Maintenance de stockage via Powershell

Une image contenant personne, technologie, intérieur, ordinateur portable

Description générée automatiquement

Figure - Image générée par Bing Copilot représentant le projet

Yann Mangiagli – FIN2

Vennes

88H

Gaël Sonney

Experts : Daniel Berney, Gabriel Maret

Table des matières

[1 Résumé de la documentation / résumé du rapport 3](#_Toc167443364)

[1.1 Comptes utilisés 3](#_Toc167443365)

[1.2 Résumé 3](#_Toc167443366)

[2 Spécifications 4](#_Toc167443367)

[2.1 Titre 4](#_Toc167443368)

[2.2 Description 4](#_Toc167443369)

[2.3 Matériel et logiciels à disposition 4](#_Toc167443370)

[2.4 Prérequis 4](#_Toc167443371)

[2.5 Cahier des charges 4](#_Toc167443372)

[2.5.1 Objectifs et portée du projet (objectifs SMART) 4](#_Toc167443373)

[2.5.2 Caractéristiques des utilisateurs et impacts 5](#_Toc167443374)

[2.5.3 Fonctionnalités requises (du point de vue de l’utilisateur) 5](#_Toc167443375)

[2.5.4 Contraintes 5](#_Toc167443376)

[2.5.5 Travail à réaliser par l'apprenti 5](#_Toc167443377)

[2.5.6 Si le temps le permet … 6](#_Toc167443378)

[2.5.7 Méthodes de validation des solutions 6](#_Toc167443379)

[2.6 Les points suivants seront évalués 6](#_Toc167443380)

[2.7 Validation et conditions de réussite 6](#_Toc167443381)

[3 Planification Initiale 6](#_Toc167443382)

[3.1.1 Méthode des 6 pas 6](#_Toc167443383)

[3.1.2 Mises à jour du planning 7](#_Toc167443384)

[4 Analyse 7](#_Toc167443385)

[4.1 Opportunités 7](#_Toc167443386)

[4.1.1 Approfondir 7](#_Toc167443387)

[4.1.2 Difficultés potentielles 7](#_Toc167443388)

[4.1.3 Matériel à exploiter 8](#_Toc167443389)

[4.1.4 Recherche d’informations particulière 8](#_Toc167443390)

[4.1.5 Solutions possibles 8](#_Toc167443391)

[4.2 Document d’analyse et conception 9](#_Toc167443392)

[4.2.1 Script 1 9](#_Toc167443393)

[4.2.2 Script 2 12](#_Toc167443394)

[4.3 Conception des tests 15](#_Toc167443395)

[4.4 Planification détaillée 15](#_Toc167443396)

[5 Réalisation 15](#_Toc167443397)

[5.1 Dossier de Réalisation 15](#_Toc167443398)

[5.1.1 Version des outils logiciels utilisés 15](#_Toc167443399)

[5.1.2 Configuration spéciale des serveurs 16](#_Toc167443400)

[5.2 Modifications 25](#_Toc167443401)

[6 Tests 25](#_Toc167443402)

[6.1 Dossier des tests 25](#_Toc167443403)

[7 Conclusion 27](#_Toc167443404)

[7.1 Bilan des fonctionnalités demandées 27](#_Toc167443405)

[7.2 Bilan de la planification 27](#_Toc167443406)

[7.3 Bilan personnel 27](#_Toc167443407)

[8 Divers 28](#_Toc167443408)

[8.1 Journal de travail 28](#_Toc167443409)

[8.2 Bibliographie 28](#_Toc167443410)

[8.3 Webographie 28](#_Toc167443411)

[9 Annexes 28](#_Toc167443412)

# Résumé de la documentation / résumé du rapport

## Comptes utilisés

|  |  |  |
| --- | --- | --- |
| **Utilisation** | **Compte** | **Mot de passe** |
| Envoi du mail automatisé | pappro2mail@gmail.com | .Etml-123 |
| Réception du mail automatisé | tpiymetml@gmail.com | .Etml-123 |
| Compte serveur YANN2K22SRV | Administrateur | .Etml-123 |
| Compte serveur YANN2K22SRV2 | Administrateur | .Etml-123 |

## Résumé

Le but du projet est de créer 2 scripts complémentaires. Le script 1 est un exécutant tandis que le script 2 sert à le propager automatiquement.

