, nous met à disposition un langage natif prévu pleinement à des fins de configurations de leur système d'exploitation. C'est donc pour cela qu'il est évident d'utiliser un outil de gestion de configuration écrit entièrement en PowerShell.

Comme dit précédemment dans la présentation de la problématique en début de ce chapitre, la décision de développer YASC n'a pas été prise seulement pour une question de langage de programmation. Un des objectifs majeurs de YASC est de pouvoir gérer une documentation décrivant de manière claire, précise et détaillée comment a été configurée une application sur une machine quelconque. Cette documentation est actuellement présentée sous

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

la forme d'un document HTML assez simple. C'est l'absence de cette documentation sous le logiciel Chef qui nous a poussé à l'abandonner.

Plus encore, lors de la création d'un script de déploiement YASC crée, en plus de la documentation, une copie de ses fichiers de configuration et du fichier de configuration de l'application. Cela permet de conserver le contexte et l'environnement dans lesquels a été généré le script de déploiement YASC.

Cela a plusieurs avantages. D'abord il est clair qu'une documentation personnelle pour chaque déploiement, classé en fonction de la date et l'heure de la génération a un avantage sérieux d'un point de vue dépannage. Si l'application qui sera déployée grâce à ce script ~~a-a~~ un bug, il est plus simple d'avoir une vue d'ensemble de ses paramètres d'installation pour réaliser son dépannage.

Il est aussi utile de conserver une copie du contexte et de l'environnement dans leur état au moment de la génération. Prenons par exemple le cas du fichier des métadonnées de YASC. Ce fichier, brièvement expliqué, permet de décrire l'environnement client dans lequel va tourner YASC. Il contient donc par exemple, les adresses IP des différents serveurs du client, les adresses mails des personnes responsables, etc.

Imaginons maintenant vouloir modifier ce fichier pour des raisons quelconques. Suite à ce changement, certains fichiers de déploiement ne fonctionnent plus correctement et refusent de déployer leurs applications. Pour isoler le problème, il suffit alors de réaliser une comparaison entre le fichier de métadonnées actuel et sa copie réalisée au moment où le déploiement de l'application fonctionnait encore.

Pour conclure, YASC est en cours de développement car, Limelogic a jugé nécessaire de réaliser un outil de déploiement d'applications :

- Écrit et ~~réaliser-réalisé~~ principalement pour configurer des machines Windows Server.
- Qui configure ces machines Windows de la façon la plus pointilleuse et la plus complète possible.
- Qui simplifie les phases de dépannage et améliore la communication entre les membres d'une équipe d'administrateurs par le biais d'une documentation ciblée.

Cependant, YASC permet maintenant de configurer d'autres appareils que des ordinateurs Windows comme par exemple des switchs HP ProCurve et des firewalls FortiGate de chez Fortinet. Son évolutivité n'a potentiellement aucune limite.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4 YASC

Ce chapitre a pour but d'expliquer de façon exhaustive le fonctionnement de YASC, les ressources qui lui sont nécessaires et les objets qui résultent de l'exécution de l'engine.

Pour détailler le plus précisément les mécanismes de YASC nous utiliserons le diagramme de fonctionnement à la page suivante. Comme celui-ci le montre de manière assez général, YASC a besoin de plusieurs ressources pour fonctionner.

En premier lieu, le fonctionnement de YASC requiert quatre fichiers de configuration : le fichier de configuration de l'engine, le fichier des métadonnées décrivant le(s) environnement(s) de déploiement, le fichier d'engine décrivant les modules PowerShell utilisé par YASC et enfin le fichier de déploiement décrivant les étapes de déploiement d'une application.

Ensuite, YASC aura besoin des modules PowerShell qui sont décrits dans le fichier d'engine. Les modules PowerShell représentent la réalisation d'une tâche atomique relative à l'application à déployer.

Le diagramme de fonctionnement exprime également le résultat d'une exécution YASC. La sortie d'un traitement YASC nous donne trois ressources intéressantes :

- Evidemment, la création d'un script permettant de déployer l'application sur la machine concernée,
- La génération d'une documentation créée sur base des cmdlets utilisés lors du déploiement de l'application. Celle-ci décrit les étapes réalisées durant le déploiement et les paramètres de configuration utilisés pour l'application,
- YASC envoie un mail au responsable de l'environnement, les notifiant qu'une application a été déployée.

Ce chapitre sera donc divisé en 3 sous-chapitres :

- I) Les ressources nécessaires pour faire fonctionner YASC, plus précisément : les différents fichiers XML.
- II) Les modules PowerShell : la façon dont ils doivent être programmés, comment les utiliser dans YASC, etc.
- III) Le résultat d'une exécution de YASC : email, script de déploiement et autre sortie.

Chaque sous-chapitre permettra de présenter une ou plusieurs ressources en plus d'expliquer comment YASC les utilise.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.1 Diagramme de fonctionnement

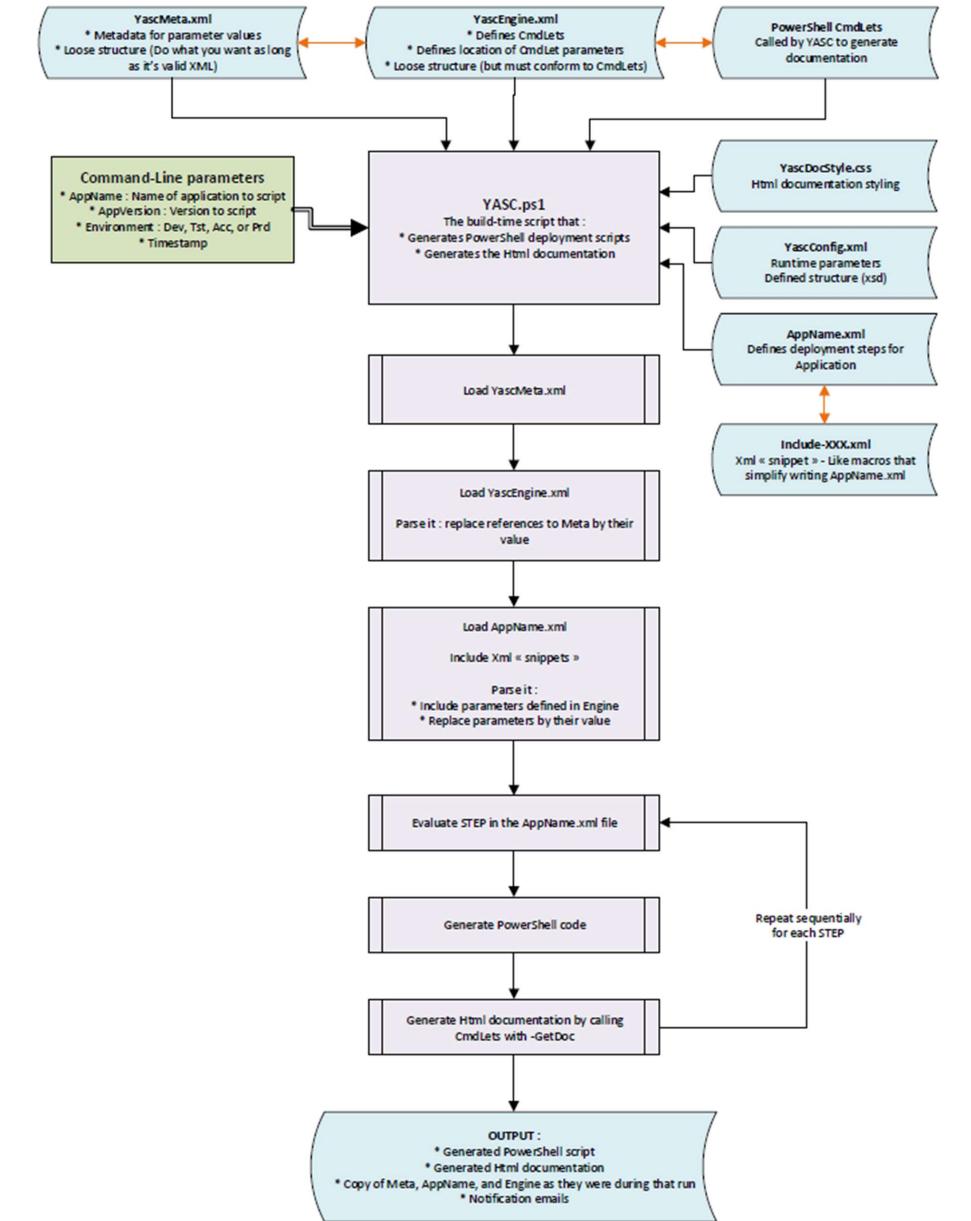


Figure 18 YASC Architecture (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

4.2 Les fichiers XML de configuration

Les fichiers de configuration que l'on peut voir sur le schéma de fonctionnement à la page précédente sont des fichiers XML. Ils sont au nombre de quatre. Ils sont très importants car ils disposent des informations nécessaires à la génération d'un script de déploiement par YASC. Ainsi dans ce chapitre, nous détaillerons le fonctionnement de YASC et la façon dont il utilise les fichiers XML.

4.2.1 Le fichier de configuration de YASC : YascConfig.xml

Le fichier de configuration est le premier fichier chargé en mémoire par YASC. Il lui permet de récupérer les informations qui lui sont nécessaires pour fonctionner. De manière générale, ce fichier permet de décrire des ressources nécessaires à YASC pour la création des scripts de déploiement. Ainsi dans ce fichier XML nous pourrons ranger ces différentes ressources :

- Le chemin vers le dossier de fichier temporaire de YASC : YascTempFolder. Qui est d'ailleurs utilisé pour télécharger et stocker temporairement les fichiers de configuration des switches.
- La définition des séparateurs de paramètres utilisés pour inclure des valeurs fournies dans d'autres fichiers XML. Ces séparateurs sont notamment utilisés lors de la création du fichier XML décrivant les étapes de déploiement, « AppData.xml »,
- Différents chemins vers des dossiers et des fichiers, par exemple le chemin vers tous les modules PowerShell,
- Les emails des personnes responsables qu'il faudra prévenir lors de la création du script de déploiement,
- Et bien d'autres encore.

Le fichier de configuration, contrairement au fichier des métadonnées (« YascMeta.xml »), doit respecter une certaine structure car ses paramètres sont significatifs pour YASC. C'est pour cela qu'il existe un fichier YascConfig.xsd qui permet de vérifier la structure du fichier. Au début de l'exécution, YASC va charger en mémoire le fichier de configuration et vérifier qu'il se conforme au schéma du fichier .xsd.

4.2.2 Le fichier des métadonnées : YascMeta.XML

Le fichier des métadonnées et le fichier de configuration que YASC charge en second. Il contient beaucoup d'informations concernant l'environnement dans lequel YASC va devoir déployer des applications. Il contient notamment :

- Une liste des serveurs sur lesquels il est susceptible de déployer. Ainsi que la signalisation du système d'exploitation que chacun d'eux fait tourner,

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

- Le chemin vers le « repository » pour chaque environnement de déploiement : production, test, ...
- La liste des emails du responsable pour chaque environnement de déploiement,
- Le nom d'hôte ou l'adresse IP du serveur SMTP à utiliser pour envoyer les mails de notifications,
- Le chemin vers le KeePass qui sera utilisé pour stocker tous les mots de passe dont YASC pourrait avoir besoin dans une cmdlet,
- D'autre information non négligeables comme les noms d'hôtes ou les adresses IP des différents switchs et firewall dans le réseau.

Une fois le fichier chargé, YASC sauvegarde chaque tag et leur valeur dans un objet déclaré comme global, lui permettant alors d'accéder depuis n'importe quel contexte aux valeurs que contient le fichier des métadonnées. Cela peut s'avérer très utile dans le cadre de données utilisées par plusieurs cmdlets.

Le fichier des métadonnées, contrairement au fichier de configuration et au fichier de l'engine, n'a pas besoin de respecter un schéma bien précis. L'utilisateur peut l'organiser comme il le souhaite, tant que cela reste écrit en XML correcte. En effet, c'est l'utilisateur lui-même qui va faire référence à ses valeurs. C'est donc à lui de voir comment il souhaite l'organiser.

4.2.3 Le fichier de définition des modules PowerShell : YascEngine.xml

Ce fichier contient une déclaration de chaque cmdlet PowerShell utilisé dans YASC pour réaliser le déploiement d'une application. Il est chargé par YASC directement après le fichier des métadonnées.

Concrètement, ce fichier organise ses cmdlets en deux catégories. Les cmdlets de test et les cmdlets d'actions.

Les cmdlets de test sont des cmdlets assez basiques permettant à YASC de faire une vérification lors du déploiement pour s'assurer qu'une condition est remplie. Nous pouvons par exemple citer la cmdlet permettant de vérifier les droits utilisateurs lors de déploiement, ou encore la cmdlet permettant de vérifier la connexion réseau.

Les cmdlets d'actions sont évidemment les cmdlets qui réaliseront la configuration d'une tâche à proprement parlé, comme par exemple créer un nouvel utilisateur dans l'annuaire d'un AD Windows ou encore configurer l'adresse d'un port de management sur un switch.

Ensuite dans chaque tag « Tests » et « Actions » sont définis les cmdlets appartenant à cette catégorie. La définition d'une cmdlet dans l'engine de YASC doit suivre certaines règles.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Prenons par exemple le cas de la cmdlet « Set-LDPyascAdPwd » qui permet de changer le password administrateur d'un active directory :

```
<ChangeAdminPwd>
    <CmdLet>Set-LDPyascAdPwd</CmdLet>
    <Param>
        <Name>newpwd</Name>
        <From>Step</From>
    </Param>
    <NeedsReboot>yes</NeedsReboot>
</ChangeAdminPwd>
```

Exemple 6 Cmdlet permettant de changer le mot de passe admin d'un A.D.

Il y a quatre règles à respecter dans le cadre de la définition d'une nouvelle cmdlet dans le fichier d'engine.

La première règle à suivre est de créer un tag XML relativement simple décrivant ce que réalise la cmdlet. Cela permettra de citer la cmdlet dans le fichier de déploiement (« AppData.xml ») et simplifie grandement l'écriture de celui-ci. Le but étant de simplifier le plus possible la lecture de ce fichier. Dans le cas de notre exemple, son « nom d'étape » est « ChangeAdminPwd ». C'est plutôt explicite.

La deuxième règle est de eître_citer entre le tag « CmdLet » le nom complet de la cmdlet. Ce tag sera utilisé lors de la génération du script de déploiement, permettant à YASC d'invoquer la cmdlet ainsi que tous les paramètres qui lui seront imputés depuis le fichier de déploiement.

La troisième règle consiste à ajouter le tag « NeedsReboot », toujours nécessaire, spécifiant si l'étape réaliser par la cmdlet nécessite un reboot de la machine. Les valeurs acceptées sont « yes » et « no ».

La quatrième règle est d'ajouter les tags « Param ». Ces tags définissent la provenance de la valeur à passer à un paramètre de la cmdlet lors de son invocation par YASC ainsi que son nom. Ainsi nous pourrons déclarer si la valeur du paramètre est à trouver dans le fichier de déploiement ou disponible dans le fichier des métadonnées. YASC réalise cela lui-même en utilisant le système de séparateurs, cités plus haut, pour savoir quand il doit remplacer une valeur par une trouvée dans un autre fichier. Pour le tag « From » on peut alors définir la valeur « Step », qui signifie que la valeur est directement insérée dans le fichier de déploiement, ou une référence dans le fichier des métadonnées :

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
<Param>
  <Name>Domain</Name>
  <From>Meta.AdDomains. ${$YBT.Environment}</From>
</Param>
```

Exemple 7 YASC ira chercher la valeur dans le fichier des métadonnées

Ceci indique à YASC d'aller chercher le nom de domaine dans le fichier des métadonnées au moment du « Build-time ».

4.2.4 Le fichier de déploiement d'application : AppData.xml

Ce fichier est appelé AppData.xml de façon générale. En réalité son nom dépend de l'application à déployer. Ainsi pour déployer Git, par exemple, le fichier s'appellera « Git.xml » et cela dans le but d'être le plus concis possible.

Ce fichier est certainement un des plus importants car il décrit toutes les étapes de déploiement d'une application. Ce fichier est chargé dans un objet PowerShell en dernier lieu une fois que tous les fichiers de configuration ont été chargés et que leur intégrité a été vérifiée.

Le fichier de déploiement respecte lui aussi une certaine architecture de construction : l'entièreté du fichier est englobée par le tag XML « Data ».

Ensuite, le tag « General » permettra de décrire l'application, ainsi que la version, qui sera déployée. Ces informations seront très utiles lors de la génération de la documentation.

```
<General>
  <AppName>Firewall</AppName>
  <AppVersion>1.0.0</AppVersion>
  <About>Automatically configure a firewall with yasc.</About>
</General>
```

Exemple 8 Tag "General"

Viens-Vient après le tag « Steps », contenant une foule de sous tag « Step ». Son but est évidemment d'inclure une série d'étapes au déploiement de l'application.

Comme montré dans la figure « [Exemple 9](#) » ci-dessous, le tag « Step » est un tag très rempli qui contient d'autres tags utiles à la génération de la documentation et du script de déploiement.

En réalité YASC, après avoir chargé en mémoire le fichier, va parcourir les différents tags « Step ». A chaque fois qu'il tombera sur le tag « Yasc », il récupère la valeur que le tag renferme. Cette valeur est en fait le nom de l'étape définie dans le fichier YascEngine.xml. Ainsi, lorsque YASC possède le nom de l'étape il peut récupérer facilement, depuis ce fichier,

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

le nom de la cmdlet et la provenance des paramètres qui la composent. Il sait ainsi, où récupérer les valeurs à passer aux paramètres.

Il y a alors deux possibilités : soit le paramètre trouve sa valeur dans le fichier des métadonnées et alors YASC n'a plus qu'à la récupérer depuis l'objet contenant en mémoire le fichier. Soit elle est spécifiée dans le tag « Step ».

Dans les deux cas, lorsque YASC parcourt le fichier de déploiement d'une application, il en résulte un script « .ps1 » composé de l'invocation d'une ou plusieurs cmdlet PowerShell chacune correspondant à des étapes de déploiement décrites dans ce même fichier. Ainsi lors de l'exécution du script, chaque cmdlet est exécuté une à une avec des valeurs paramétriques récupérées dans le fichier XML de déploiement et/OU des métadonnées.