# Spécifications

## Titre

Maintenance automatique et régulière du stockage des postes de travail à distance

## Description

Les étudiants de la section informatique sont amenés à utiliser de nombreuses machines virtuelles pendant leur formation qu’ils et elles laissent sur leur lecteur (H :) qui leur servent de répertoire utilisateur sans les utiliser de nouveau. Ces machines virtuelles sont donc stockées en vain sur les machines de l’ETML plutôt que sur le SSD des étudiants et prennent beaucoup de place avec le temps. Le but de ce projet est de limiter l’impact des machines virtuelles sur le stockage des postes de l’école en avertissant quand le stockage d’un poste atteint 80% et en envoyant un mail au responsable qui pourra par la suite décider de supprimer les fichiers OVA de plus de 3Go des répertoires des élèves s’étant connecté à cette machine.

## Matériel et logiciels à disposition

PC standard de l’ETML sous Windows 10

PowerShell

PowerShell ISE

Deux machines distantes (WINSERV2022) pour le déploiement du script à distance avec les droits d’administrateur configurées par l’apprenti en amont.

## Prérequis

Module 431 - Planification de projets et de tâches

Module 122 – PowerShell

Module 117 – Réseau (Windows Server 2022, Câblage simple, Remote Desktop)

## Cahier des charges

### Objectifs et portée du projet (objectifs SMART)

Combien

Quoi

Qui

Comment

Où

Quand

Pourquoi

Spécifique

Mesurable

Atteignable

Réaliste : Tous les élèves doivent quand même pouvoir poser des fichiers volumineux sans que ceux-ci ne se fassent supprimer s’ils ne sont pas des ova.

Temporel : Le projet doit être fait en 88H , du lundi 13 mai 2024 au lundi 3 juin 2024

### Caractéristiques des utilisateurs et impacts

Les utilisateurs du script seraient des administrateurs système et réseau désirant optimiser le stockage des postes de travail supportant des machines virtuelles, comme le service informatique de l’ETML. Le but est donc d’automatiser les scripts tout en laissant aux administrateurs le moyen de personnaliser l’utilisation du script avec des paramètres. Par exemple, choisir la taille maximum tolérée par le script (par défaut 3Go comme mentionné plus haut).

### Fonctionnalités requises (du point de vue de l’utilisateur)

Pour le premier script, l’utilisateur doit pouvoir récupérer les informations concernant l’espace disque utilisé ainsi que l’espace libre, le nom de l’ordinateur, la version de son système, ainsi que les versions des mises à jour de l’OS. Le script envoie un courriel sur le mail d’entreprise de l’administrateur un avertissement quand 80% du stockage est utilisé, et doit pouvoir supprimer les fichiers trop volumineux, avec pourquoi pas un paramètre permettant de décider ce qui est considéré comme « trop volumineux ». Le deuxième script est plus simple et contient moins de fonctionnalités. L’utilisateur a juste besoin de le lancer pour qu’il copie le script 1 sur une machine à distance ne le possédant pas et qu’il crée la tâche planifiée correspondante. Le script peut gérer une liste de 10 ordinateurs en simultané pour son exécution.