```
<Steps>
  <Step>
    <Name>Configure a fortigate address</Name>
    <DetailedDescription>Configure a fortigate address in subnet 192.168.2.0/24</DetailedDescription>
    <Type>Action</Type>
    <Yasc>Fortigate-Address</Yasc>
    <Prereq>None</Prereq>
    <IsFatal>yes</IsFatal>
    <Param>
      <Name>s.lan</Name>
      <NetLan>192.168.2.0</NetLan>
      <NetLanMask>255.255.255.0</NetLanMask>
      <GUIColor>24</GUIColor>
      <Coll_OverPolicy>
        <NetLan>$true</NetLan>
        <NetLanMask>$true</NetLanMask>
        <GUIColor>$false</GUIColor>
      </Coll_OverPolicy>
    </Param>
  </Step>
</Steps>
```

Exemple 9 Les étapes de déploiement d'un firewall

4.2.5 En résumé

En résumé, lorsque l'on lance la création d'un script de déploiement avec YASC, celui-ci charge 4 fichiers.

Il commence par initialiser le premier fichier : le fichier de configuration, lui indiquant où trouver toutes les informations qu'il aura besoin par la suite (modules PowerShell, chemins d'accès, etc.). Une fois qu'il a toutes ces ressources, il peut charger les 3 autres fichiers qu'il sauvegardera dans des objets PowerShell à portée globale.

Il charge en dernier le fichier de déploiement propre à l'application demandée et parcourt chaque tag « Step » du fichier XML (par le biais de l'objet PowerShell). Pour chaque tag correspondant à une cmdlet définie dans le fichier YascEngine.xml, YASC ajoute au script une invocation de la cmdlet correspondante à l'étape. Ce script va alors, au moment de l'exécution, invoquer toutes ces cmdlets une à une.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.3 Les modules PowerShell

Les modules PowerShell de YASC sont des cmdlets utilisées par celui-ci pour réaliser une action ou un test. Elles ont pour fonction d'être le plus atomique et le plus simple possible. L'intérêt d'opter pour un mode de fonctionnement en module est de pouvoir améliorer YASC indépendamment de son propre code. En effet, dans le chapitre précédent, nous avons vu que YASC pouvait générer un script de déploiement. Ce script invoque l'une après les autres les cmdlets réalisant les étapes du fichier de déploiement d'une application quelconque.

Ainsi, il est évidemment très simple de programmer une nouvelle fonctionnalité et de l'ajouter dans YASC. Pour ce faire, il faut réaliser trois étapes :

- Programmer la cmdlet PowerShell en respectant certaines règles de bonne pratique.
- Déclarer la cmdlet dans le fichier YascEngine.xml
- L'ajouter au fichier XML de déploiement de l'application qui a besoin de cette nouvelle fonctionnalité.

4.3.1 Règles de bonne pratique pour la programmation d'un module PowerShell.

C'est assez simple de programmer un module PowerShell pour YASC. Cependant certaines règles de bonne pratique sont à observer lorsque l'on programme pour conserver une certaine cohérence avec le reste des cmdlets.

La toute première règle de bonne pratique à respecter est extrêmement importante car elle a un impact sur la génération automatique de la documentation. Il s'agit de l'en-tête de description de la cmdlet. Non seulement cet en-tête est utilisé pour la documentation mais il permet aussi de récupérer les informations relatives à ses paramètres, ses commentaires, etc. lors de l'utilisation de la cmdlet « Get-Help ». En somme, plus votre en-tête est complet, plus facile il sera de comprendre la cmdlet :

```
<#
.SYNOPSIS
Copy files.
.DESCRIPTION
Copy files.
#>
```

Exemple 10 Exemple d'en-tête de description

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

La seconde règle concerne les paramètres eux-mêmes de la cmdlet. Les paramètres des variables PowerShell doivent être le plus explicite possible et cela pour deux raisons. Premièrement, le nom des paramètres sera utilisé dans la documentation générée par YASC. De plus, la propriété « HelpMessage » sera utilisée pour décrire ce paramètre. Il est donc important que les deux soient clairs et précis. Deuxièmement, comme vu précédemment, le fichier YascEngine.xml précise d'où proviennent les valeurs des paramètres pour chaque cmdlet. De ce fait le nom de chaque paramètre utilisé dans la cmdlet sera référencé dans le fichier d'engine. Il est donc important, pour la compréhension de ce fichier, que les noms soient explicites.

```
param (
    [Parameter(Mandatory = $True, HelpMessage = 'AD Domain')] [string]$Domain,
    [Parameter(Mandatory = $True, HelpMessage = 'Username prefix')] [string]$UserNamePrefix,
    [Parameter(Mandatory = $True, HelpMessage = 'Username')] [string]$UserName,
    [Parameter(Mandatory = $True, HelpMessage = 'Abbreviations to use for long user names')] $Coll_Abbreviations,
    [Parameter(Mandatory = $True, HelpMessage = 'Organization Unit')] [string]$OU,
```

Exemple 11 Exemple de nomination des paramètres (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Ensuite, un module YASC nécessite toujours une variable « Mode ». Cette variable permet de spécifier le mode de fonctionnement du module. Une cmdlet PowerShell destinée à fonctionner sous YASC doit toujours, dans la mesure du possible, implémenter au minimum un mode « Create », un mode « Audit » et un mode « GetDoc ». Il se peut aussi que la cmdlet implémente un mode « Update » mais c'est plus rare.

```
[Parameter(Mandatory = $True, HelpMessage = 'Yasc Mode')] [ValidateSet('Create', 'Update',
'Audit', 'GetDoc')] [string]$Mode,
```

Exemple 12 Syntaxe du paramètre "Mode" de YASC (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Le mode « GetDoc » permet de récupérer sous format HTML, la documentation de la cmdlet. Pour ce faire, la cmdlet doit respecter une règle supplémentaire. Elle doit commencer par le code ci-dessous :

```
#Mandatory : Set $Me / Return html-formatted help if asked for
$Me = $MyInvocation.MyCommand
if ($Mode -eq 'GetDoc')
{
    return (Get-LDPyascParametersDoc -Command $Me -Parameters $MyInvocation.BoundParameters)
```

Exemple 13 Code de démarrage d'une cmdlet (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Ce code réalise tout simplement l'appel à la fonction « Get-LDPyascParametersDoc » qui renvoie le code HTML de la documentation pour cette cmdlet. Il faut évidemment passer la valeur « GetDoc » au paramètre de « mode ».

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Le mode « Audit » et le mode « Create » sont extrêmement importants. Le mode « Create » permet tout simplement d'exécuter la tâche de la cmdlet. Par exemple, placer un utilisateur Windows dans un groupe d'utilisateurs. Le mode « Audit » est extrêmement important. En effet, idéalement, la configuration appliquée par un script de déploiement crée par YASC ne doit JAMAIS être modifiée à la main.

Le mode « Audit » de chaque cmdlet a pour but de vérifier si la configuration actuelle de la tâche est toujours la même que celle appliquée lors du mode « Create ». Pour reprendre l'exemple précédent, le mode « Audit » de cette même cmdlet permettrait de vérifier si l'utilisateur appartient toujours bien au groupe spécifié dans le fichier de déploiement. Le mode « Audit » n'est pas toujours forcément facile à implémenter. Selon le champ d'activité de la cmdlet, il se peut qu'il soit impossible à créer. Cependant, la cmdlet doit quand même gérer le mode et renvoyer un message d'erreur indiquant à l'utilisateur que cette étape ne peut être auditée.

En quatrième lieu, la cmdlet doit posséder un paramètre « Logger » qui est une collection d'objets Log4net. Cet objet permet à la cmdlet d'enregistrer les logs sur un support variable, c'est-à-dire dans un fichier, base de données, par mail, etc.

Et pour finir, la cmdlet doit avoir un système de gestion d'erreurs. Cela débute avec un objet « Status » qui possède plusieurs variables membres. La première des variables est la variable de retour, c'est un simple booléen définissant le succès de la cmdlet. Ensuite viennent trois tableaux. Ces tableaux ont pour objectif de contenir les différents logs. En effet, il y a trois niveaux de log : le niveau d'info, le niveau des « warnings » et le niveau des erreurs.

```
#Mandatory : Prepare return object
$Status = New-Object System.Object
$Status | Add-Member -MemberType NoteProperty -Name RetVal -Value $False
$Status | Add-Member -MemberType NoteProperty -Name Info -Value @()
$Status | Add-Member -MemberType NoteProperty -Name Warn -Value @()
$Status | Add-Member -MemberType NoteProperty -Name Error -Value @()
```

Exemple [J44](#) Objet "Status" (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Peu importe le support utilisé pour sauvegarder les logs, la réalisation de ces derniers se fait toujours de la même manière d'un point de vue programmation :

Mis en forme : Français (Belgique)

Mis en forme : Français (Belgique)

Code de champ modifié

Mis en forme : Français (Belgique)

Code de champ modifié

Mis en forme : Français (Belgique)

Mis en forme : Français (Belgique)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
$Status.Info += $Logger | Write-LDPLogger -Severity Info -Message "Information message example" -Invocation SME
```

OU :

```
$Status.Warn += $Logger | Write-LDPLogger -Severity Warn -Message "warning message example" -Invocation SME
```

OU :

```
$Status.Error += $Logger | Write-LDPLogger -Severity Error -Message "Error message example" -Invocation SME
```

Exemple 15 Logger une info dans une cmdlet (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Ainsi, l'objet « Log4net » assure que l'information soit sauvegardée correctement, que le script appelant l'information puisse gérer l'échec ou la réussite de la cmdlet, affichant alors de façon globale le nombre d'erreurs ou de « warnings » produits ainsi que les messages leurs étant liés.

4.3.2 Déclarer une cmdlet dans le fichier YascEngine.xml

La seconde étape dans l'intégration d'un module PowerShell est de déclarer la cmdlet dans le fichier de l'engine, c'est-à-dire le fichier YascEngine.xml. Dans le chapitre présentant les différents fichiers de configuration de YASC, la syntaxe du fichier YascEngine.xml a déjà été abordée cependant, un exemple complet et une explication du cas des collections de données semblent nécessaires.

De façon générale, la déclaration d'une cmdlet dans YascEngine.xml peut avoir cette forme :

```
<Add-WindowsFeature>
    <CmdLet>Add-IDPyscWindowsFeature</CmdLet>
    <Param>
        <Name>FeatureList</Name>
        <From>Step</From>
    </Param>
    <NeedsReboot>yes</NeedsReboot>
</Add-WindowsFeature>
```

Exemple 16 Cmdlet déclarée dans le fichier YascEngine.xml

Le fichier YascEngine.xml associe donc n'importe quelle étape/action susceptible d'être utilisée dans un fichier de déploiement à une cmdlet qui réalise cette action (grâce au tag « CmdLet »). Dans cet exemple, nous avons déclaré une étape permettant d'ajouter une « feature » Windows. Nous avons donc, arbitrairement, choisi d'appeler cette étape « Add-WindowsFeature ». À noter que le choix de ce nom est totalement dépendant du programmeur,

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

cependant c'est sous ce nom que l'étape sera référencée dans le fichier de déploiement. C'est ainsi que YASC fait la liaison entre l'étape et la cmdlet.

Par la déclaration des variables de la cmdlet, le tag « Param » est composé de deux tags : le tag « Name » et le tag « From ». Ce dernier peut référencer le fichier de déploiement ou le fichier des métadonnées comme source de la valeur du paramètre. Mais de cela nous avons déjà parlé dans le chapitre 4.2.4. Ainsi discutons du cas des collections de données dans YASC.

Un cas particulier lors de la déclaration des paramètres d'une cmdlet dans le fichier engine est l'utilisation d'une collection de données. Pour passer, en tant que paramètre, une collection de données on utilise une syntaxe bien spécifique pour le nom du paramètre. En effet, cela permettra à YASC de créer un objet Hashmap² contenant les données récupérées dans le fichier XML. Dans YASC, toute collection de données passées en paramètres doit être nommée de cette manière : « \$Coll_NomDeLaCollection ».

```
<Param>
    <Name>Coll_Search_Replace</Name>
    <From>Step</From>
</Param>
```

Exemple 17 Déclaration d'une collection dans YascEngine.xml

Comme pour des variables classiques, pour passer des valeurs à cette collection il est possible soit de les récupérer dans le fichier de déploiement de l'application ou dans le fichier des métadonnées.

Prenons par exemple une collection de données contenant un ensemble d'abréviations. Ainsi, pour déclarer les valeurs dans le fichier des métadonnées :

² Une Hashmap est une collection de données qui dont les données sont organisées selon des clés et des valeurs. Une valeur s'obtient à partir de la clé.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
<Abbreviations>
  <Coll_Abbreviations>
    <Long>WebSales</Long>
    <Short>WS</Short>
  </Coll_Abbreviations>
  <Coll_Abbreviations>
    <Long>Simulation</Long>
    <Short>Sim</Short>
  </Coll_Abbreviations>
  <Coll_Abbreviations>
    <Long>Portima</Long>
    <Short>Ptm</Short>
  </Coll_Abbreviations>
  <Coll_Abbreviations>
    <Long>WebService</Long>
    <Short>WebSvc</Short>
  </Coll_Abbreviations>
  <Coll_Abbreviations>
    <Long>External</Long>
    <Short>Ext</Short>
  </Coll_Abbreviations>
  <Coll_Abbreviations>
    <Long>IndividualLife</Long>
    <Short>IndLife</Short>
  </Coll_Abbreviations>
</Abbreviations>
```

Exemple 18 Déclaration d'une collection dans le fichier YascMeta.xml

Le tag « Abbreviations » servira purement est simplement de balise pour spécifier dans le fichier YascEngine.xml **ou où** trouver la collection. Le plus important pour YASC sont les tags « Coll_Abbreviations ». Ces tags contiennent la clé et la valeur utilisées pour un élément de la Hashmap.

Si l'on souhaite déclarer les valeurs des données au sein du fichier de déploiement de l'application, la syntaxe est plus ou moins la même que dans le fichier YascMeta.xml, les tags « Coll_xxxx » seront encadrés par le tag « Param » et pour chaque nouvelle entrée à ajouter dans la Hashmap, correspondra l'ouverture d'un tag « Coll_xxxx » :

```
<Param>
  <SourceFile>{[Meta.ConfigRepository.$($YBT.Environment)}\TotalCommander\wrx_ftp.ini</SourceFile>
  <TargetFile>C:\TotalCmd\wrx_ftp.ini</TargetFile>
  <Coll_Search_Replace>
    <Search>{[IpAddress]}</Search>
    <Replace>{[Meta.Mainframe.$($YBT.Environment).IpAddress]}</Replace>
  </Coll_Search_Replace>
  <Coll_Search_Replace>
    <Search>{[CicsPort]}</Search>
    <Replace>{[Meta.Mainframe.$($YBT.Environment).CicsPort]}</Replace>
  </Coll_Search_Replace>
</Param>
```

Exemple 19 Déclaration de la collection dans le fichier de déploiement "AppData.xml"

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Pour chaque tag « Coll_xxxx », on ajoutera un ou plusieurs tags. Le nom du tag donnera le nom de la clé dans la Hashmap et le contenu du tag spécifiera la valeur correspondante à cette clé.

4.3.3 Utilisation de la cmdlet dans le fichier de déploiement de l'application

Maintenant que notre cmdlet a été créée en suivant les règles minimales nécessaires pour son bon fonctionnement dans YASC et qu'elle a été intégrée au fichier d'engine de YASC, il faut l'utiliser dans le fichier de déploiement d'application.

Nous pouvons diviser un fichier de déploiement en deux parties. La première partie est composée du tag « General ». Ce tag permet de décrire l'application à déployer. La deuxième partie est composée du tag « Steps ». C'est ce tag qui accueille toutes les étapes de déploiement. Comme déjà expliqué, à chaque étape correspond un tag « Step ».

Comme on peut le voir sur l'exemple ci-dessous. Une étape se déclare entre le tag et l'anti-tag « Step ». Celui-ci en contient d'autres, chacun ayant sa propre importance :