### Contraintes

Plusieurs contraintes peuvent exister avec des scripts comme ceux-là. Par exemple, le mail envoyé par le script a une chance non négligeable de finir dans les spam, car il n’y a pas le temps d’installer un serveur SMTP et c’est pour cela qu’une adresse Gmail sera utilisée, car il est plus simple et plus rapide de se connecter via PowerShell sur ce serveur SMTP. Comme la sécurité n’est pas mentionnée dans le cahier des charges, la sécurité du script ne sera pas forcément optimale car cela ne sera pas une priorité. Par exemple, pour automatiser l’enregistrement des identifiants en utilisant la commande « get-credential », il faudra passer par écrire le mot de passe en clair dans le script pour ne pas perdre du temps là-dessus. Cela pourrait être un point à améliorer s’il reste du temps quand les 2 scripts seront prêts.

### Travail à réaliser par l'apprenti

L’apprenti doit réaliser 2 scripts ainsi qu’un rapport, une planification et un journal de travail. Le premier script doit récupérer des informations système comme le nom, version de l’OS, Espace disque libre & utilisé, version des mises à jour et envoyer un mailcontenant les informations permettant d’identifier le poste de travail à l’administrateur quand le stockage utilisé égale ou dépasse les 80%. Il sera ensuite possible de supprimer en priorité les fichiers de machine virtuelles, en particulier les fichiers avec une extension ova. Le deuxième script est une automatisation. Il permet de sélectionner une liste jusqu’à 10 ordinateurs distants, et d’implémenter le premier script afin qu’il puisse scanner et surveiller le stockage des postes de travail sur lesquels il vient de s’installer, tout en créant une tâche planifiée qui planifiera ce scan. Le rapport doit contenir toute la documentation qu’il a créée durant le projet ainsi que les tests réalisés.

### Si le temps le permet …

Il serait judicieux d’améliorer la sécurité pour rendre le script utilisable par une entreprise ou une école, ainsi qu’installer un serveur SMTP pour ne pas avoir à utiliser une adresse gmail mais une adresse eduvaud dans l’exemple de l’ETML utilisé comme prétexte pour ce TPI.

### Méthodes de validation des solutions

Chaque fonctionnalité demandée dans le script sera testée en premier lieu seule pour être sûr que la solution fonctionne avant de l’incorporer dans le script 1 ou 2 suivant l’instruction(s) testée(s). Selon la commande, elle devra être testée sur une machine distante, comme la commande de suppression des fichiers ova de 3Go. Le maximum de résultats de tests seront sujet à une capture d’écran montrant le résultat. Ils seront aussi documentés dans la partie réalisation du rapport.

## Les points suivants seront évalués

* Le rapport
* Les planifications (initiale et détaillée)
* Le journal de travail
* Le code et les commentaires
* Les documentations de mise en œuvre et d’utilisation

## Validation et conditions de réussite

* Compréhension du travail
* Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
* Etat de fonctionnement du produit livré

# Planification Initiale

Le seul jour spécial est pentecôte, ce qui rend le lundi 20 mai 2024 chômé. Le lendemain, l’expert 2 vient voir l’avancée du projet vers 13h30.

### Méthode des 6 pas

Le projet commence le 13 mai 2024 et se termine le lundi 3 juin 2024. La méthode des 6 pas a été choisie pour ce projet. Cette méthode de planification consiste à se fixer des objectifs

**S’informer :**

Comprendre et enregistrer la tâche à accomplir, se renseigner sur le travail, sur ce qui est attendu, ce qui est « libre ». C’est dans ce moment-là qu’on se renseigne sur l’échéance, sur si on est capable ou non de réaliser ce projet, le matériel à disposition… Dans ce projet, on peut faire le parallèle avec la prise de connaissance du cahier des charges, les questions posées au chef de projet et aux experts en rapport avec ce document, la configuration des deux machines Windows server 2022 utilisées dans le projet comme machines distantes.