```
<Data>
  <General>
    <AppName>switchTest</AppName>
    <AppVersion>1.0.0</AppVersion>
    <About>Automatically configure a switch with yasc.</About>
  </General>
  <Steps>
    <Step>
      <Name>Create vlan 30 and configure its name.</Name>
      <DetailedDescription>Create vlan 30 and set its name to "AuthVlan".</DetailedDescription>
      <Type>Action</Type>
      <Yasc>Switch-VlanName</Yasc>
      <Prereg>None</Prereg>
      <IsFatal>no</IsFatal>
      <Param>
        <VlanName>AuthVlan</VlanName>
        <VlanNumber>30</VlanNumber>
      </Param>
    </Step>
  </Steps>
</Data>
```

Exemple 20 Configuration d'un vlan sur un switch HP ProCurve

- Le tag « Name » : sert à la documentation du déploiement.
- Le tag « DetailedDescription » : sert également à la documentation. Il fournit une description détaillée de l'étape et de ce qu'elle produit.
- Le tag « Type » : le fichier YascEngine est séparé en deux catégories : les tests et les actions. Ce tag permet de dire à YASC dans quelle catégorie se situe la cmdlet.
- Le tag « Yasc » : sûrement un des tags les plus importants, le tag « Yasc » définit le nom de l'étape. C'est grâce à cela que YASC va récupérer la cmdlet correspondant à l'étape dans le fichier YascEngine.xml.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

- Le tag « Prereq » : permet de spécifier si une étape est nécessaire avant la réalisation de celle-ci.
- Le tag « IsFatal » : permet de terminer l'exécution de YASC dans le cas où l'étape se serait terminé sans succès. Par exemple il n'est pas intéressant de configurer les informations d'authentification R.A.D.I.U.S d'un switch si la configuration du serveur n'est pas déroulée correctement.
- Le tag « Param » : ce tag contient d'autres sous-tags nommés chacun selon les paramètres de la cmdlet et contenant les valeurs leurs devant être assignés lors de l'exécution. Cependant, il se peut que ce tag ne contienne pas tous les paramètres. Pour rappel, il est possible de récupérer les valeurs depuis le fichier des métadonnées.

4.3.4 En résumé

En conclusion, pour ajouter une nouvelle cmdlet à YASC, il suffit de respecter quelques règles de bonnes pratiques, comme la gestion des erreurs, l'ajout des modes principaux d'une cmdlets, la description de la cmdlet pour la documentation, etc.

Il faut aussi ajouter à l'engine la description de la cmdlet et l'étape associée. La seule petite difficulté dans cette étape peut se trouver dans la façon dont fonctionne tout le système de paramétrage. En effet, au début il peut paraître un peu déroutant de pouvoir spécifier la valeur d'un paramètre depuis ... ~~un à~~ peu près n'importe où. Cependant, ce système permet à YASC de réaliser sa tâche avec le plus de souplesse possible.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.4 En sortie du processus YASC

Pour rappel, YASC ne réalise pas le déploiement à proprement parler. Il construit en réalité un autre script PowerShell réalisant le déploiement. En fonction de l'application à déployer le script pourra s'utiliser de différentes manières : soit directement sur la machine, soit par connexion SSH, etc.

Cependant, ce n'est pas la seule résultante de l'exécution de YASC. Comme nous pouvons le voir sur le schéma de fonctionnement de YASC à la page 35, l'engine réalise aussi une documentation complète du déploiement, envoie un mail à la personne responsable et réalise une copie complète du contexte d'exécution (copie des fichiers de configurations).

Pour décrire plus en détails le script de déploiement, son schéma de fonctionnement est disponible à la page suivante.

4.4.1 Schéma de fonctionnement d'un script de déploiement

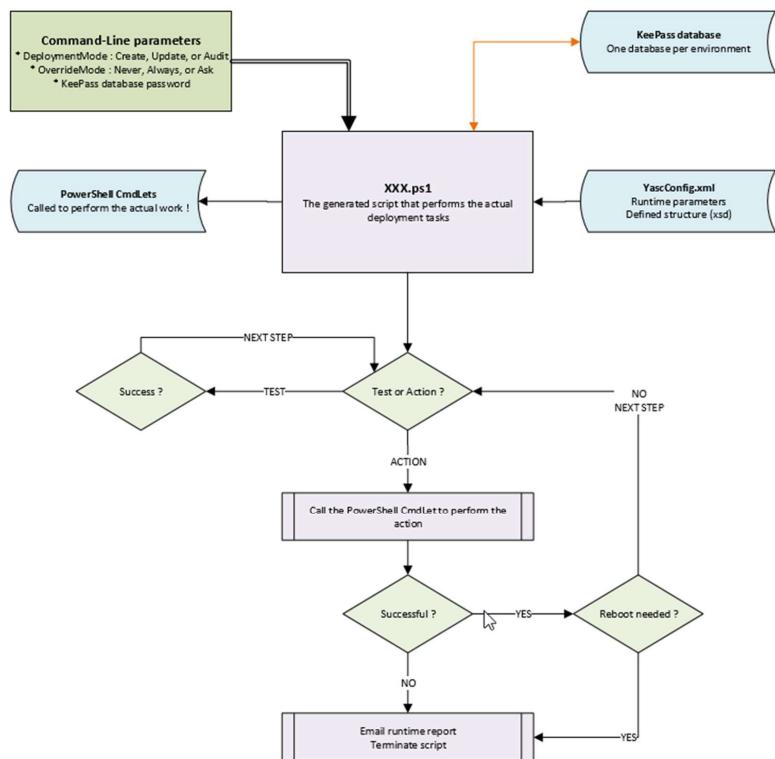


Figure 19 Schéma de fonctionnement d'un script de déploiement (YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION, 2016)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.4.2 Le script de déploiement

4.4.2.1 Les paramètres

Sur ce schéma nous pouvons voir en premier lieu que, tout comme une cmdlet PowerShell, le script prend des paramètres. En effet, le script de déploiement peut être appelé depuis un autre script PowerShell ou depuis l'invite de commande PowerShell comme une simple CmdLet.

Ainsi, le script de déploiement généré par YASC possède une section « Param » décrivant tous les paramètres nécessaires à son exécution :

```
<#
.SYNOPSIS
Install / Configure / Audit Application Firewall Version 1.0.0 for environment Dev
This script was generated by Yasc (c) Limelogic 2013
Script generated by Administrator on 2017-04-11_15-02-49
#>
param(
[Parameter(Mandatory = $true, HelpMessage = "Deployment mode (Create, Update or Audit)")][ValidateSet('Create', 'Update', 'Audit')][string]$DeploymentMode,
[Parameter(Mandatory = $true, HelpMessage = "Override mode (Never, Always, or Ask)")[ValidateSet('Never', 'Always', 'Ask')][string]$OverrideMode,
[Parameter(Mandatory = $false, HelpMessage = "Password of your keepass database")][string]$keepassDatabasePw,
[Parameter(Mandatory = $false, HelpMessage = "Shadow copy error is fatal")][bool]$shadowErrorIsFatal = $true,
[Parameter(Mandatory = $false, HelpMessage = "Return an object or an exit code")][ValidateSet('Object', 'ExitCode')][string]$returnset = 'Object',
[Parameter(Mandatory = $false, HelpMessage = "Root path of gpm (Global PowerShell Modules) folders")][string]$gpmRootDir = 'C:\PowerShell\gpm',
[Parameter(Mandatory = $false, HelpMessage = "Root path of Yasc folders")][string]$yascRootDir = 'C:\PowerShell\LPS\yasc',
[Parameter(Mandatory = $false, HelpMessage = "Root path of config folders")][string]$cfgRootDir = 'C:\PowerShell\LPS'
)
#Script generated by Administrator - 2017-04-11_15-02-49
```

Figure 20 En-tête "Param" d'un script de déploiement généré par YASC dans le cadre de la configuration d'un firewall.

Le paramètre « DeploymentMode » fait évidemment référence au paramètre mode que chaque Cmdlet doit posséder (c.f page [4144](#)). Chaque cmdlet appelée au sein de ce script de déploiement aura comme mode de fonctionnement celui passé par l'utilisateur au moment de lancer le script. Ainsi, si l'utilisateur souhaite vérifier si la configuration respecte toujours bien celle créée, il n'aura qu'à lancer le script en mode Audit. Cela aura pour effet de lancer chaque cmdlet en mode Audit.

Le paramètre « OverrideMode » est un peu spécial. En effet, lors de la première exécution du script de déploiement, après la réalisation avec succès d'une étape, le script écrit dans le registre Windows, si l'action a été réalisée. Cela permet de conserver un historique des actions déjà réussies pour ne pas devoir les refaire si jamais une étape fatale échoue et que le script doit s'arrêter. De ce fait, avant de réaliser une étape, le script vérifie d'abord dans les registres Windows si elle n'a pas déjà été réalisée. Le paramètre « OverrideMode » peut prendre plusieurs valeurs :

- « Never » : permet de spécifier que le script ne réalisera jamais deux fois une action qui a été marquée comme réalisée dans les registres Windows.
- « Always » : permet d'obliger le script à toujours réaliser l'étape même si elle a déjà été faite.
- « Ask » : permet de laisser le choix, en temps réel, à l'utilisateur.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Le paramètre « KeePassDataBasePwd » est lui aussi assez singulier. En effet, la gestion des mots de passe utilisateur se réalise via le software KeePass. Il est évident que pour accéder à ce logiciel il faut un mot de passe. Le paramètre « KeePassDataBasePwd » représente sous forme de string le mot de passe du KeePass.

Viennent ensuite les paramètres contenant les chemins vers les dossiers critiques de YASC : « GpmRootDir » permettant de spécifier le chemin vers les modules PowerShell globaux de YASC, « YascRootDir » permettant de spécifier le chemin vers le répertoire par défaut de YASC et enfin « CfgRootDir » permettant de spécifier le chemin du dossier « cfg » qui contient les quatre fichiers XML de configuration de YASC.

4.4.2.2 L'exécution et les emails

Comme cité précédemment, le script de déploiement contient une suite d'appel à des cmdlets. Ces cmdlets sont importées, au moment de la création du script de déploiement, par YASC. Au moment de l'exécution, chaque cmdlet parcourue sera exécutée suivant cette logique :

- I) Si la cmdlet fait partie de la catégorie « test », on réalise le test :
 - 1) S'il est réussi on passe à la cmdlet suivante.
 - 2) Sinon on termine l'exécution du script si nécessaire (fatale ou pas).
- II) Si la cmdlet fait partie de la catégorie « action », on réalise l'action en appelant la cmdlet.
 - 1) Si elle est réussie, on vérifie si un reboot est nécessaire.
 - o Si le reboot est nécessaire on stop le script de déploiement et on envoie un email à la personne responsable de l'environnement dans lequel on déploie.
 - o Sinon on passe à la cmdlet suivante.
 - 2) Si l'étape n'est pas réussie, on vérifie si elle doit être fatale pour le script ou pas. Si elle l'est on termine l'exécution du script de déploiement et on envoie également un email.

Ainsi, une fois que toutes les cmdlets ont suivies cette logique d'exécution, le script de déploiement finis-finit par envoyer un mail à toute personne concernée par le déploiement de l'application. Plusieurs étapes sont nécessaires pour l'envoi. Les informations de déploiement, comme les adresses emails des responsables, étant dépendantes du client, il est nécessaire de fournir au script ces informations au moment du « Runtime ». Ces informations sont pour la plupart contenues dans le fichier de configuration YascConfig.xml et stockées dans l'objet « YRT » au début du script. Ce sont ces informations qui seront utilisées en toute fin pour envoyer un ou plusieurs mails aux personnes concernées.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Ainsi au début d'exécution d'un script de déploiement généré par YASC on commence par charger le fichier de configuration en mémoire, YascConfig.xml. Ensuite on crée l'objet YRT et on lui fournit :

- L'objet XML permettant de parcourir le fichier de configuration,
- Le mode de déploiement,
- Le nom de l'environnement de déploiement (Dev, Tst, etc.),
- L'objet Log4net pour continuer à logger des informations.

```
#Load the configuration file
$OneStatus = Initialize-LDPMycConfigXml -ConfigXml "$CfgRootDir\cfg\YascConfig.xml" -Logger $RollingLogger
$Status .Info += $OneStatus .Info
$Status .Warn += $OneStatus .Warn
$Status .Error += $OneStatus .Error
if (-not $OneStatus .RetVal) {
    switch ($ReturnSet) {
        'Object' {return $Status}
        'ExitCode' {return 3}
    }
}
$SC = $OneStatus .RetVal

#Runtime parameters
$YRT = New-Object System.Object
$YRT | Add-Member -MemberType NoteProperty -Name DeploymentMode -Value $DeploymentMode
$YRT | Add-Member -MemberType NoteProperty -Name Environment -Value Dev
$YRT | Add-Member -MemberType NoteProperty -Name Logger -Value $RollingLogger
$YRT | Add-Member -MemberType NoteProperty -Name Config -Value $C .Config
```

Exemple 21 Chargement du fichier YascConfig.xml et création de l'objet YRT

C'est grâce à l'information contenue dans la variable membre « Config » que l'on peut parcourir le fichier XML et donc récupérer les informations nécessaires à l'envoi d'un mail. Ainsi, à la fin du script de déploiement, après que toutes les cmdlets aient été invoquées, le code ci-dessous est appelé :

```
$EmailSubject = "Yasc DEPLOY : Firewall/1.0.0/Dev"
$Dummy = Send-LDPEmail `
    -EmailSender $($YRT .Config .Reporting .Dev .EmailSender) `
    -EmailRecipients $($YRT .Config .Reporting .Dev .EmailRecipients) `
    -EmailSubject $EmailSubject `
    -EmailBody $EmailDeployBody `
    -EmailAttachments $RollingLogFile `
    -SmtpServerName $($YRT .Config .Reporting .Dev .SmtpServer) `
    -BodyAsHtml:$true `
    -Logger $YRT .Logger
```

Exemple 22 Envoi d'un email à la fin du script de déploiement

L'objet « Config », dans l'objet « YRT », possède une variable membre correspondant au tag « racine » du fichier XML. Cette variable possède elle aussi une variable pour chaque sous-tag et ainsi de suite jusqu'à trouver la valeur recherchée.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

4.4.3 La documentation et la copie du contexte d'exécution

Cela fait plusieurs chapitres que l'on parle de la documentation. C'était même l'argument principal qui justifiait la conception de YASC. Il est donc temps d'en parler.

À contrario de l'envoi d'un email de notification dans YASC, la génération de la documentation technique correspondante au déploiement d'une application est effectivement bien le résultat de l'exécution de YASC, exactement comme le script de déploiement. En réalité, lorsque YASC crée le script de déploiement, il réalise l'analyse de chaque étape du fichier de déploiement et donc de chaque cmdlet ainsi que leurs paramètres et leurs valeurs. C'est à ce moment-là que YASC réalise la documentation du déploiement sous format HTML.

En premier lieu, YASC réserve un objet rien que pour la création de la documentation. Cet objet contiendra des variables membres permettant de créer de manière indépendante l'en-tête, le corps et le pied de page du fichier HTML.

```
#Will hold generated script and generated documentation
$Code=@()
$DocHeader=@()
$DocBody=@()
$DocFooter=@()

$YascResult = New-Object System.Object
$YascResult | Add-Member -MemberType NoteProperty -Name Code -Value $Code
$YascResult | Add-Member -MemberType NoteProperty -Name DocHeader -Value $DocHeader
$YascResult | Add-Member -MemberType NoteProperty -Name DocBody -Value $DocBody
$YascResult | Add-Member -MemberType NoteProperty -Name DocFooter -Value $DocFooter
```

Exemple 23 Objet contenant sous forme textuelle la documentation

Ensuite, YASC va réaliser la construction de l'en-tête de la documentation. Pour ce faire YASC utilise une cmdlet : "New-LDPyascScriptHeader". Cette fonction permet de créer l'en-tête de la documentation en fonction de l'environnement d'exécution, du mode de déploiement, etc. Il y a alors ensuite quelques informations HTML purement statiques qui sont rajoutées à la main ainsi que du code CSS.

Le corps de la documentation, c'est-à-dire la description de chaque étape, les valeurs passées aux paramètres, le nom de la cmdlet correspondante, etc. sont construits au moment de la construction du script de déploiement. En réalité, lorsqu'il aura reconstruit l'appel complet des cmdlets pour chaque étape, YASC va appeler chaque cmdlet en mode « GetDoc ». Ce mode appelle une fonction qui permet de renvoyer un bout de code HTML propre à chaque cmdlet et décrivant ce qu'elle réalise et les valeurs qu'elle utilise. En définitive, c'est lors de la construction des cmdlets utilisés dans le script de déploiement que YASC crée la plus grosse partie de la documentation.

Pour terminer YASC va rajouter le pied de page HTML grâce à quelques portions de code statique.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

À titre d'exemple, voici la documentation générée pour une application permettant de créer une adresse sur un firewall FortiGate de chez Fortinet :

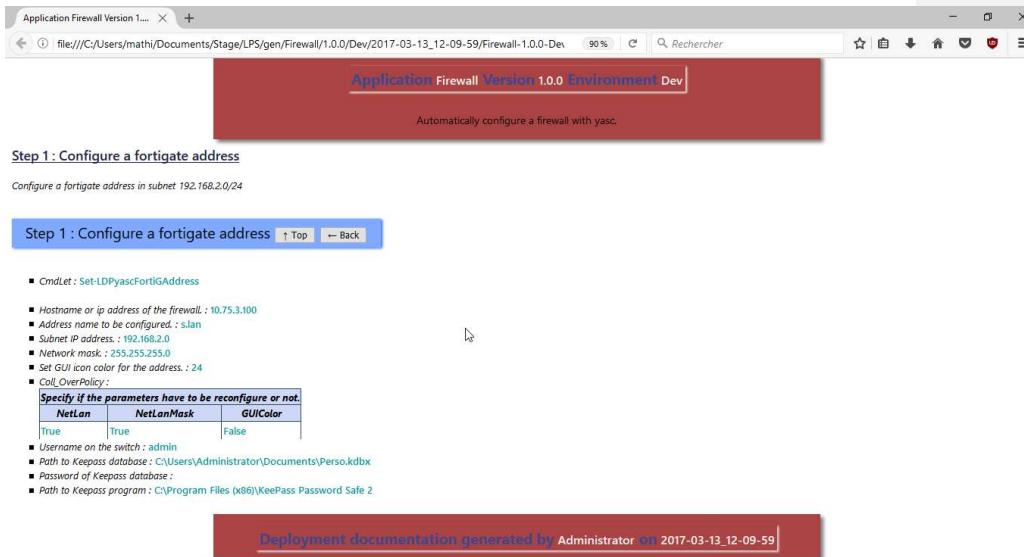


Figure 21 Documentation d'un déploiement à une étape

On constate que presque toutes les informations qui ont été passées en paramètre à YASC ont été utilisées : le « Timestamp », le nom de l'application, la version, l'environnement de déploiement, etc. En plus des informations qu'il a récupérées lui-même comme le nom de l'utilisateur qui a déployé l'application ainsi qu'évidemment la cmdlet utilisée pour l'étape et les valeurs ayant été imputées à ses paramètres.

En parallèle à la génération de la documentation, YASC réalise ce qu'on appelle « une copie du contexte d'exécution ». Cela consiste à copier les quatre fichiers de configuration XML dans le but de garder en mémoire le contexte dans lequel YASC a créé le script de déploiement. En effet, il sera souvent nécessaire de modifier certains fichiers XML pour améliorer ou compléter YASC. Cela peut engendrer le disfonctionnement-dysfonctionnement de certains scripts de déploiement créés dans le passé. Il est alors intéressant de réaliser une différence entre les nouveaux fichiers XML et leurs versions à l'époque où le(s) script(s) fonctionnai(en)t encore. C'est une sorte de sécurité pour celui qui devra développer sous YASC ou tout simplement rajouter des cmdlets/fonctionnalités.

4.5 En résumé

Pour conclure ce chapitre, récapitulons une brève fois ce que nous savons sur YASC.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

YASC est un engine, qui a pour but de réaliser un script écrit en PowerShell permettant de déployer une application. Pour ce faire il utilise quatre fichiers XML :

- Le fichier de configuration qui contient toutes les informations nécessaires à YASC pour s'exécuter.
- Le fichier des métadonnées contenant toutes les informations propres à l'environnement d'exécution du client.
- Le fichier de l'engine qui contient en réalité une association logique entre le nom d'une cmdlet (ainsi que ses paramètres) et une étape.
- Le fichier de déploiement de l'application qui décrit toutes les étapes nécessaires au déploiement de l'application.

Pour réaliser ce script de déploiement, YASC le construit toujours à partir d'une base commune (paramètres, vérification du chemin pour enregistrer le script définitif, etc.). Ce « template » commun il le remplit au fur et à mesure d'appel vers des cmdlets. Pour ce faire, il analyse le fichier de déploiement de l'application. À l'intérieur il y trouve des noms d'étapes ainsi que les paramètres et leurs valeurs dont elles ont besoin. Pour chaque étape, il parcourt le fichier de l'engine et récupère le nom de la cmdlet correspondante à l'étape. Une fois qu'il a récupéré le nom de la cmdlet pour une étape et les valeurs des paramètres il ajoute au script un appel vers cette cmdlet puis il passe à l'étape suivante. Ces cmdlets sont atomiques et relativement faciles à programmer, ainsi n'importe qui s'y connaissant un minimum en PowerShell et ayant pris connaissances des chapitres précédent pourrait créer une cmdlet pour YASC. Pour finir, le script de déploiement envoie un ou plusieurs mails aux personnes en charge de l'environnement dans lequel on déploie.

Pour conclure, YASC est un script PowerShell permettant de générer les ressources nécessaires pour déployer une application, à configurer un équipement réseau, à configurer un serveur d'annuaire d'utilisateur ainsi que bien d'autres applications et services. Il fournit la documentation propre à chaque déploiement ainsi qu'une copie du contexte d'exécution dans l'optique de faciliter la vie des techniciens en cas de dépannages. De plus, il assure la vérification de chaque configuration appliquée après déploiement via son mode « Audit » et notifie tout déploiement aux personnes en charge de l'environnement pour éviter les modifications non voulues.

Voyons comment utiliser tout ceci pour créer des cmdlets de configuration pour un switch HP ProCurve et un firewall Fortigate de chez Fortinet.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

5 Développement de modules de gestion d'un switch HP.

Durant ce stage, il a été demandé de réaliser deux tâches bien concrètes. La première était de réaliser la configuration complète d'un switch HP ProCurve via l'outil qui vient d'être présenté : YASC.

Peu importe le moyen qui doit être utilisé pour configurer le switch, le but étant d'automatiser la configuration de ce matériel en interagissant le moins possible avec celui-ci.

Ce chapitre abordera les différences notables entre un switch HP et un switch Cisco avec lequel l'on a le plus l'habitude de travailler dans le cadre des études d'analyste programmeur à la haute école de la province de Liège. Il expliquera aussi les moyens qui ont été déployés pour configurer de tel switch via YASC. C'est-à-dire les différents modules utilisés, les problèmes rencontrés ainsi que les solutions trouvées.

5.1 HP ProCurve et switch Cisco.

HP ProCurve désignait autrefois la division réseau de la société Hewlett-Packard de 1998 à 2010. Elle était évidemment associée à leur produit phare : le switch HP ProCurve. Le nom de la division a depuis lors été renommé en « HP Networking » cependant, le switch a conservé son nom jusqu'à récemment. La marque HP a décidé de renommer toute sa gamme de switch en « Aruba switches ».

Durant ce chapitre, nous aborderons comment le déploiement d'un switch HP a été réalisé avec YASC. Evidemment pour ce faire il a fallu au préalable réaliser l'étude d'une configuration HP préexistante. De cette analyse sont ressorties des remarques et des constatations quant aux différences qu'il peut exister entre les switches Cisco, que l'on a plus l'habitude de manipuler lors des laboratoires réseaux, et les switches HP.

La suite de ce chapitre traitera de plusieurs points de la configuration d'un switch HP. Pour chaque élément de configuration qu'il a été nécessaire de configurer sur le switch un point de comparaison sera réalisé avec un système d'exploitation de chez Cisco : Cisco IOS.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

5.2 Configurer un switch HP ProCurve grâce à YASC

Pour rappel, pour réaliser une tâche avec YASC il convient de la diviser en étapes atomiques. À chacune des étapes correspond une cmdlet réalisant une tâche le plus simple possible. Il est relativement simple de saisir ce concept dans le cadre de la configuration d'un switch.

Pour configurer un switch plusieurs étapes peuvent être nécessaires en fonction du contexte. On peut avoir besoin de configurer une interface de management avec une adresse IP et donc un vlan de management correspondant. On peut avoir besoin de configurer le protocole SNMP, de configurer 802.1X ainsi que bien d'autre fonctionnalités. Il est donc évident que pour chacune de ces fonctionnalités une cmdlet sera créée.

Il reste maintenant à savoir comment communiquer avec le switch par PowerShell. Pour accéder au switch en PowerShell, la solution la plus évidente et la plus simple a été de réaliser une connexion SSH sur ce switch.

5.2.1 L'accès par SSH

En PowerShell pour réaliser une connexion SSH il convient d'utiliser un module complémentaire appelé « Posh-SSH ». Ce module permet, via PowerShell, de réaliser une session SSH avec la cmdlet « New-SSHSession », d'envoyer une commande sous forme de string via la cmdlet « Invoke-SSHCommand », de récupérer une session SSH ouverte via « Get-SSHSession » et enfin de supprimer une session ouverte via « Remove-SSHSession ». Il n'y a rien de plus simple. A savoir que Posh-SSH contient aussi beaucoup de commandes supportant différents protocoles comme SFTP, SCP, etc. Nous en reparlerons plus tard car nous en aurons besoin.

Les premières tentatives ont consisté à réalisés-réaliser à la main des connexions SSH vers un switch HP et d'envoyer les commandes par la cmdlet « Invoke-SSHCommand ». Cependant, les commandes ne se réalisaient jamais et il a fallu passer par une étape intermédiaire.

En effet, voici la technique qui aurait été la plus simple d'utiliser :

```
#Création de la commande
$Cmd = "conf t
vlan 10
name VlanMgmt
wr mem n"

#Initier la connexion SSH
$SessionID = New-SSHSession -ComputerName "10.74.1.1" -Credential $switchCredential -AcceptKey -Force
#Envoyer la commande
Invoke-SSHCommand -Index $SessionID -Command $Cmd
#Fermer la connexion
Remove-SSHSession |
```

Exemple 24 Création d'une session SSH et envoi d'une commande

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Malheureusement le switch HP n'avait pas l'air d'accepter les commandes envoyées par la cmdlet « Invoke-SSHCommand ». Il a donc fallu passer par un objet intermédiaire permettant de créer un flux :

```
New-SSHSession -ComputerName $SwitchIPAddr -Credential $mycreds -AcceptKey -Force  
$session = Get-SSHSession -SessionId 0  
$stream = $session.Session.CreateShellStream("xterm", 1000, 1000, 1000, 1000, 1000)  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("")  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("conf t")  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("vlan $VlanNumber")  
Sleep 3  
$stream.WriteLine("name $VlanName")  
Sleep 2  
$stream.WriteLine("wr mem")  
Sleep 1  
Remove-SSHSession -SessionId 0
```

Exemple 25 Création d'une session SSH avec un flux ShellStream

Ce flux permet d'envoyer les commandes au switch à la place de passer par la cmdlet « Invoke-SSHCommand », cependant il a un gros défaut. En effet certaines commandes envoyées au switch nécessitent plus ou moins de temps pour être réalisées en fonction des entrées à créer dans le fichier de configuration. Cela a pour conséquence qu'une commande envoyée au switch avant la fin de l'intervalle de temps requis pour réaliser la commande précédente sera tout bonnement ignorée. Une temporisation a donc été nécessaire.

5.2.2 Le mode « Create »

Pour créer une cmdlet utilisable par YASC il faut obligatoirement implémenter le mode « Create » et le mode « Audit ». Pour rappel le mode « Create » applique une configuration lue dans le fichier XML de déploiement et le mode « Audit » vérifie si la configuration dans ce même fichier a bien été appliquée ou est toujours bien d'actualité.

Le mode « Create » des cmdlets de configuration du switch, mis à part quelque exception, est relativement simple. Il consiste à utiliser la méthode citée précédemment pour envoyer des commandes propres au « CLI ³ » du switch HP. Cependant, le mode « Create » ne consiste pas qu'en un envoi de commandes brut vers le switch. On peut séparer le mode « Create » en quatre étapes majeures.

³ Une « Interface en ligne de commande » est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte (...) (Interface en ligne de commande - Wikipedia, 2016)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

La première étape consiste à lancer une première fois le mode « Audit », dont on parlera plus tard, et de vérifier, avant d'appliquer une nouvelle configuration, si le switch n'est pas déjà dans l'état voulu. Il suffit donc d'appeler la cmdlet une seconde fois, en lui passant exactement les mêmes paramètres, mais en mode audit :

```
if($Mode -eq "Audit"){...}
elseif($Mode -eq 'Create')
{
    $Status = Set-LDPyascHPSwitchVlanName -SwitchIPAddr $switchIpaddr -VlanNumber $VlanNumber -VlanName $VlanName
    -PathToKeePassProgram $PathToKeePassProgram -KeePassDataBase $KeePassDataBase -KeePassDatabasePwd $KeePassDatabasePwd
    -Username $Username -Mode Audit -Logger $Logger
    if($Status.RetVal -ne $true)
    {
        try
        {
            #region Create Mode...
        }
        catch
        {
            $Status.Error += $Logger | Write-LDPLogger -Severity Error -Message $_ -Invocation $Me -Verbose
            $Status.retVal = $false
            return $status
        }
    }
}
```

Exemple 26 Première étape : appeler le mode "Audit" de la cmdlet pour vérifier si la configuration n'est pas déjà appliquée.

La seconde étape consiste à récupérer dans le KeePass, les informations nécessaires à l'authentification sur le switch. Comme on peut le constater, chaque cmdlet prend plusieurs paramètres nécessaires à l'accès au KeePass :

- PathToKeePassProgram : qui est tout simplement le chemin d'accès vers le programme KeePass.
- KeePassDataBase : qui est le chemin d'accès vers le fichier KeePass contenant tous nos mots de passe.
- KeePassDataBasePwd : qui est le mot de passe d'accès au fichier KeePass.
- Username : c'est le nom de l'entrée dans le KeePass permettant de récupérer le mot de passe. Typiquement, c'est le nom de l'utilisateur avec lequel se connecter sur le switch. Par exemple « manager », « operator », ou d'autres utilisateurs créés à la main :



Figure 22 Entrée pour un utilisateur "manager" dans le KeePass.

Le code pour accéder au KeePass est relativement simple et se place un peu avant la construction de la commande SSH (troisième étape).

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

En troisième lieu vient se placer la construction de la commande et son envoi comme expliqué précédemment. Pour terminer, la quatrième étape consiste à appeler une deuxième fois le mode audit de la cmdlet pour vérifier si la configuration a effectivement bien été appliquée :

```
elseif($Mode -eq 'Create')
{
    #region premier audit...
    if($status.RetVal -ne $true)
    {
        try
        {
            #Initialisation du KeePass et récupération du mot de passe.
            $domain = Initialize-LDPyascHPswitchVlanName -PathToKeePassProgram $PathToKeePassProgram -ErrorAction SilentlyContinue
            $keyPwd = Get-LDPyascHPswitchVlanName -KeepassDatabase $keepassDatabase -KeepassDatabasePwd $keepassDataBasePwd -UserNameToFind $username
            $secPasswd = ConvertTo-SecureString $keyPwd -AsPlainText -Force
            $mycreds = New-Object System.Management.Automation.PSCredential ($username, $secPasswd)

            #region build the command .....
            #Second audit ...
            $status = Set-LDPyascHPswitchVlanName -SwitchIPAddr $switchIPaddr -VlanNumber $VlanNumber -VlanName $VlanName
            -PathToKeePassProgram $PathToKeePassProgram -KeepassDataBase $keepassDatabase -KeepassDatabasePwd $keepassDataBasePwd -username $username -Mode Audit -Logger $logger
            if($status.RetVal)
            {
                $status.Info += $logger | Write-LDPLLogger -Severity Info
                -Message "Name for vlan $VlanNumber has been successfully configured : $VlanName" -Invocation SME -verbose
            }
            return $status
        }
        catch
        {
            $status.Error += $logger | Write-LDPLLogger -Severity Error -Message $_ -Invocation SME -Verbose
            $status.retval = $false
            return $status
        }
    }
}
```

Exemple 27 Mode "Create" de la cmdlet permettant de créer un vlan et de configurer son nom.

5.2.3 Le mode « Audit »

Le mode « Audit » est un peu plus subtil que le mode « Create ». En effet, il faut trouver le moyen de récupérer la configuration actuelle du switch pour pouvoir vérifier si des modifications sont à apporter. Pour réaliser cela, le moyen le plus simple est de télécharger en SFTP le fichier de configuration et de l'analyser ligne par ligne par PowerShell.

Dans l'exécution d'une cmdlet en mode « Audit » trois étapes sont nécessaires. La première étape consiste à appeler la cmdlet récupérant le fichier de configuration du switch. Cette commande offre la possibilité de récupérer soit le fichier « running-config » soit le fichier « startup-config » depuis le switch via SFTP.

La seconde étape consiste à charger ce fichier en mémoire et le parcourir pour en retirer les informations nécessaires. Pour ce faire on utilise la cmdlet « Initialize-LDPyascHPSwitchCfg ». Cette cmdlet prend comme paramètre le chemin vers le fichier de configuration à charger ainsi que le type d'information qui est demandé en retour. Elle va alors s'occuper de découper tout le fichier pour retrouver, si elles existent, les informations propres à la cmdlet.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Par exemple, dans le cadre de la configuration du nom d'un vlan, cette cmdlet permet de récupérer le nom de chaque vlan dans une Hashmap. Il suffit alors de vérifier si cette Hashmap possède une entrée pour le vlan numéro « X » et, dans ce cas, vérifier si le nom correspond bien à celui qui est attendu :

```
if($Mode -eq "Audit"){
    $SwitchInfo = get-LDPyascHPSwitchConfig -switchHost $SwitchIPAddr -PathToKeePassProgram $PathToKeePassProgram
    -KeePassDataBase $KeePassDataBase -KeePassDatabasePwd $KeePassDatabasePwd -username $username -StartupConfig
    if(-not $SwitchInfo.Retval)
    {
        #region exit ...
    }
    $FilePath = $SwitchInfo.path + "\startup-config"
    $Status.Info += $Logger | Write-LDPLLogger -Severity Info -Message "Parsing $FilePath" -Invocation SME -Verbose
    #Découpe le fichier de configuration et retourne une Hashmap contenant tous les vlans et leurs noms.
    $Arr = Initialize-LDPyascHPSwitchCfg -FilePath $filePath -type NAME
    #region Vérifie si la configuration est correcte...
    #delete the configuration temp file
    Remove-Item -Path $SwitchInfo.path -Recurse
    return $Status
}
```

Exemple 28 Mode "Audit" de la cmdlet permettant de créer un vlan et de configurer son nom.

La troisième et dernière étape consiste donc à vérifier si les informations passées en paramètre de la cmdlet correspondent bien à ceux retrouver sur le switch et que la fonction « Initialize-LDPyascHPSwitchCfg » renvoie sous la forme d'une Hashmap :

```
if($Arr)
{
    $Vlan = "vlan " + $VlanNumber
    $Cmp= $Arr.$Vlan
    if($Cmp -eq $VlanName){
        #region ok...
    }
    else
    {
        #region not ok...
    }
}
else{
    $Status.Retval =$false
    $Status.Warn += $Logger | Write-LDPLLogger -Severity Warn -Message "No vlan found on $SwitchIPAddr" -Invocation SME -Verbose
    return $Status
}
```

Exemple 29 Vérifier si le nom du vlan "X" a bien été configuré avec le nom souhaité.

Evidemment une fois que les comparaisons ont été effectuées la copie du fichier de configuration est supprimée, pour des raisons de sécurité évidentes.

Globalement ce schéma de programmation de cmdlet pour YASC a été respecté pour la majorité des cmdlets programmés dans l'optique de configurer le switch HP ProCurve. Cependant, certaines fonctionnalités étant bien spécifiques, il a parfois été nécessaire de procéder autrement. Nous pouvons citer l'exemple de la configuration 802.1X qui a nécessité de travailler avec plusieurs cmdlets différentes. Il est donc intéressant de se pencher sur plusieurs cas bien spécifiques de cmdlets qui s'éloignent des chantiers battus.

5.3 Les cmdlets de configuration du switch HP ProCurve

Ce chapitre aborde différentes cmdlets créées sur YASC ayant pour objectif de configurer une fonctionnalité d'un switch HP ProCurve. Dans le but de limiter les répétitions, seules les cmdlets ayant des spécificités par rapport aux autres ou ayant été programmés différemment seront abordées. Les portions de codes présentées sont réduites aux commandes envoyées au switch dans le mode « Create » des cmdlets, cela dans le but de réduire le nombre d'informations superflues.

5.3.1 La configuration de 802.1X

802.1X est un standard lié à la sécurité des réseaux. Il permet de contrôler l'accès au réseau par le biais d'équipement comme des switches, des points d'accès, etc.

802.1X se base sur le protocole EAP pour transporter les informations d'identification vers un serveur d'authentification de type RADIUS, TACACS+, CAS ou autre. Ces protocoles sont des protocoles AAA, c'est-à-dire qu'ils réalisent trois fonctions : l'authentification, l'autorisation et la traçabilité. Le schéma classique d'une authentification par EAP comporte trois intervenants :

- Le « Peer » est le client demandant l'accès au média par l'intermédiaire de l'authenticator.
- « L'authenticator » est le matériel réseau fournissant l'accès au réseau, par exemple, un switch, un hub ou un point d'accès.
- « L'authentication server » et le serveur qui a pour objectif d'authentifier le peer et donc de lui accorder ou lui refuser l'accès au média.

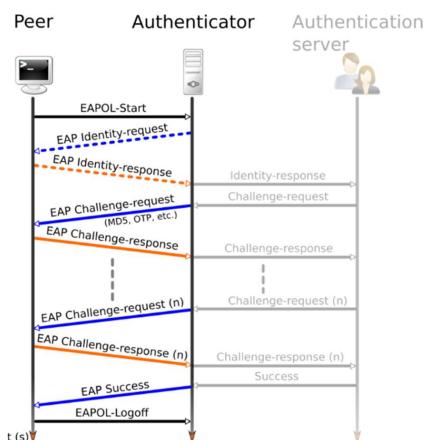


Figure 23 Schéma d'authentification EAP sur LAN (EAP - Wikipedia, 2017)

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Les switch HP ProCurve, tout comme les switch Cisco, sont compatibles avec la norme 802.1X. Cela veut dire qu'il est possible de contrôler l'accès à leurs ports. Ce chapitre n'abordera pas la partie de configuration propre à « l'authentication server ».

La configuration de l'authentification 802.1X sur un switch HP ProCurve se réalise en trois étapes (outre la création des différents vlans nécessaires). Ces étapes étant chacune très conséquentes. C'est pour cela que trois cmdlets ont été conçues dans YASC.

5.3.1.1 Configurer le serveur RADIUS sur un switch HP ProCurve

La première étape consiste à configurer les informations relatives au serveur RADIUS. La cmdlet « Set-LDPyascHPSwitchRadius » s'occupe de cette fonctionnalité. Cette cmdlet prend en paramètre :

- L'adresse IP ou le nom d'hôte du serveur RADIUS,
- Le « timeout », qui est l'intervalle temps avant que la connexion se coupe dans le cas où le serveur de répond plus,
- Le nombre de paquet retransmis en cas de perte,
- Les informations du KeePass nécessaire pour aller chercher le secret partagé entre l'authenticator (le switch) et le serveur RADIUS