**Planifier :**

C’est tout ce qui concerne la planification du projet pour en arriver à bout. Il faut donc se fixer des priorités dans le projet, faire en premier lieu ce qui est le plus important comme les fonctionnalités principales du script avant la gestion des erreurs par exemple. En gros, quelles tâches réaliser en premier pour avoir le plus vite possible un prototype fonctionnel. Il est aussi décidé quelle méthode de gestion de projet est utilisée, comme agile, 6 pas, etc. Il faut aussi estimer le temps que chaque tâche prendra. C’est un exercice très difficile car on ne peut pas prévoir tous les problèmes qu’un projet va inévitablement rencontrer, ni leur gravité, comme une perte totale des données par exemple.

**Décider :**

Que le projet soit à réaliser seul(e) ou à plusieurs, il est impératif d’avoir un consensus avec tous les partis impliqués dans le projet. À plusieurs, il faut être d’accord entre collègues, pour pas que chacun fasse quelque chose de différent qui fasse juste perdre du temps.

**Réaliser / exécuter :**

Partie principale du projet. C’est à ce moment qu’on travaille sur le projet en le réalisant. Dans l’idéal, il faudrait que ça se déroule exactement comme dans la planification. Pour faire un parallèle avec ce projet, c’est la création des scripts.

**Contrôler :**

On regarde le travail réalisé plus haut et on le vérifie. On reteste, on relit la documentation et on la modifie si besoin.

**Évaluer :**

Passage en revue du projet effectué, on parle de ce qui a été bien fait, de ce qui a été mal fait, de ce qui aurait pu être amélioré. Dans ce rapport on pourrait assimiler cette étape au bilan des fonctionnalités demandées dans la conclusion.

### Mises à jour du planning

14.05.24 :

Modification de la rencontre avec expert 2, décidée arbitrairement dans la planification initiale par l’apprenti pour le 23 mai avant d’être modifié au 21 quand l’expert 2 et l’élève se sont mis d’accord.

Ce paragraphe présente le planning d'origine (date de début, date de fin, vacances et congés, liste hiérarchique des tâches ou GANTT, jalons, durée totale)

# Analyse

## Opportunités

### Approfondir

Il faudra approfondir certaines connaissances en PowerShell pour mener à bien le projet.

* Commande « New-ScheduledTaskAction » à apprendre à utiliser
* Apprendre à exécuter les scripts depuis un autre (Copy-Item ?)

### Difficultés potentielles

Plusieurs difficultés peuvent survenir, surtout à cause d’imprévus.

* Malgré le script d’envoi de mail fonctionnel réalisé en amont, rien ne promet qu’il fonctionnera dans un contexte différent
* Comment vérifier que le script 1 s’exécute bien à 80% alors que les machines à disposition ont des disques vides ?
* La copie du script sur une liste de machines n’a pas l’air simple à mettre en place
* Pour le script de copie, il faut copier un script sur la machine principale et le coller sur la machine distante
* Un serveur qui ne réagit pas comme l’autre
* Il faut internet pour envoyer un mail

### Matériel à exploiter

Le matériel exploitable peut lui aussi causer des difficultés, par exemple un serveur mal paramétré en amont, ou encore si le réseau de l’école est changé. C’est pour cela qu’il est impossible d’utiliser le script depuis une machine distante sur la machine locale, car elle a besoin de droit administrateur que je ne possède pas, et elle est protégée derrière docker kubernetes.

### Recherche d’informations particulière

Comme PowerShell et ses erreurs sont bien documentées, il y’a peu de chances de rencontrer des difficultés de ce côté-ci. En revanche, un problème d’environnement rendrait tout de suite la tâche plus ardue.

### Solutions possibles

* Réduction du quota de 80% à 3% par exemple
* Une boucle « foreach » sur un tableau ayant la liste des ordinateurs
* Supprimer Invoke-Command, car on essayait d’accéder à la session depuis la session, c’est donc normal que ça ne fonctionnait pas
* Les pares-feux des 2 serveurs n’étaient pas identiques. Le premier était désactivé et pas l’autre.

## Document d’analyse et conception

### Schéma de conception

Une image contenant texte, diagramme, Plan, ligne

Description générée automatiquement

Figure - Schéma démontrant comment les scripts ont été pensés

### Script 1

Pour le script 1, des tests ayant été faits durant un projet précédent ont mis en exergue le fait qu’utiliser Send-MailMessage était obsolète et n’était plus supporté par Microsoft. Malgré le fait que des alternatives existent (MimeKit & MailKit), il a été décidé de rester sur cette façon de faire car elle est plus simple bien que moins sécurisée. Je suis parti du principe que vu que le script 2 permettra de copier le script à distance et de le faire s’exécuter périodiquement via une tâche planifiée, le script 1 n’a pas besoin d’être utilisable à distance ; il suffira de lancer le script 2 avant pour le copier sur la machine distante.

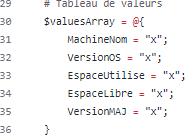


Figure - Création du tableau

Un tableau associatif est créé afin de lier chaque valeur récupérée dans le script pour pouvoir tout imprimer dans le mail d’un coup grâce à une boucle foreach. La lettre du disque est pour le moment en commentaire car peut-être qu’elle ne servira pas. Bien qu’il soit possible de directement mettre les valeurs dans le tableau, j’ai préféré les rajouter plus tard afin d’avoir un code plus lisible.



Figure - Variable d'environnement du nom de l'ordinateur

Le nom de la machine est stocké dans une variable qu’on appelle variable d’environnement, d’où le $env : devant. Elles existent déjà dans le système, et la variable $env :COMPUTERNAME donne le nom de l’ordinateur qui exécute le script. Elle est utilisée ici pour être entrée dans le tableau associatif la liant à l’indice MachineNom.



Figure - Variables d'environnement de la version de l'OS

Même concept que pour la valeur de la ligne 39, mais avec deux valeurs. $env :OS donne le nom du système d’exploitation (dans le contexte de la machine sur laquelle je fais mes tests, cela équivaut à Windows NT, tandis que la seconde valeur contient le numéro de build de la version). Cela permet d’avoir un résultat plus esthétique, comme suit sans les guillemets : « Version de l’OS : Windows\_NT 19045 ».



Figure - Enregistrement de l'espace libre

Une valeur temporaire est créée à la ligne 45 qui sera réinstanciée pus tard pour économiser la mémoire en ne créant pas trop de variables pour rien. Cette instruction récupère l’espace libre stocké dans Get-PSDrive puis stocke la partie Free du tableau de PSDrive dans $valuesArray avec la clé EspaceLibre.



Figure - Enregistrement de l'espace utilisé

Comme la valeur temporaire a été utilisée, et que sa valeur a été inscrite dans le tableau, on peut la réutiliser sans perdre les données. Le script utilise donc les mêmes instructions que pour l’espace libre mais on remplace la valeur recherchée par Used. On aura donc l’espace utilisé dans les disques.



Figure - Enregistrement de l'historique des versions de mise à jour

La variable $tempValue est de nouveau réinitialisée en mettant une nouvelle valeur à l’intérieur qui représente la liste de toutes les mises à jour réalisées sur le système.



Figure - Calcul du stockage utilisé

Dans l’intervalle des lignes (53-54) se trouve une valeur permettant de lancer un mail même si le stockage n’atteint pas 80% à des fins de débogage. Pour les lignes 56 et 57, l’instruction renvoie plusieurs valeurs.



Figure - Contenu de $usedSpace

Il faudra donc rajouter un « .Sum » à la fin de $usedSpace pour avoir la valeur voulue.



Figure - Calcul de l'espace libre entre tous les disques

La même stratégie est appliquée pour l’espace libre.



Figure - Calcul du stockage total

Vient ensuite l’addition de la somme du stockage plein et du stockage vide afin d’avoir le stockage total pour calculer le pourcentage du stockage utilisé en octets, qui est ensuite stocké dans une variable.