```
New-SSHSession -ComputerName $SwitchIPAddr -Credential $mycreds -AcceptKey -Force
$session = Get-SSHSession -SessionId 0
$stream = $session.Session.CreateShellStream("xterm", 1000, 1000, 1000, 1000, 1000)
Start-Sleep -Milliseconds 250
$stream.WriteLine("")
Start-Sleep -Milliseconds 250
$stream.WriteLine("conf t")
Start-Sleep -Milliseconds 250
$stream.WriteLine("radius-server host $RadiusServerHost key $SharedSecret")
if($SharedSecret){$stream.WriteLine("radius-server host $RadiusServerHost key $SharedSecret`n")}
else{$stream.WriteLine("radius-server host $RadiusServerHost`n")}
Sleep 1
$stream.WriteLine("radius-server timeout $timeout`n")
Sleep 1
$stream.WriteLine("radius-server retransmit $retransmit`n")
Sleep 1
$stream.WriteLine("wr mem`n")
Sleep 1
Remove-SSHSession -SessionId 0
```

Exemple 30 Configurer les informations sur le serveur RADIUS

5.3.1.2 Configurer les informations AAA sur un switch HP ProCurve

La seconde étape consiste à configurer les informations relatives au modèle AAA d'authentification. La cmdlet « Set-LDPyascHPSwitchAAA » s'occupe de cette fonctionnalité. Notamment :

- Le type d'accès sécurisé par RADIUS : SSH, Telnet, Web interface ou console,
- Mode de privilège à protéger : login, enable, ou les deux,
- Un mode d'authentification secondaire dans le cas où le serveur RADIUS serait tombé : local ou rien.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
New-SSHSession -ComputerName $SwitchIPAddr -Credential $mycreds -AcceptKey -Force
$session = Get-SSHSession -SessionId 0
$stream = $session.Session.CreateShellStream("xterm", 1000, 1000, 1000, 1000, 1000)
Start-Sleep -Milliseconds 250
$stream.WriteLine("")
Start-Sleep -Milliseconds 250
$stream.WriteLine("conf t `n")
Start-Sleep -Milliseconds 250
$stream.WriteLine("aaa authentication login privilege-mode`n")
Start-Sleep -Milliseconds 500
$PrAuth = $RuntimeParameterDictionary.PrimaryAuth.value
$stream.WriteLine("aaa authentication $AccessMethods $privileges $PrAuth $SecondaryAuth`n")
Sleep 1
$stream.WriteLine("wr mem`n")
Sleep 1
Remove-SSHSession -SessionId 0
```

Exemple 31 Configurer les informations d'authentification AAA.

À noter que cette cmdlet amène un concept intéressant qui n'est pas utilisé dans les autres : les paramètres dynamiques. Pour plus d'informations voir en annexe page [9493](#).

5.3.1.3 Configurer les ports d'accès 802.1X et les vlans

La troisième et dernière étape consiste à configurer les ports voulus comme étant des « authenticator ports ». C'est-à-dire qu'ils ne donnent l'accès qu'au clients authentifiés. Cette fonctionnalité est gérée par la cmdlet « Set-LDPyasnPSwitch802Dot1X ». Elle permet notamment de :

- Configurer une liste de ports comme étant des « authenticator ports »,
- Configurer le vlan des clients non autorisés et le vlan des clients autorisés,
- Configurer le type d'authentification 802.1X : eap-radius,chap-radius,local,etc.

```
New-SSHSession -ComputerName $SwitchIPAddr -Credential $mycreds -AcceptKey -Force
$session = Get-SSHSession -SessionId 0
$stream = $session.Session.CreateShellStream("xterm", 1000, 1000, 1000, 1000, 1000)
Start-Sleep -Milliseconds 250
$stream.WriteLine("")
Start-Sleep -Milliseconds 250
$stream.WriteLine("conf t `n")
Start-Sleep -Milliseconds 250
$stream.WriteLine("aaa authentication port-access $AccessMethod`n")
Sleep 1
$stream.WriteLine("aaa port-access authenticator active`n")
Sleep 1
$stream.WriteLine("aaa port-access authenticator $AccessPorts`n")
Sleep 1
$stream.WriteLine("aaa port-access authenticator $AccessPorts auth-vid $AuthVlan`n")
Sleep 1
$stream.WriteLine("aaa port-access authenticator $AccessPorts unauth-vid $UnAuthVlan`n")
Sleep 1
$stream.WriteLine("wr mem`n")
Sleep 1
Remove-SSHSession -SessionId 0
```

Exemple 32 Configuration des "authenticator ports" et des vlans pour les clients autorisés et non autorisés.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

5.3.1.4 Configuration de 802.1X sur un switch CISCO

Sous IOS, le système d'exploitation que font tourner les switches et les routeurs CISCO, la configuration de 802.1X est similaire à celle sous un HP ProCurve.

Elle peut elle aussi être divisée en plusieurs étapes dont la première est la configuration d'un modèle AAA :

```
Switch# configure terminal
Switch(config)# aaa new-model
Switch(config)# aaa authentication dot1x default group radius
Switch(config)# dot1x system-auth-control
Switch(config)# interface gigabitethernet2/0/1
Switch(config)# switchport mode access
Switch(config-if)# dot1x port-control auto
Switch(config-if)# end
```

Figure 24 Configuration d'un modèle AAA. (Configuring 802.1X Port-Based Authentication, s.d.)

Les commandes « aaa authentication dot1x default group radius » et « dot1x system-auth-control » correspondent à l'étape réalisée par la cmdlet « Set-LDPyascHPSwitchAAA » sur le switch HP, c'est-à-dire activation de 802.1X pour un serveur RADIUS.

La commande « dot1x port-control auto » permet d'activer, pour un port, l'authentification AAA. Il est nécessaire qu'un port soit placé en « access port » sur un switch Cisco pour que 802.1X soit correctement configuré.

Comme pour le switch HP, il faut configurer les informations identifiantes le serveur RADIUS :

```
Switch(config)# radius-server host 172.120.39.46 auth-port 1612 key rad123
```

Figure 25 Configurer les informations du serveur RADIUS sur un switch Cisco. (Configuring 802.1X Port-Based Authentication, s.d.)

On constate donc que la configuration du standard 802.1X sur un switch HP ProCurve se réalise de façon similaire sur un switch Cisco. La syntaxe est évidemment différente cependant la logique reste la même.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

5.3.2 La configuration d'un VLAN

5.3.2.1 Configurer un vlan sur un switch HP ProCurve

Avant de parler des cmdlets réalisant la configuration complète d'un vlan, c'est-à-dire la création d'un ou de plusieurs trunks et la mise en place d'un ou plusieurs ports dans ce VLAN, il est opportun d'expliquer comment, sous un HP ProCurve, se déroule la gestion des différents VLANS, des trunks et des ports tagués dans ces VLANS. Attention cependant, la notion de trunk pour un switch HP n'équivaut pas à la notion de trunk pour un switch CISCO. Un trunk sur un switch HP est une agrégation de port équivalent à un port-channel sous CISCO IOS.

Pour créer un vlan sur un switch HP ProCurve il faut tout d'abord passer en mode de configuration globale et taper la commande « `vlan x` » (« `x` » représentant le numéro de VLAN). Cela permet de créer un VLAN et de passer l'utilisateur dans le mode de configuration de ce VLAN.

C'est dans ce mode de configuration que l'on va placer les ports dans le VLAN et qu'on créera les trunks. Pour ce faire on utilise respectivement la commande « `untagged` » et la commande « `tagged` ».

La commande « `untagged` » permet de placer un port dans un VLAN, c'est-à-dire que si une trame se présente sur le port et qu'elle n'est pas tagué elle sera placée automatiquement dans le VLAN. Par exemple, plaçons les ports 12 et 13 dans le VLAN 10 :

```
« ProCurve(config)#vlan  
ProCurve(vlan 10)#untagged 12-13 »
```

10

Mis en forme : Anglais (Royaume-Uni)

La commande « `tagged` » permet de créer des trunks. Ainsi si l'on souhaite que le port 14 soit un port trunk permettant de transporter des trames taguées dans le VLAN 10 et 15 on procède comme ceci :

```
"Procurve(config)# vlan 10  
ProCurve(vlan10)# tagged 14  
ProCurve(config)# vlan 15  
ProCurve(config)# tagged 14"
```

En réalité ces deux commandes peuvent être interprétées comme ceci : la commande « `untagged` » dans le mode de configuration d'un VLAN « `X` » signifie que si le port rencontre un paquet non tagué il le considérera comme étant dans le VLAN « `X` ». La commande « `tagged` » dans le mode de configuration du VLAN « `X` » signifie que les ports spécifiés après la commande « `tagged` » peuvent transporter ce VLAN « `X` » et agiront donc comme un trunk CISCO.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Au niveau de la cmdlet permettant de réaliser ce travail il a fallu gérer le paramètre concernant les ports qui peut prendre différents formats : « 5,7-9 », « 5 », « 5-9 », etc. Sinon sa structure est semblable à celle des autres cmdlets. Cette fonctionnalité est gérée par la cmdlet « Set-LDPyascHPSwitchTags » :

```
New-SSHSession -ComputerName $switchIPAddr -Credential $mycreds -AcceptKey -Force  
$session = Get-SSHSession -SessionId 0  
$stream = $session.Session.CreateShellStream("xterm", 1000, 1000, 1000, 1000, 1000)  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("")  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("conf t`n")  
Start-Sleep -Milliseconds 250  
$stream.WriteLine("vlan $VlanNumber`n")  
Sleep 3  
$stream.WriteLine("$taggingMode $startport-$endport`n")  
Sleep 2  
$stream.WriteLine("wr mem `n")  
Sleep 1  
Remove-SSHSession -SessionId 0
```

Exemple 33 Commande permettant de configurer un port dans un mode quelconque pour un vlan donné.

5.3.2.2 Configurer un vlan sur un switch CISCO

Configurer un vlan sur un switch CISCO se fait d'une manière bien différente. Sous CISCO IOS cette configuration se réalise sur l'interface elle-même. L'interface peut être dans deux modes bien distincts. Le premier étant le mode « access » et le second le mode « trunk ».

Le mode « access » signifie que le port est placé dans un VLAN spécifique et que tout paquet provenant de cette interface étant non tagué sera tagué dans ce VLAN. Cependant si un paquet tagué dans un autre VLAN se présente sur le port il sera détruit.

Le mode « trunk » signifie que le port peut transporter plusieurs VLAN. Il sert généralement de lien entre deux switches qui mettent en place des VLANS.

Les commandes CISCO pour configurer une interface dans un VLAN :

```
« Cisco(config)#interface fastethernet0/1  
Cisco(config-if)#switchport mode access  
Cisco(config-if)#switchport access vlan 10 »
```

Les commandes CISCO pour configurer une interface comme un trunk :

```
« Cisco(config)#interface fastethernet0/1  
Cisco(config-if)#switchport trunk encapsulation dot1q  
Cisco(config-if)#switchport mode trunk  
Cisco(config-if)#switchport trunk allowed vlan add 10-15 »
```

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

5.4 En résumé

Pour terminer, nous pouvons dire que la création d'une cmdlet pour le switch HP a été ~~sûrement~~sûrement la tâche la plus compliquée. Le système ne réagissait pas positivement aux commandes SSH envoyées par le module PoshSSH. Cependant la tâche a été menée à bien. Certains points restent à ajouter mais globalement la tâche a été remplie avec succès. Passons donc à la configuration d'un firewall Fortigate qui a été une tâche bien moins complexe mais bien plus longue ~~a à réaliser~~à réaliser.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6 Développement de modules de gestion d'un FortiGate.

6.1 Firewall Fortigate de chez Fortinet

Fortinet est une multinationale américaine. Elle conçoit et commercialise des logiciels, des équipements et des services de cyber-sécurité comme les pare-feu Fortigate.

Les firewalls Fortigate sont le deuxième objectif du stage. Le but étant, comme pour les switch HP, d'automatiser leurs configurations et leurs déploiements.

Ce chapitre détaillera comment les cmdlets dédiées à YASC ont été imaginées et programmées pour configurer un firewall Fortigate. Nous verrons que certains points de programmation ressemblent aux cmdlets utilisées pour le switch HP et, inversement, que tout n'a pas été réalisé totalement de la même manière.

Les techniques d'accès au firewall étant similaires à celles du switch HP, nous ne nous attarderons pas longtemps dessus pour nous concentrer sur les cmdlets. En effet, celles-ci ont été programmées différemment des autres dû à l'architecture de YASC et du firewall. La configuration du firewall Fortigate introduit ainsi le système de collection de données présenté au chapitre 4.3.2 « [Déclarer une cmdlet dans le fichier YascEngine.xml](#) ». Ce qui donne un ensemble de cmdlet relativement différentes les unes par rapport aux autres.

6.2 L'accès par SSH

Nous l'avons vu au chapitre précédent, pour le switch HP il a fallu télécharger le fichier de configuration pour pouvoir récupérer les informations de configuration du switch. Pour le Fortigate ce n'est pas la peine. En effet, le CLI du firewall se prête parfaitement à l'exercice. Prenons pour illustrer ces propos un exemple simple : la cmdlet de configuration d'une adresse. Une adresse sur le firewall représente l'adresse d'un hôte ou un réseau.

Commençons par le mode « Audit ». Dans ce mode nous devons vérifier que l'adresse (dans cet exemple) n'est pas déjà créée. La mode « Audit » commence par une récupération des mots de passe dans le KeePass tout comme pour le switch HP. Ensuite on réalise une connexion SSH avec la cmdlet « New-SSHSession ». Jusqu'ici tout se passe comme sur le switch. Cependant, c'est maintenant que les choses divergent. En effet, pour le firewall nous n'allons pas télécharger un fichier de configuration.

Il faut savoir que le Fortigate fonctionne par mode de priviléges comme les switch CISCO et HP. Ainsi, si l'on souhaite accéder à toutes les adresses du firewall il suffit de taper : « config firewall address ». Pour éditer une adresse bien particulière, il faut alors taper « edit MyAddress ». Pour chaque mode de configuration, le firewall permet de taper la commande

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

« show full-configuration ». Ainsi on peut récupérer, soit toutes les informations concernant toutes les adresses, soit toutes les informations d'une seule adresse.

En somme, la CLI du firewall nous permet de réaliser un filtrage naturel sans avoir à le faire par nous-mêmes. De plus, contrairement au switch HP, la cmdlet « Invoke-SSHCommand » fonctionne sur le Fortigate.

```
#Create a ssh session
$Id = New-SSHSession -ComputerName $Firewall -Credential $mycreds -AcceptKey -Force
$SessionID = $Id.SessionId

#Invoke a command and catch the output
$RetCmd = Invoke-SSHCommand -SessionID $SessionID -Command "conf firewall address
edit $Name
show full-configuration"

$Ret=Remove-SSHSession -SessionId $SessionID

$Conf = $RetCmd.Output
$Conf = $Conf -replace "--More--", ""
$Conf = $Conf.Trim()
```

Exemple 34 Crée une session SSH vers le Fortigate et récupérer la configuration pour une adresse.

Ainsi, après avoir créé la connexion SSH, on récupère les informations pour une adresse donnée via la commande ci-dessous passée à la cmdlet « Invoke-SSHCommand » :

```
"conf firewall address
    edit $Name
        show full-configuration"
```

On obtient alors en sortie un objet qui possède une variable membre « Output ». Elle contient le retour de la commande envoyée au firewall. Il suffit alors d'utiliser les outils que PowerShell nous met à disposition pour récupérer les informations concernant l'adresse qui nous intéresse et de les comparer avec les valeurs attendues.