Figure - Calcul en % de l'espace utilisé par rapport au stockage total

Le calcul pour récupérer le pourcentage est simple. On multiplie par 100 la somme du stockage utilisée, puis on divise par le stockage total. Par exemple, sur la machine de l’école en conditions réelles, le stockage est utilisé à environ 63.62%.



Figure - Condition pour faire fonctionner la suite du script

Le script effectue une vérification du pourcentage de stockage utilisé. S’il est de 80% ou plus, la suite du script s’exécute, sinon le script se termine. Tout ce qui suit est donc contenu dans le bloc de la condition. On pourrait améliorer les performances du script en faisant en sorte que le script aille chercher les infos que si le pourcentage de stockage autorisé et dépassé, comme ça pour la plupart du temps le script se fera plus vite.



Figure - Informations du serveur SMTP

On donne ici les premières informations importantes sur le serveur comme le port qui doit être à 587 pour fonctionner avec -UseSSL afin que Gmail permette d’envoyer le mail via leur serveur smtp.



Figure - Informations de l'expéditeur

Pour ce qui est de l’expéditeur et du destinataire, il s’agit juste de 2 strings classiques qui seront affectés par la suite dans le Send-MailMessage à la fin du script pour aérer un peu cette dernière commande.

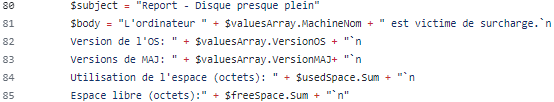


Figure - Sujet et corps de mail

Cette partie du code est la partie où le mail est rédigé $ body pourrait être optimisé avec une boucle foreach.



Figure - Mot de passe d'application Google utilisé

La variable du mot de passe est composée d’un mot de passe d’application généré aléatoirement par Google qui permet de se connecter à son compte Google depuis n’importe quelle application.



Figure - Sécurisation du mot de passe

Le mot de passe Google est converti en string sécurisé, un type créé exprès pour les mots de passe. Sans ça, Google considérera que la connexion n’est pas sécurisée même avec SSL et le mail ne s’enverra pas. Comme le mot de passe est en clair dans le code ce n’est pas sécurisé.



Figure - Création d'un objet de crédentité

Un objet spécial pour les logins est créé prenant en paramètre le nom d’utilisateur et le mot de passe d’un compte. Ici, le compte est l’adresse électronique, d’où le $sender qui est utilisé, suivi par le mot de passe d’application Google.



Figure - Envoi du mail via la commande Send-MailMessage

Envoi du mail en précisant le serveur smtp, son port, l’utilisation de SSL, l’adresse mail et son mot de passe, puis une précision de l’expéditeur ainsi que du destinataire. Le sujet du mail et le corps sont définis aussi.



Pour finir, le script supprime les fichiers OVA de plus de 3 Go de tout le disque.



Le script se termine sur un message annonçant que le script est terminé.

### Script 2

Le deuxième script se focalise sur une façon d’utiliser le premier. En effet, celui-ci copie le premier script depuis la machine locale et le colle dans un répertoire spécifique sur la ou les machines distantes (jusqu’à un maximum de 10). C’est par celui-ci que le projet a été commencé, jugé plus simple à implémenter et tester.



Figure - Instanciation du tableau

Un tableau est créé comme paramètre afin de pouvoir mettre plusieurs adresses IP.



Figure - Instanciation d'un deuxième tableau

Un autre tableau d’une taille de 10 éléments est créé afin de pouvoir stocker les paramètres entrés dans $remoteip.



Figure - placeholder - boucle foreach

La boucle foreach permet de naviguer dans le tableau $remoteip et de récupérer toutes les adresses IP pour les ajouter dans $computerList mentionné ci-dessus. Out-Null est utilisé afin de masquer l’output fait par la boucle à chaque passage dans celle-ci.