En mode « Create » c'est encore plus simple. La cmdlet « Invoke-SSHCommand » fonctionnant sur le Fortigate, il suffit d'envoyer les commandes sans se tracasser de devoir gérer une quelconque temporisation.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
#Create a ssh session
$id = New-SSHSes$ion -ComputerName $Firewall -Credential $mycreds -AcceptKey -Force
$SessionID = $id.SessionId

$Cmd = "conf firewall address
edit $Name
set subnet $NetLan $NetLanMask
set color $GUIColor
next
end"

#region Autosave
$RetCmd = Invoke-SSHCommand -SessionID $SessionID -Command $Cmd
$Ret=Remove-SSHSes$ion -SessionId $SessionID
```

Exemple 35 Configurer une adresse sur le Fortigate via la cmdlet "Invoke-SSHCommand".

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.3 Les cmdlets de configuration du firewall Fortigate

6.3.1 Le mode de sauvegarde de la configuration.

Sur un firewall Fortigate plusieurs possibilités existent pour sauvegarder une configuration. Il embarque en réalité trois modes de sauvegardes :

- Le mode « Automatique » : ce mode permet de confirmer le changement de la configuration de manière automatique et de la sauvegarder.
- Le mode « manuel » : nécessite que l'utilisateur confirme son besoin de sauvegarder la configuration actuelle.
- Le mode « revert » : permet de sauvegarder la configuration et réalise un retour à l'état précédent dans le cas d'un timeout. En somme ce mode est très utile si l'utilisateur réalise un malencontreux changement qui lui fait perdre l'accès distant à son firewall.

Il est évident que dans le cas où le mode de sauvegarde du firewall n'est pas configuré sur « Automatique » il est obligatoire d'ajouter la commande de sauvegarde manuelle à toute commande SSH qui voudra modifier la configuration du firewall. C'est pourquoi, il existe une cmdlet appelée « Test-LDPyascFortiGAutoSave ». Cette cmdlet se contente de vérifier si le mode de sauvegarde automatique est activé ou non :

```
#Create a ssh session
$Id = New-SSHSession -ComputerName $Firewall -Credential $mycreds -AcceptKey -Force
$SessionID = $Id.SessionId
#Invoke a command and catch the output
$retCmd = Invoke-SSHCommand -SessionID $SessionID -Command 'conf system global
show full-configuration'
$conf = $retCmd.Output
$conf = $conf -replace "--More--"," "
$conf = $conf.Trim()
#Verify if the configuration auto saving is enabled
if($conf.Contains('set cfg-save automatic'))
{
    $status.RetVal = $true
    $status.Info += $logger | Write-LDPLogger -Severity Info -Message "Configuration a
}
else
{
    $status.Info += $logger | Write-LDPLogger -Severity Info -Message "Configuration au
}
$ret=Remove-SSHSession -SessionId $SessionID
if(-not $ret)
{
    $status.Error += $logger | Write-LDPLogger -Severity Error -Message "Failed to remo
}
return $status
```

Exemple 36 Vérifie le mode de sauvegarde d'un firewall.

Ce code doit obligatoirement être appelé dans chaque cmdlet de YASC. Sinon l'utilisateur s'expose à la possibilité que toute sa configuration soit totalement ignorée par le firewall.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.3.2 La configuration d'une interface

Un firewall, comme un switch, possède des interfaces qu'il est possible de gérer. L'intérêt principale est de pouvoir configurer son adresse IP, son alias, les méthodes d'accès, etc. La cmdlet utilisée sous YASC pour configurer une interface de firewall est la cmdlet « Set-LDPyascFortiGInterface ». Il est intéressant d'explorer le fonctionnement de cette cmdlet en premier lieu car elle est assez simple et permet d'expliquer comment utiliser une collection de données avec YASC. Or, contrairement à ce que l'on a pu voir pour le switch, énormément de cmdlet de configuration du Fortigate utilisent des collections de données.

Les paramètres de la cmdlet sont au nombre de sept (sans compter les paramètres pour l'accès au KeePass et l'adresse IP du firewall). Ils sont tous obligatoires et permettent de configurer de la façon la plus complète l'interface :

- « IntName » : spécifie le nom de l'interface,
- « IPAddr » & « Mask » : l'adresse IP et le masque de sous-réseau,
- « Coll_Allowaccess » : la collection de données permettant de spécifier les accès autorisés à cette interface, par exemple, ssh, telnet, http, etc,
- « Alias » : ce paramètre spécifie un alias pour l'interface,
- « Identification » : permet l'identification du firewall,
- « SnmpIndex » : configure un index pour le protocole SNMP.

D'un point de vue programmation, deux points d'intérêt sont à relever dans cette cmdlet : la collection de données et la cmdlet de vérification des adresses IP pour éviter les chevauchements.

6.3.2.1 La collection de données

La première concerne la collection de données « Coll_Allowaccess ». Dans le cas de la cmdlet de configuration de l'interface, cette collection de données doit contenir tous les moyens de connexion possible à cette interface et spécifier s'ils sont autorisés ou non. Par défaut nous considérons que tout ce qui se trouve dans la collection est autorisé. En effet, la commande du firewall pour configurer un moyen d'accès sur une interface écrase toute valeur déjà préexistante pour cette interface. En somme, si l'interface numéro quatre du firewall est configurée pour accepter les connexions telnet ou ssh via la commande « set allowaccess ssh telnet », alors la commande « set allowaccess https http » remplace les méthodes d'accès ssh et telnet par http et https.

Une collection de données dans YASC prend toujours la forme d'une Hashmap. Dans le mode « Audit » de la cmdlet on réalise une double vérification. On parcourt une première fois la Hashmap pour vérifier si tous les moyens d'accès demandés sont bien configurés sur le firewall. Ensuite, on parcourt les valeurs récupérées depuis le firewall et on vérifie si aucune

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

valeur n'est en trop par rapport à l'HashMap. Cela voudrait dire que quelqu'un a ajouté une méthode d'accès à l'interface non autorisée par l'utilisateur.

```
$AllowedMethods = $Conf | select-string -Pattern "set allowaccess\s+"
if($AllowedMethods)
{
    $AllowedMethods= $AllowedMethods -replace 'set', ''
    $AllowedMethods = $AllowedMethods -replace 'allowaccess', ''
    $AllowedMethods = $AllowedMethods.Trim()
    $Methods = $AllowedMethods -split " "
    #Verify if the allowaccess command miss an access methods.
    foreach($M in $Coll_Allowaccess){
        $Member = $M.allowaccess
        if(-not ($Methods -contains $Member)){
            $Status.Warn += $Logger | Write-LDPLLogger -Severity Warn -Message "Access method {$Member} is a missing allowed access"
            $Status.Retval = $false
        }
    }
    #Verify if there is unallowed access methods in the allowaccess command.
    foreach($M in $Methods)
    {
        if(-not ($Coll_Allowaccess.Allowaccess -contains $M)){
            $Status.Warn += $Logger | Write-LDPLLogger -Severity Warn -Message "Access method {$M} is not in the allowed access met"
            $Status.Retval = $false
            break
        }
    }
}
```

Exemple 37 Vérifier les méthodes d'accès d'une interface.

Au niveau du mode « Create » on se contente de créer la commande CLI du firewall pour configurer l'autorisation de toutes les méthodes d'accès contenues dans la HashMap « Coll_Allowaccess ».

```
foreach($access in $Coll_Allowaccess.Allowaccess)
{
    $listOfAccess += "$access "
}
$Cmd+="set allowaccess '$listOfAccess'"
```

Exemple 38 Créer une commande CLI sur base d'une collection de données.

6.3.2.2 La déclaration dans les fichiers XML

Maintenant voyons comment déclarer cette collection dans les fichiers XML. Cet exemple est une application de la théorie vue au chapitre 4.3.2.

Tout d'abord il faut déclarer l'étape dans le fichier XML de l'engine : YascEngine.XML. Pour ce faire, on donne un nom à l'étape, par exemple, « Fortigate-Interface » :

```
<Fortigate-Interface>
    <CmdLet>Set-LDPyascFortiGInterface</CmdLet>
    <NeedsReboot>no</NeedsReboot>
</Fortigate-Interface>
```

Exemple 39 Déclaration de l'étape dans le fichier de l'engine.

À cette étape est associé la cmdlet « Set-LDPyascFortiGInterface » par le biais du tag « Cmdlet ». Il faut ensuite déclarer les paramètres grâce à des tags « Param ». Pour rappel, ce

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

tag est composé de deux sous-tags. Le sous-tag « Name » permettant de spécifier le nom du paramètre et le sous-tag « From » permettant de spécifier à YASC ou-ù trouver la valeur du paramètre. Pour simplifier l'exemple, déclarons seulement la collection de données « Coll_Access ». Nous pouvons déclarer sa valeur depuis le fichier des métadonnées ou depuis le fichier de déploiement. Dans notre cas, nous déclarons cette collection de données dans le fichier de déploiement (cependant un exemple avec le fichier des métadonnées sera abordé plus tard), pour ce faire :

```
<Fortigate-Interface>
  <CmdLet>Set-LDPyascFortiGInterface</CmdLet>
  <Param>
    <Name>Coll_Access</Name>
    <From>Step</From>
  </Param>
  <NeedsReboot>no</NeedsReboot>
</Fortigate-Interface>
```

Exemple 40 Ajouter à la déclaration un paramètre (dans ce cas-ci, une collection de données).

Ainsi, pour remplir cette collection de données depuis le fichier de déploiement on précède comme ceci :

```
<Step>
  <Name>Configure a fortigate interface</Name>
  <DetailedDescription>Configure a fortigate interface with ip address 192.168.2.5/24</DetailedDescription>
  <Type>Action</Type>
  <Yasc>Fortigate-Interface</Yasc>
  <Prereq>None</Prereq>
  <IsFatal>yes</IsFatal>
  <Param>
    <IntName>internal4</IntName>
    <IPaddr>192.168.2.5</IPaddr>
    <mask>255.255.255.0</mask>
    <alias>s.voice</alias>
    <Identification>enable</Identification>
    <SnmpIndex>5</SnmpIndex>
    <Coll_Access>
      <Allowaccess>ssh</Allowaccess>
    </Coll_Access>
    <Coll_Access>
      <Allowaccess>https</Allowaccess>
    </Coll_Access>
    <Coll_Access>
      <Allowaccess>http</Allowaccess>
    </Coll_Access>
  </Param>
</Step>
```

Exemple 41 Donner les valeurs attendues à la collection de données.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.3.2.3 Gérer les chevauchements d'adresses IP

Un point particulier propre à la cmdlet de configuration d'une interface est la gestion des adresses IP. En effet, lorsque l'on configure une adresse sur une interface dans un firewall, celle-ci ne doit pas être en conflit avec l'adresse d'une autre interface. Si l'on tente de configurer une adresse IP statique sur une interface donnée et qu'elle est déjà utilisé sur une autre le firewall renverra un code d'erreur. Cependant il est plus agréable d'obtenir un retour moins violent du firewall. C'est pour cela que la cmdlet « Set-LDPyascFortiGInterface » appelle « Test-LDPyascIPExists ». Cette cmdlet s'assure que les informations composées du paramètre « IPAddr » et « Mask » ne soient pas en conflit direct avec une adresse déjà configurée sur le firewall.

En réalité, cette cmdlet est appelée une seule fois lors du mode « Audit ». Pour rappel, le mode « Create » réalise deux « Audit ». Une fois au début du mode et une fois à la fin. Ainsi lors du premier « Audit » on réalise ces tâches :

- Appel de la cmdlet « Test-DPyascFortiGIPExists ». S'il y a chevauchement d'adresses IP alors on configure un booléen sur « vrai ». Ensuite, lors de la construction de la commande à envoyer au firewall dans le mode « Create », s'il y a chevauchement, on n'ajoute pas la commande de configuration de l'adresse. S'il n'y a pas de chevauchement alors on vérifie seulement si l'adresse sur l'interface est correctement configurée.
- On vérifie si les autres paramètres sont bien configurés sur l'interface (alias, SnmpIndex, etc.).

```
#Verify if the ip address is not already used by an other interface
$IPExists = Test-LDPyascFortiGIPExists -Firewall $Firewall -IntName $IntName -IPAddr $IPAddr -Mask $Mask -PathToKeePassProgram $PathToKeePassProgram
-KeppassDataBase $KeppassDataBase -KeppassDatabasePwd $KeppassDatabasePwd -Username $Username -Mode Audit -Logger $Logger
if(-not $IPExists){
    $IPCmd = $Conf | select-string -Pattern "set ip\s"
    if($IPCmd)
    {...}
    else{
        $Status.Warn += $Logger | Write-LDPLogger -Severity Warn -Message "Subnets overlapping : IP address {$IPAddr} and subnet mask {$Mask} already used"
        $NoOverlapping = $true
        $Status.Retval = $false
    }
}
```

Exemple 42 Vérifier un éventuel chevauchement d'adresses IP via la cmdlet "Test-LDPyascFortiGIPexists".

```
if(-not $NoOverlapping){$Cmd+="set ip $IPAddr $mask`n" }
```

Exemple 43 Dans le cas où un chevauchement a été détecter, on n'ajoute pas la commande de configuration de l'adresse IP à la commande globale.

Pour terminer le deuxième mode « Audit » repasse une fois sur la configuration dans le cas où elle n'aurait pas été appliquée et qu'un conflit d'adresses IP persiste toujours.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.4 La configuration d'une « Policy »

Un des intérêts principaux d'un firewall est de pouvoir filtrer le trafic entrant et sortant via ce qu'on appelle des « Policies ». Une policy, en français une politique, est un ensemble de règle permettant de définir le type de trafic accepté selon plusieurs critères. On peut citer notamment :

- L'interface entrante et sortante,
- Le port de destination et source,
- L'adresse IP de destination et source,
- Le type de protocole,
- L'heure et la date,
- Etc.

Il faut savoir que sur les firewalls de chez Fortinet, les Fortigates, certaines fonctionnalités peuvent être perçues comme des objets auxquels l'on peut faire référence. Ainsi une adresse possède un alias et peut être référencée au sein d'une policy⁴. Il en va de même pour beaucoup d'autres options du firewall comme les « scheduler », qui spécifient une heure et une date.

Ces objets sont beaucoup utilisés lors de la configuration d'une policy. En effet, comme abordé plus haut, la policy nécessite plus ou moins d'informations pour filtrer le trafic. Ainsi, avant de configurer une policy, il faut avoir configuré les autres objets (car la policy est aussi un objet).

La cmdlet dans YASC s'occupant de créer ou de vérifier les policies sur le Fortigate se nomme « Set-LDPyascFortiGPolicy ». Cette cmdlet est relativement différentes des autres.

Tout d'abord c'est la dernière cmdlet réalisée. En effet les autres cmdlets, configurant les différents aspects du firewall, sont en charges de créer les objets dont la policy aura besoin. Cela signifie que cette cmdlet est basées sur le résultat de toute les autres.

Cela a pour conséquence directe que cette cmdlet fait appel à des fonctions permettant de vérifier l'existence de chaque objet. Ainsi, la création de cette cmdlet à-a nécessiter-nécessité le développement de six autres.

⁴ Pour rappel une adresse sur le Fortigate est un objet représentant un réseau ou un hôte

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.4.1 Les objets

Comme dit précédemment la configuration d'une policy sur le firewall fortigate nécessite d'avoir préconfiguré des objets représentant des adresses IP, des schedulers, des services etc. Ainsi, il est extrêmement important que la cmdlet « Set-LDPyascFortiGPolicy » vérifie l'existence de ces objets.

Voici un tableau récapitulatif des paramètres, donc des objets à configurer pour une policy, et les cmdlets correspondantes utilisées pour vérifier leurs présences.

Objet(s)	Paramètres de la cmdlet	Cmdlet
Interface source	SrcInterface	Get-FortiGInterface
Interface destination	DestInterface	Get-FortiGInterface
Adresse source	SrcAddr	Get-FortiGAddress
Adresse destination	DestAddr	Get-FortiGAddress
Date et heure de l'activité de la policy	Schedule	Get-FortiGSchedule
Nom du service	Service	Get-FortiGService
Limitation de bande passante	TrafficShaper et TrafficShaperReverse	Get-FortiGShaper
Nom du groupe d'utilisateur autorisé à utiliser la policy	Groups	Get-FortiGGroups

Toutes ces cmdlets sont utilisées au sein du mode « Create » de la cmdlet « Set-LDPyascFortiGPolicy ». Le but étant d'ignorer la configuration d'une fonctionnalité si l'objet à utiliser sur le firewall n'existe pas. Le second « Audit » du mode « Create » se chargera alors de prévenir l'utilisateur que certains paramètres n'ont pas été configurés car l'objet est inexistant.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.5 La configuration d'un filtre web

Une des nombreuses fonctionnalités très intéressantes d'un firewall est la possibilité de créer des filtres de pages web. Ces filtres sont aussi représentés sur le firewall comme des objets et peuvent s'apparenter à des policies sauf qu'à la place de manipuler des services, les filtres web manipulent des pages web.

Configurer un filtre web du Fortigate par YASC nécessite d'utiliser la cmdlet « Set-LDPPyascFortiGWebFilter ». Cette cmdlet est très spéciale car elle est la première à utiliser une collection de données qui a été déclarée dans le fichier des métadonnées.

En effet, un filtre web permet de spécifier quel type de page doit être filtré. Or, sur le Fortigate à chaque type de page correspond un entier. Il n'est malheureusement pas possible d'entrer le nom du type, de la catégorie, de la page dans la commande car celle-ci n'accepte que le chiffre correspondant à la catégorie.

C'est pourquoi cette cmdlet utilise une collection de données déclarées dans le fichier des métadonnées. Cette collection est une Hashmap contenant de multiples clés. Chacune correspond à une catégorie et a pour valeur associer un entier. Ces valeurs sont évidemment celles utilisées sur le firewall. Cette collection de données a pour seul et unique but de permettre à la cmdlet de réaliser une corrélation un nom de catégorie de page web et son entier.

En réalité pour déterminer quelle page doit être filtrée ou pas, la cmdlet s'équipe d'une seconde collection de données, « Coll_Filters », qui sera déclarée au sein du fichier de déploiement de l'application. Cette collection contient aussi des clés correspondantes aux catégories web. Chaque clé à une valeur associée permettant de déterminer la politique d'autorisation d'une catégorie : « allow » ou « block ».

6.5.1 La déclaration de la collection

La collection « Coll_Categories » contient donc toutes les catégories existantes sur le firewall et leurs entiers associés. Un exemple de déclaration de cette collection se trouve en annexe page [9695](#).

Il a été décider-décidé de définir cette collection au sein du fichier des métadonnées car entre chaque version du firmware du firewall de nouvelles catégories peuvent être ajoutées et il est plus convenable d'éditer un fichier XML que de manipuler le fichier source des cmdlets. Une erreur est effectivement bien trop vite arrivée et peut faire perdre énormément de temps.

Cette collection de données est utilisée aussi bien dans le mode « Audit » que dans le mode « Create » de la cmdlet.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.5.2 Utilisation dans le mode « Audit »

Dans le mode « Audit », on procède en trois étapes pour vérifier les différentes catégories et leurs politiques d'accès.

La première étape consiste à créer une Hashmap après avoir récupérer la configuration du filtre web. Cette Hashmap va contenir toutes les catégories configurées sur le firewall ainsi que la politique d'accès pour chaque catégorie. Ce tableau est construit en découplant le fichier de configuration et en parcourant chacune de ses lignes.

```
#Create a tab with categories and policies for each category.
$FiltersMap = @{}

foreach($line in $Conf){
    $CtgLine = $line | Select-String -Pattern "set category\s"
    $PolicyLine = $line | Select-String -Pattern "set action\s"
    if($CtgLine)
    {
        $CmdTab = $CtgLine -split " "
        $Ctg = $CmdTab[2]
    }
    if($PolicyLine)
    {
        $CmdTab = $PolicyLine -split " "
        $Policy = $CmdTab[2]
    }
    if($Policy -and $Ctg){
        $FiltersMap.Add($Ctg,$Policy)
        $Ctg=''
        $Policy=''
    }
}
```

Exemple 44 Construction du tableau contenant les catégories et leur politique d'accès.

La seconde étape consiste à parcourir chaque élément de la Hashmap « Coll_Filters » contenant les catégories et leurs politiques à appliquer sur le firewall. On vérifie alors si chaque catégorie est présente dans la Hashmap créée à la première étape ainsi que la politique d'accès appliquée. Si la catégorie existe et que la politique d'accès est correcte alors la catégorie est configurée comme souhaitée. Cela permet d'éviter que quelqu'un supprime une des catégories configurées sans permissions.

```
#Verify if a category is missing for the web filter.
foreach($Map in $Coll_Filters)
{
    $Ctg = $Map.Keys
    $Policy = $Coll_Filters.$Ctg
    $CtgNumber = $Coll_Categories.$Ctg
    $ActualPolicy = $FiltersMap.$CtgNumber
    if(-not ($FiltersMap.keys -contains $CtgNumber))
    {
        #region Warning log...
        $MissingFilters+=$CtgNumber
    }
    elseif(-not($Policy -eq $ActualPolicy))
    {
        #region Warning log...
        $Status.RetVal = $False
    }
}
```

Exemple 45 Vérifier si une catégorie n'a pas été supprimée par rapport à la configuration souhaitée.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

La troisième étape réalise en fait le traitement inverse. Nous allons vérifier si dans la hashmap contenant toutes les catégories configurées pour le filtre web sur le firewall, il n'y a pas de catégories qui sont configurées alors qu'elles ne devraient pas. C'est-à-dire qu'il faut repérer toutes les catégories présentes dans la hashmap de configuration mais non présentes dans la hashmap « Coll_Filters ». Cela permet d'être prévenu dans le cas où quelqu'un aurait ajouter une catégorie non souhaitée.

```
#Verify if an unallowed category is set on the firewall.
foreach($ctg in $FiltersMap.Keys)
{
    #Get the name of the categories
    $ctgName = $Coll_Categories.Keys.Where({$Coll_Categories.$_ -eq $ctg; }, [System.Management.Automation.WhereOperatorSelectionMode]::First);
    if(-not ($Coll_Filters.Keys -contains $ctgName))
    {
        #region Warning log...
    }
}
```

Exemple 46 Vérifier si une catégorie n'a pas été ajouter par rapport à la configuration souhaitée.

6.5.3 Utilisation dans le mode « Create »

Dans le mode « Create », on se contente de parcourir la Hashmap « Coll_Filters » et d'ajouter à la commande les catégories à configurer sur le firewall ainsi que les actions à leurs appliquer.

```
$i=1
Foreach($filter in $Coll_Filters){
    $f = $filter.keys
    $fid = $Coll_Categories.$f
    $action = $Coll_Filters.$f
    $Cmd += "delete $i
edit $i
set category $fid
set action $action
next `n"
    $i++
}
```

Exemple 47 Ajouter à la commande console à envoyer au firewall, la configuration des catégories et leur politique d'accès.

6.6 La configuration d'IPSec

6.6.1 IPSec et IKE

« Internet protocol security », alias IPSec, est un ensemble de protocoles permettant le transport de données sécurisées sur un réseau IP. Contrairement aux anciens standards de sécurité, IPSec opère au niveau de la couche 3 du modèle OSI, la couche réseau, et permet l'utilisation de plusieurs algorithmes de hashage et de chiffrement. IPSec est un composant essentiel dans un VPN qui assure sa composante sécurité. Il n'est cependant pas le seul protocole de sécurité permettant ce genre d'utilisation. Nous pouvons par exemple citer SSL.

La suite de protocole IPSec fonctionne en deux phases pour établir une connexion chiffré et sécurisé. Ces deux phases sont appelées les phases IKE⁵:

- IKE phase I : la phase I de IKE consiste à établir une ISAKMP⁶ SA⁷. Pour cela la phase I de IKE va permettre aux deux terminaux de négocier les termes de la communication chiffrée, c'est-à-dire les algorithmes de chiffrement et d'authentification, les certificats, etc. ISAKMP est en fait un sous-protocole utilisé par IKE pour permettre d'obtenir toutes ces informations sous forme d'une association de sécurité : « SA ». Cette ISAKMP SA permet alors de créer un tunnel chiffré, généralement en UDP sur le port 500, permettant de négocier de façon plus sécurisée les informations nécessaires à l'établissement d'un tunnel chiffré entre les deux parties, mais cette fois pour la communication finale entre les deux « endpoints ».
- IKE phase II : cette phase permet de négocier le tunnel VPN final entre les deux « endpoints » via le tunnel créé en phase I. De cette négociation résulte de nouvelles associations de sécurité utilisées pour la communication VPN entre les deux parties. La communication peut être portée par deux protocoles différents : AH ou ESP. AH offre l'intégrité mais pas la sécurité de chiffrement alors que ESP offre les deux.

⁵ « Internet Key Exchange »

⁶ « Internet Security Association and Key Management Protocol »

⁷ « Security Association »

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.6.2 IPSec sur le firewall Fortigate

Pour configurer un VPN IPSec sur un firewall Fortigate il faut configurer les informations nécessaires aux deux phases de IKE. Cela se réalise sous deux menus différents : « config vpn ipsec phase1-interface » et « config vpn ipsec phase2-interface ».

Les deux phases sont configurées respectivement par les cmdlets « Set-LDPyascFortiGIPSecPhase1 » et « Set-LDPyascFortiGIPSecPhase2 ».

Voici, sous forme de tableau, la liste des paramètres utilisés dans la cmdlet « Set-LDPyascFortiGIPSecPhase1 » pour configurer la phase I de IKE sur le Fortigate :

Nom du paramètre	Description
Interface	Nom de l'interface sur laquelle configurer le VPN IPSec.
Localid	ID locale utilisé pour l'authentification.
Localid-type	Type du localid : adresse IP, fqdn, etc ...
Dhgrp	Définit le groupe Diffie-Hellman pour la phase I de IKE.
CryptoAlgo	Définit l'algorithme de chiffrement utilisé pour le tunnel d'échange de clés.
HashAlgo	Définit l'algorithme de hashage utilisé pour le tunnel d'échanges de clés.
KeePassSharedSecret	Clé utilisée dans le KeePass pour retrouver le mot de passe à configurer comme secret partagé pour IPSec.
RemoteGWAddress	Adresse de la passerelle par défaut.

D'un point de vue programmation, rien de spéciale. On peut cependant noter l'ajout du système de paramètre dynamique (annexe, page [9493](#)) utilisé pour obtenir une combinaison algorithmes de chiffrement/algorithme de hashage correcte et cohérente. En effet, le firewall n'accepte que certaines combinaisons, par exemple AES256 et MD5 ou SHA256. À noter que pour configurer l'algorithme de hashage et de chiffrement le firewall ne demande qu'une seule commande. Il a été décidé de diviser la commande en deux paramètres pour permettre de gérer plus facilement les valeurs.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Voici, sous forme de tableau, la liste des paramètres utilisés dans la cmdlet « Set-LDPyascFortiGIPSecPhase2 » pour configurer la phase II de IKE sur le Fortigate:

Nom du paramètre	Description
Phase1Name	Le nom donné à l'étape IPSec phase I dans la première cmdlet.
Dhgrp	Définit le groupe Diffie-Hellman pour la phase I de IKE.
CryptoAlgo	Définit l'algorithme de chiffrement utilisé pour le tunnel d'échange de clés.
HashAlgo	Définit l'algorithme de hashage utilisé pour le tunnel d'échanges de clés.
KeepAlive	Permet d'autoriser le firewall à négocier une nouvelle SA pour la phase II avant que la SA actuelle expire, permettant alors de garder le tunnel actif. Si ce paramètre est désactivé, une nouvelle SA est négocié seulement s'il y a du trafic dans le tunnel VPN.
AutoNegotiate	Permet d'autoriser la négociation de la SA de phase II même s'il n'y a pas de trafic. Cela se répète toute les cinq secondes jusqu'à la réussite de la négociation.
KeyLifeSeconds	Configure un nombre de secondes après lequel la clé de phase II expire.

Cette cmdlet implémente le même système de paramètre dynamique que la cmdlet « Set-LDPyascFortiGIPSecPhase1 » concernant les algorithmes de chiffrement et de hashage.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.7 Divers

Certaines des fonctionnalités du FortiGate ne sont pas abordées pendant la formation d'informaticien réseaux à la Haute Ecole de la Province de Liège. Ce chapitre a pour but d'expliquer les nouvelles notions vues lors de l'apprentissage de ces différentes fonctionnalités. Sauf exception, ce dernier chapitre se concentre surtout sur la partie théorique de ces fonctionnalités, donc nous n'aborderons pas, ou très peu, de cmdlets dans ce chapitre.

6.7.1 S'authentifier par LDAP

LDAP, « Lightweight Directory Access Protocol » est un protocole permettant d'interroger et de modifier un service d'annuaire. Il fonctionne sur base d'une structure arborescente dont chacun des nœuds est constitué d'attributs. Ce modèle est utilisé par les Active Directory Windows Server.

Globalement l'arborescence se construit de cette manière : on utilise le nommage DNS pour les éléments de base de l'annuaire (racine et premières branches) et on les représente par le « domain components » ou « dc ». Cela peut correspondre au nom d'un domaine. Les branches suivantes de l'arborescence sont des OU (« organizational unit ») ou des personnes/ordinateurs et sont représenté~~s~~ respectivement par le diminutif « OU » pour les « organizational unit » et « CN » (« common name ») pour les personnes/ordinateurs.

L'assemblage de tous les composants de l'arbre donne à un utilisateur de l'annuaire son « distinguished name ». Sous cet exemple, le « distinguished name » du portable 001 s'exprime comme ceci : « *cn=lpt001, ou=machines, dc=test, dc=local* »

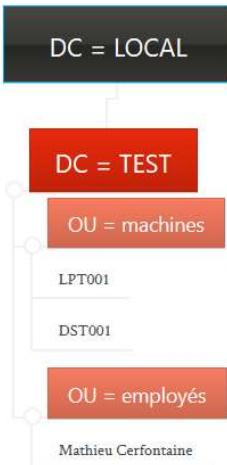


Figure 26 Arborescence d'un service d'annuaire.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Il est possible de s'authentifier sur le firewall à l'aide du protocole LDAP. Cela est particulièrement utile notamment dans le cadre de l'utilisation d'un VPN SSL (mode web et mode tunnel⁸). Pour cela il faut suivre plusieurs étapes de configuration.

En premier lieu il faut configurer toutes les informations du serveur LDAP. La plupart du temps c'est le contrôleur de domaine. Pour ce faire il faut se situer dans mode de configuration des « users » du firewall. Attention, ; ce terme peut porter à confusion. Ici, on ne configure pas un utilisateur mais un serveur LDAP, autrement dit, un annuaire d'utilisateurs.

```
config user ldap
    edit "s.ldap1"
        set server "172.10.1.1"
        set cnid "SamAccountName"
        set dn "dc=test,dc=local"
        set type regular
        set username "CN=svldap,OU=Services,DC=test,DC=local"
        set password ENC IHRvb+D8lsjINVd7/61cmdOLnTPizHJuf7a8Zs5vPSY64Ts182cmWHleosN7Sgnrg
        set password-expiry-warning enable
        set password-renewal enable
    next
```

Figure 27 Configuration d'un serveur LDAP sur le Fortigate.

Voici un tableau des paramètres les moins évidents et leurs descriptions :

Nom du paramètre	Description
CNID	Correspond à l'identifiant du « common name » du serveur LDAP. La plupart du temps cette valeur est « cn ». Cependant on peut aussi utiliser SamAccountName, uid ou autre.
DN	Le DN est le distinguished name. Il correspond dans ce cas au DN utilisé pour rechercher les entrées au sein du serveur LDAP. Il reflète la hiérarchie de la base de données des objets du serveur LDAP.

⁸ Le mode web permet d'accéder au VPN par http, on peut alors naviguer par l'intermédiaire d'une interface web ou de télécharger des logiciels complémentaires comme FortiClient. Le mode tunnel utilise justement le logiciel FortiClient pour créer une connexion VPN.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

Type	Permet de spécifier le type d'authentification pour les recherches LDAP. Il existe trois types : <ul style="list-style-type: none">- Anonymous : recherche via l'utilisateur « anonymous »,- Regular : rechercher via un nom d'utilisateur et un mot de passe entré en paramètre,- Simple : simple authentification par mot de passe.
Username	Le paramètre « Username » est disponible seulement si le type est configuré sur « regular ». Pour une authentification du type « regular » il faut un nom d'utilisateur et un mot de passe.

Pour permettre à un groupe d'utilisateurs d'utiliser le VPN SSL il faut alors faire le lien entre un groupe d'utilisateur sur le firewall et le « distinguished name » d'un groupe d'utilisateurs sur le serveur LDAP. Pour se faire on se positionne dans le mode de configuration des groupes d'utilisateurs :

```
config user group
    edit "s.vpnusers"
        set member "s.ldap1"
        config match
            edit 1
                set server-name "s.ldap1"
                set group-name "CN=gs-VpnUsers,OU=Accounts,DC=test,DC=local"
            next
        end
    next
end
```

Figure 28 Créeer un groupe d'utilisateur lié à un groupe sur le serveur LDAP.

Il ne reste plus qu'à configurer le VPN SSL et les « policies » permettant aux utilisateurs d'avoir accès au LAN et à Internet (« policies à double sens »).

La configuration des paramètres du VPN SSL se réalise dans le mode de configuration « config vpn ssl settings ». Ce mode contient toutes les informations concernant l'IP, les interfaces, les adresses, etc. nécessaires au fonctionnement du VPN.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

```
config vpn ssl settings
    set servercert "Fortinet_Factory"
    set tunnel-ip-pools "s.sslvpn"
    set dns-server1 10.0.0.1
    set port 82
    set source-interface "any"
    set source-address "all"
    set source-address6 "all"
    set default-portal "full-access"
end
```

Figure 29 Configuration des paramètres du VPN SSL.

Une fois les paramètres **configurer configurés**, il faut encore activer le « Web mode » et/ou le « Tunnel mode » avant de créer les policies. En réalité cette étape consiste juste à permettre à l'utilisateur d'accéder au VPN via l'interface web et/ou simplement par le tunnel en lui-même. On peut aussi configurer du « split-tunneling » qui permet d'assurer que seul le trafic ciblé accède au réseau local du VPN.

```
config vpn ssl web portal
    edit "full-access"
        set tunnel-mode enable
        set web-mode enable
        set ip-pools "s.sslvpn"
        set split-tunneling-routing-address "s.lan" "s.mgmt" "s.voice"
    config bookmark-group
        edit "gui-bookmarks"
        next
    end
next
end
```

Figure 30 Configuration du mode tunnel ET du mode web avec du "split-tunneling".

Inutile de s'attarder sur les policies. Il faut savoir qu'elles servent seulement à s'assurer que seuls les membres du groupe VPN et le réseau correspondant à ces membres puissent avoir accès au VPN pour accéder à internet et au LAN.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.7.2 SSL Web portal

Le portail Web du VPN SSL est, comme expliqué au chapitre précédent, une page web servant d'intermédiaire à la connexion VPN. Via ce portail on a la possibilité de télécharger des outils comme FortiClient, permettant d'implémenter le mode tunnel du VPN, d'accéder à des pages web d'internet de façon sécurisé par l'intermédiaire du VPN ou encore de créer des bookmarks qui sont en réalité des moyens de connexion rapide et sécurisé vers des ressources du réseau comme par exemple des serveurs de FTP ou des switches. Les figures ci-dessous illustrent l'apparence que peut prendre le portail web du Fortigate. Cependant cette apparence change en fonction des versions de l'OS.

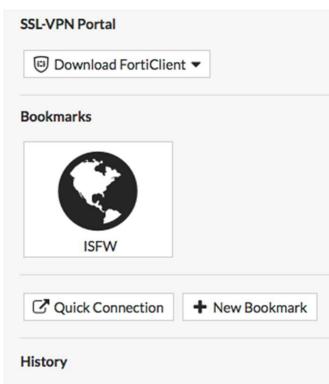


Figure 31 Créez des bookmarks, téléchargez FortiClient, etc ... (SSL VPN using web and tunnel mode - Fortinet Cookbook, 2015)



Figure 32 Le portail web permet d'accéder aux ressources du réseau via plusieurs protocoles différents. Il permet aussi l'accès à des pages web ou des serveurs FTP et tout cela de façon totalement sécurisée (SSL VPN using web and tunnel mode - Fortinet Cookbook, 2015).

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.7.3 Les « traffic shapers »

Un shaper, pour un firewall Fortigate, est un objet permettant de limiter la bande passante utilisée par une adresse IP ou un service donné. Cet objet est souvent utilisé dans des policies qui s'appliquent donc à des réseaux, des interfaces ou encore des services. Ce sont donc des objets indispensables pour gérer correctement un réseau. Avec des shapers on peut limiter la connexion internet provenant de sites web à certaines heures de la journée ou, inversement, l'augmenter jusqu'à son maximum à d'autres heures de la journée. Prenons par exemple le cas des réseaux sociaux qui peuvent être meurtriers en termes de productivité au travail. Il existe deux types de shapers : les shapers et les reverse shapers.

Les shapers affectent la vitesse en upload vers des destinations alors que les reverse shapers affectent la vitesse en download d'une destination vers la source.

Le fonctionnement est assez simple, lors de l'analyse d'une policy si celle-ci doit être appliquée à un trafic (cela peut donc concerner l'adresse IP d'un hôte ou d'un réseau, tout utilisateurs d'une interface, une période de la journée, etc...) alors les limites de la bande passante sont appliquées à ce trafic.

Il est possible de configurer plusieurs paramètres pour un shaper. La plupart du temps on se contente du paramètre « Maximum-bandwidth » et « guaranteed-bandwidth ». Le premier paramètre permet de limiter la bande passante à une valeur exprimée en Kbps. Le second paramètre permet d'offrir au trafic une valeur minimale pour cette bande passante. Ainsi on pourra, par exemple, assurer au trafic VoIP une qualité minimale requise pour un confort de conversation.