Figure - Conversion du mot de passe

C’est dans cette partie du script que le mot de passe est choisi et stocké pour l’utiliser par la suite. Cette instruction cause des problèmes de sécurité car avoir un mot de passe en clair permet à n’importe quelle personne ayant accès au script de voir le mot de passe. Pour palier à cette problématique, une solution serait de créer un fichier avec les mots de passe hachés à l’intérieur, puis de venir le récupérer avec une commande spécifique.



Figure - Récupération du nom d'utilisateur

Le nom d’utilisateur est récupéré ici. Comme le mot de passe et le nom d’utilisateur sont identiques pour les 2 machines distantes, ils sont renseignés en dur dans le script. Si ce n’était pas le cas, des paramètres auraient été utilisés. Cela crée un problème de sécurité similaire à celui du mot de mot de passe mentionné précédemment.



Figure - Implantation des chemins

Pour ce qui est des chemins, ils sont aussi en dur. Pour régler ce problème il faudra créer une valeur avec le nom du fichier et faire une concaténation avec $PWD pour rendre le script plus indépendant de la machine utilisée. $startpath est le chemin à optimiser, pour que le script respecte entièrement le cahier des charges. C’est lui qui va chercher le script 1 pour le copier plus tard. $destinationPath est le chemin de destination où sera stocké le script copié plus tôt. La lettre C:\ a été choisie car c’est la lettre par défaut d’un stockage. On pourrait améliorer ça en permettant à l’administrateur de choisir le chemin de destination via un paramètre.

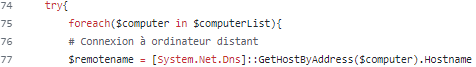


Figure - Début de la boucle

Cette boucle foreach commence par aller chercher le nom DNS des machines qui sera utile pour créer une session par la suite. C’est dans cette boucle qu’on utilise le tableau $computerList créé plus tôt.



Une session est créée ici pour chaque passage dans la boucle décidé par le nombre d’éléments dans le tableau $computerList cette session contient un id, un nom d’ordinateur, l’état de la connexion, le nom de la configuration qui s’apparente à comment la session a été configurée, et si la connexion et disponible ou non, comme le montre la capture d’écran ci-dessous.

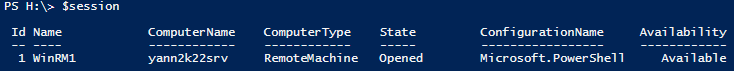


Figure - Informations d'une session ouverte



Cette instruction copie le fichier situé au chemin de départ et se connecte à l’ordinateur distant grâce au l’instruction -ToSession, puis colle le script dans le chemin de destination situé dans l’ordinateur distant.



Figure - Argument servant à lancer la tâche planifiée

Écriture du chemin complet du script dans une variable afin de pouvoir l’utiliser comme argument pour créer une tâche planifiée par la suite à l’intérieur de « Invoke-Command ». Celle-ci permet d’exécuter une commande sur la machine distante d’une manière plus efficace que Enter-PSSession, car Enter PSSession doit d’abord se connecter à la machine distante, tandis que invoke-command permet de rester sur la machine locale. Les guillemets simples permettent d’écrire les guillemets doubles et que ce ne soit pas considéré comme la façon d’écrire un string.



Figure - Création de la tâche planifiée

Une variable de tâche dénommée $task est créée en utilisant New-ScheduledTaskAction. On précise que la tâche utilisera PowerShell en le notant dans -Execute et on précise le script qui sera lancé grâce à son chemin.



Figure - Création de la répétivité de la tâche$

$taskInterval consiste à décider de la fréquence à laquelle le script sera exécuté. Il s’exécutera donc tous les jours à 16h00. La suite du script donne un nom à la tâche et le stocke dans une variable du nom de $taskName.