```
config firewall shaper traffic-shaper
    edit "s.low"
        set guaranteed-bandwidth 16
        set maximum-bandwidth 16776000
        set priority low
    next
    edit "s.medium"
        set guaranteed-bandwidth 16
        set maximum-bandwidth 16776000
        set priority medium
    next
    edit "s.high"
        set guaranteed-bandwidth 16
        set maximum-bandwidth 16776000
    next
end
```

Figure 33 Shapers créés pour limiter la bande passante selon trois niveaux. "Accès limité", "Accès médium" et "Accès privilégié".

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

6.8 En résumé

La configuration du firewall de chez Fortinet, le Fortigate, s'est avérée très simple à programmer pour YASC. En effet, la structure du système facilite la gestion des configurations actuelles. De plus, le firewall répond très bien aux commandes SSH envoyées par le module PoshSSH ce qui a permis améliorer davantage la technique utilisée à la base sur le switch HP.

Ce travail arrive donc à sa fin et il est temps de le conclure. Voici donc deux conclusions : une conclusion technique et la conclusion personnelle de l'auteur.

7 Conclusions.

7.1 Conclusion technique.

Le travail d'administrateur des réseaux est un travail très dense qui nécessite des compétences dans beaucoup de champs d'activités informatiques. La gestion d'un réseau met à rude épreuve les connaissances d'un informaticien d'un point de vue aussi bien logiciel que matériel.

En passant par la configuration d'un switch ou d'un firewall, par la gestion d'un serveur de mail ou d'un parc informatique de serveur Windows ou LINUX, les ressources mises en œuvre à l'heure actuelle pour coordonner les activités des différentes entreprises autour de la planète sont tout simplement immenses. Ces ressources demandent toujours plus de connaissances et de temps de la part des administrateurs informatiques.

Avec l'avancée des technologies, on retrouve toujours ce besoin d'automatiser toujours plus les tâches difficiles et fastidieuses. Ce constat n'a pas attendu l'ère numérique pour apparaître. En effet c'est le propre de l'homme de vouloir se faciliter la tâche.

YASC est donc un outil créé pour faciliter l'administration des systèmes et des réseaux. Son but est de pouvoir se concentrer sur les tâches importantes d'une entreprise, épargnant alors du temps sur des petites tâches comme les configurations de matériel et/ou réseau.

YASC n'est pas le seul outil de gestions de configuration existant sur le marché. Cependant il a le mérite de vouloir offrir, à terme, une gestion complète d'un environnement Windows. Il permet aussi la génération d'une documentation servant de support pour les dépannages, offrant ainsi non seulement un gain de temps sur les déploiements mais aussi une efficacité augmenter-augmentée lors d'un dépannage réseau ou logicielle. Ce dernier point est très important car un réseau informatique inactif à l'heure d'aujourd'hui signifie des pertes immenses pour une entreprise.

Pour conclure, YASC est un outil à grand potentiel qui doit évidemment encore mûrir. De nombreuses cmdlets peuvent encore lui être rajouter-rajoutées. Cependant la priorité est de pouvoir gérer complètement les tâches effectuées en entreprise et qui consomment du temps inutilement comme le déploiement de serveurs, la création des machines virtuelles sur ceux-ci ou, pourquoi pas, l'installation d'OS quelconque. Une interface web facilitera au passage l'utilisation du software qui reste encore, malheureusement, fort réservé aux développeurs. Il est aussi envisagé de créer un logiciel « maître d'orchestre » qui utilisera YASC comme un daemon et qui permettra d'orchestrer les déploiements. Une chose est sure-sûre : YASC a encore beaucoup d'avenir devant lui et de potentiel à développer."

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

7.2 Conclusion personnelle.

Depuis le premier jour où j'ai posé la main sur un ordinateur, je suis fasciné par la puissance de ces machines. Le choix d'entamé des études en informatiques a été pour moi le meilleur des choix. J'ai donc réalisé une première année à la Haute Ecole de la Province de Liège en options « Informatique et systèmes ». Durant cette année j'ai reçu différents cours généraux d'informatique. Celui qui m'a le plus parlé a été le cours de réseau. Durant ce cours j'ai appris comment fonctionnais-fonctionnait un réseau de manière générale et comment utiliser des machines CISCO.

Ce cours m'a permis de développer un intérêt pour la télécommunication et les réseaux. Cette capacité, offerte par l'informatique et l'électronique, de pouvoir travailler avec n'importe qui, n'importe où dans le monde m'a passionné. Depuis ce jour je n'ai eu de cesse de me frayer un chemin dans le métier d'administrateur réseau et système en commençant par en faire ma spécialisation durant ma deuxième et ma troisième année à la Haute Ecole.

Par ensuite-la suite j'ai obtenu un stage en développement dans une entreprise d'infogérance : Limelogic. Ce stage m'a permis non seulement d'appréhender le métier d'administrateur réseau et système mais il m'a aussi permis de m'insérer dans le monde professionnel. Cette expérience a été l'une des plus riches en apprentissage. Elle m'a permis de me développer en tant qu'adulte responsable et en tant qu'informaticien.

Ce travail de fin d'étude conclut donc trois années passée à l'école à me former pour devenir informaticien systèmes spécialisés en réseaux et télécommunications. Durant ces trois années j'ai eu l'occasion d'apprendre énormément de choses passionnantes au côté de mes collègues de classes et de mes professeurs qui ont par ailleurs toujours été à l'écoute.

Cependant, je ne considère pas la fin de mes études comme une finalité mais plutôt comme un début. Il me reste énormément à apprendre du monde des télécommunications. Ce métier, je l'espère et j'en suis sûr, m'offrira la possibilité d'évoluer et de découvrir de nouvelles choses. Dans ce monde où l'on invente chaque jour de nouveau outils pour travailler et communiquer ensemble, il ne manquera pas de sujets passionnants à apprivoiser qui ne seront jamais assez nombreux pour étancher ma soif de curiosité.

« *L'enseignement devrait être ainsi : celui qui le reçoit le recueille comme un don inestimable mais jamais comme une contrainte pénible* ». - (Mein Weltbild ; The World As I See It, 1949)

8 Annexe

8.1 Les paramètres dynamiques

En PowerShell, un paramètre dynamique est un paramètre qui est créé au moment de l'exécution (Runtime). Ces paramètres dynamiques peuvent « voir » les paramètres statiques. Cela signifie que l'on peut configurer leurs valeurs ainsi que bien d'autres attributs en fonction des valeurs d'un paramètre statique. Cependant les paramètres dynamiques ne sont pas visibles dans l'aide de la cmdlet (Get-Help).

8.1.1 Structure d'une cmdlet à paramètres dynamiques

Généralement lorsque l'on crée une cmdlet classique, avec des paramètres statiques, sa structure se compose du bloc d'instructions « function », permettant de déclarer une fonction au sein de la cmdlet, et du bloc d'instructions « param » qui permet de déclarer les paramètres de la fonction :

```
function CmdLetName()
{
    param(
        [parameter(Mandatory=$true,HelpMessage="Name.")] [string]$Name
    )
    #Corps de la cmdlet.
    Write-Verbose -Message "My name is : $Name" -Verbose
}
```

Exemple 48 Structure d'une cmdlet à une fonction.

Une cmdlet peut être composée de différentes fonctions. Cependant il est de bonne pratique de créer un fichier script par fonction pour faciliter le débogage.

Pour créer une cmdlet utilisant des paramètres dynamiques il faut rajouter deux composants : le bloc d'instruction « DynamicParam » et « Process ». En premier lieu il faut utiliser le bloc « DynamicParam ». Ce mot-clé réservé du langage PowerShell permet de créer une portée dans laquelle seront créés les paramètres dynamiques au moment du « Runtime ». Le second composant est le bloc « Process » qui contiendra le corps de la fonction en elle-même :

```
function CmdLetName()
{
    param(
        [parameter(Mandatory=$true,HelpMessage="Name.")] [string]$Name
    )
    DynamicParam{
        #Création des paramètres dynamiques
    }
    process{
        #Corps de la fonction
        Write-Verbose -Message "My name is : $Name" -Verbose
    }
}
```

Exemple 49 Cmdlet utilisant des paramètres dynamiques.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

8.1.2 Création de paramètres dynamiques

La création et la configuration des paramètres dynamiques d'une cmdlet se situent dans la portée « DynamicParam ». Il y a plusieurs façons de procéder mais globalement on utilise toujours les mêmes objets. Il est possible de modifier plusieurs attributs d'un paramètre comme son nom, son « HelpMessage », ses « ValidateSet », etc. Chaque paramètre est ensuite ajouté au dictionnaire de paramètres de la cmdlet.

```
DynamicParam{
    # Création des paramètres dynamiques
    # Donner au paramètre "FirstName" son label.
    $parameterName = 'FirstName'

    # Créer le dictionnaire de paramètres.
    $runtimeParameterDictionary = New-Object System.Management.Automation.RuntimeDefinedParameterDictionary

    # Créer une collection d'attributs
    $attributeCollection = New-Object System.Collections.ObjectModel.Collection[System.Attribute]

    # Créer des attributs pour le paramètre comme "Mandatory" ou "HelpMessage"
    $parameterAttribute = New-Object System.Management.Automation.ParameterAttribute
    $parameterAttribute.Mandatory = $true
    $parameterAttribute.HelpMessage = "Firstname"

    # Ajouter les attributs à la collection d'attributs
    $attributeCollection.Add($parameterAttribute)

    # Générer un ValidateSet pour le nouveau paramètre dépendant du paramètre "Name".
    $arrSet = @()
    switch($accessMethods)
    {
        'Armstrong'{
            $arrSet+= 'Lance'
            $arrSet+= 'Neil'
            $arrSet+= 'Louis'
        }
    }

    if($arrSet.Length -gt 0){
        # Créer l'objet ValidateSet et l'initialiser avec le tableau.
        $validateSetAttribute = New-Object System.Management.Automation.ValidateSetAttribute($arrSet)

        # Ajouter l'objet ValidateSet à la collection d'attributs.
        $attributeCollection.Add($validateSetAttribute)

        # Créer un objet représentant le paramètre dynamique, lui donner le nom du paramètre, son type et sa collection d'attributs.
        $runtimeParameter = New-Object System.Management.Automation.RuntimeDefinedParameter($parameterName, [string], $attributeCollection)

        # Ajouter l'objet au dictionnaire des paramètres.
        $runtimeParameterDictionary.Add($parameterName, $runtimeParameter)
    }
}
```

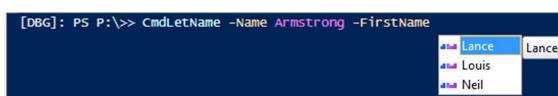
Exemple 50 Création d'un paramètre dynamique.

Ensuite pour accéder aux paramètres ajoutés dans la portée « Process » il faut passer par le dictionnaire de paramètres :

```
process{
    #Corps de la fonction
    $firstName = $runtimeParameterDictionary.FirstName.value
    write-verbose -Message "My name is $Name and my firstname is $firstName." -verbose
}
```

Exemple 51 Utilisation du paramètre dynamique dans la portée "Process".

Lors de l'appel de la cmdlet si le premier paramètre prend la bonne valeur on peut voir apparaître-apparaître le paramètre dynamique et ses « ValidateSet » :



Exemple 52 Appel d'une cmdlet avec des paramètres dynamiques.

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

8.2 Déclarer une collection de données dans le fichier des métadonnées

Voici un exemple de déclaration de collection de données dans le fichier des métadonnées. Il concerne la cmdlet « Set-LDPyascFortiGWebFilter » permettant de configurer un filtre web sur base de catégorie/type de page web. Cette collection de données contient toutes les catégories utilisées sur un firewall Fortigate et les chiffres entiers correspondant à ces catégories.

```
<Categories>
    <Coll_Categories>
        <Unrated>0</Unrated>
    </Coll_Categories>
    <Coll_Categories>
        <DrugAbuse>1</DrugAbuse>
    </Coll_Categories>
    <Coll_Categories>
        <AlternativeBeliefs>2</AlternativeBeliefs>
    </Coll_Categories>
    <Coll_Categories>
        <Hacking>3</Hacking>
    </Coll_Categories>
    <Coll_Categories>
        <IllegalorUnethical>4</IllegalorUnethical>
    </Coll_Categories>
    <Coll_Categories>
        <Discrimination>5</Discrimination>
    </Coll_Categories>
    <Coll_Categories>
        <ExplicitViolence>6</ExplicitViolence>
    </Coll_Categories>
    <Coll_Categories>
        <Abortion>7</Abortion>
    </Coll_Categories>
    <Coll_Categories>
        <OtherAdultMaterials>8</OtherAdultMaterials>
    </Coll_Categories>
    <Coll_Categories>
        <AdvocacyOrganizations>9</AdvocacyOrganizations>
    </Coll_Categories>
    ...
</Categories>
```

Exemple 53 Déclaration d'une collection de données dans le fichier XML des métadonnées : YascMeta.xml

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

9 Bibliographie

An overview of chef. (s.d.). Récupéré sur chef.io: https://docs.chef.io/chef_overview.html

Bash Scripting Tutorial - 1. What is a Bash Script? (2017). Récupéré sur ryanstutorials.net: <http://ryanstutorials.net/bash-scripting-tutorial/bash-script.php>

Blawat, B. J. (2015). *Mastering Windows PowerShell Scripting*. Birmingham: Packt Publishing Ltd.

Chef (logiciel) - Wikipedia. (2017, janvier 1). Récupéré sur Wikipedia: [https://fr.wikipedia.org/wiki/Chef_\(logiciel\)](https://fr.wikipedia.org/wiki/Chef_(logiciel))

chef-solo - Chef Docs. (s.d.). Récupéré sur chef.io: https://docs.chef.io/chef_solo.html

Ciaccio, R. S. (2010, décembre 18). *Powershell vs unix shell*. Récupéré sur Superuser: <https://superuser.com/questions/223300/powershell-vs-the-unix-shell>

Configuring 802.1X Port-Based Authentication. (s.d.). Récupéré sur cisco.com: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750/software/release/12-1_19_ea1/configuration/guide/3750scg/sw8021x.pdf

EAP - Wikipedia. (2017, mars 9). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Extensible_Authentication_Protocol#Radius

Einstein, A. (1949). *Mein Weltbild ; The World As I See It.* (Flammarion, Éd.)

Fortinet - Wikipedia. (2017, avril 9). Récupéré sur Wikipedia: <https://fr.wikipedia.org/wiki/Fortinet>

Gérer vos codes sources avec GIT. (2017, janvier 12). Récupéré sur openclassrooms: <https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git>

GitHub. (s.d.). Récupéré sur GitHub.com: <https://github.com/>

IEEE 802.1x - Wikipedia. (2017, mars 8). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/IEEE_802.1X

Interface en ligne de commande - Wikipedia. (2016, décembre 27). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Interface_en_ligne_de_commande

Internet Protocol Security - Wikipedia. (2017, mars 25). Récupéré sur Wikipedia: https://fr.wikipedia.org/wiki/Internet_Protocol_Security#R.C3.A9f.C3.A9rences

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

IPSec phase 1 and phase 2. (s.d.). Récupéré sur brocade.com:

<http://www.brocade.com/content/html/en/vrouter5600/40r1/vrouter-40r1-ipsecvpn/GUID-0B3591F2-F0FE-4F64-ABF9-A3B5F05ABD96.html>

Langage de script - Wikipedia. (2016, novembre 8). Récupéré sur Wikipedia:

https://fr.wikipedia.org/wiki/Langage_de_script

Les cmdlets. (s.d.). Récupéré sur univ-mlv: <http://igm.univ-mlv.fr/~dr/XPOSE2008/Introduction%20au%20Powershell/cmdlets.html>

Lightweight Directory Access Protocol - Wikipedia. (2017, mars 29). Récupéré sur Wikipedia:

https://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Sch.C3.A9ma

Perl (langage) - Wikipedia. (2017, mars 4). Récupéré sur Wikipedia:

[https://fr.wikipedia.org/wiki/Perl_\(langage\)](https://fr.wikipedia.org/wiki/Perl_(langage))

Puppet Enterprise user's guide. (2017, janvier). Récupéré sur puppet.com:

<https://docs.puppet.com/pe/latest/index.html>

Python (langage) - Wikipedia. (2017, mars 16). Récupéré sur Wikipedia:

[https://fr.wikipedia.org/wiki/Python_\(langage\)](https://fr.wikipedia.org/wiki/Python_(langage))

Qu'est-ce que Perl ? (s.d.). Récupéré sur openclassrooms:

<https://openclassrooms.com/courses/apprenez-a-programmer-en-perl/qu-est-ce-que-perl>

Qu'est-ce que Python ? (s.d.). Récupéré sur openclassrooms:

<https://openclassrooms.com/courses/apprenez-a-programmer-en-python/qu-est-ce-que-python>

Rudder (logiciel) - Wikipedia. (2017, janvier 17). Récupéré sur Wikipedia:

[https://fr.wikipedia.org/wiki/Rudder_\(logiciel\)](https://fr.wikipedia.org/wiki/Rudder_(logiciel))

Rudder 4.0 - User's manual. (2016). Récupéré sur rudder-project.org: <http://www.rudder-project.org/rudder-doc-4.0/rudder-doc.pdf>

SSL VPN using web and tunnel mode - Fortinet Cookbook. (2015, décembre 23). Récupéré sur cookbook.fortinet.com: <http://cookbook.fortinet.com/ssl-vpn-using-web-and-tunnel-mode-54/>

SUBSYSTEM (Specify Subsystem). (2017). Récupéré sur microsoft.com:

<https://msdn.microsoft.com/en-us/library/fcc1zstk.aspx>

Gestion et déploiement d'applications et d'équipement réseaux via YASC.

What is shell ? - gnu.org. (2016, septembre 7). Récupéré sur gnu:

http://www.gnu.org/software/bash/manual/bashref.html#What-is-Bash_003f

Windows Powershell -Wikipedia. (2016, décembre 16). Récupéré sur Wikipedia:

https://fr.wikipedia.org/wiki/Windows_PowerShell

YASC (YET ANOTHER SOFTWARE CONFIGURATOR) DOCUMENTATION. (2016). Liège:

Limelogic.