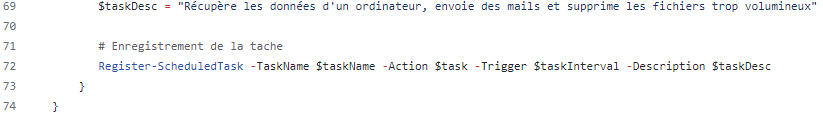


Figure - Description de la tâche et enregistrement de cette tâche

Le script donne une description à la tâche, puis l’enregistre définitivement ensuite. Les 2 accolades permettent de refermer invoke-command ainsi que le foreach.

Ce paragraphe décrit le fonctionnement de manière détaillée.

Autant que possible de manière graphique, imagée, tableaux, etc.

Tous les cas particuliers devraient y être spécifiés…

Il s’agit d’y présenter les fonctionnalités à développer :

Découpage en étapes, en modules, en fonctionnalités, etc.

Formulaires, interfaces graphiques, pages web, etc.

Schémas de navigation, schémas événementiels, structogramme, pseudocode, etc.

## Conception des tests

La stratégie de test employée est triviale ; Chaque ligne est testée indépendamment des autres pour avoir une suite d’instructions qui fonctionnent. Une fois ces instructions mises ensembles, elles sont retestées par groupe jusqu’à qu’elles fonctionnent elles-aussi. Cette opération sera effectuée tant que le script ne fonctionne pas entièrement.

## Planification détaillée

A ce stade, après l’analyse complète du projet, un planning détaillé et complet (avec tâches, sous-tâches, dépendances, durée, …) peut être finalisé.

Le planning détaillé doit s’inscrire dans le planning initial. Il faut que l’on puisse situer cette planification détaillée par rapport à la planification initiale.

# Réalisation

## Dossier de Réalisation

### Version des outils logiciels utilisés

Machine locale

|  |  |
| --- | --- |
| **Logiciel** | **Version** |
| PowerShell Desktop | 5.1.19041.4291 |

Yann2k22srv

|  |  |
| --- | --- |
| **Logiciel** | **Version** |
| PowerShell Desktop | 5.1.20348.558 |

Yann2k22srv2

|  |  |
| --- | --- |
| **Logiciel** | **Version** |
| PowerShell Desktop | 5.1.20348.558 |

### Configuration spéciale des serveurs

Tout ce qui n’est pas précisé est un paramètre par défaut. Les 3 machines ont la police d’exécution déterminée sur « Unrestricted »

Les 2 machines distantes sont configurées de la même manière, la seule différence est que yann2k22srv2 a une autre adresse ip (192.168.10.53). Toutes les captures d’écran proviennent de yann2k22srv. Les deux machines ont leur pare-feu désactivé.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure - Spécifications de Windows Server

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Figure - Configuration du pare-feu

Les pares-feux sont désactivés car ils bloquent l’utilisation de la commande

$remotename = [System.Net.Dns]::GetHostByAddress($remoteip).Hostname

Cela empêche donc le script 2 de résoudre le DNS et de récupérer le nom de la machine qui est utilisé par la suite pour créer une session dans laquelle il sera possible de coller le script 1. En bloquant le pare-feu, on s’affranchit des problèmes du genre. Malgré tout, cela cause d’énormes problèmes de sécurité. Le plus optimal serait de créer une règle qui autoriserait la commande de s’exécuter, et dans le même temps il faudrait désactiver toutes les règles qui coupent le script.

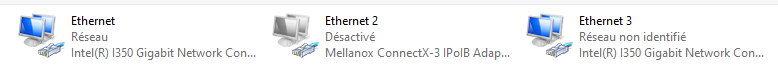


Figure - Configuration des cartes réseau

La partie réseau est décidée par le **réseau** de l’école et la partie hôte a été décidée arbitrairement. Comme c’est un masque /24, seulement la dernière partie est une partie hôte. Voici un tableau regroupant les adresses IP utilisées et à quelles machines elles correspondent :

|  |  |
| --- | --- |
| yann2k22srv | **192.168.10**.51 |
| INF-A13-M215 | **192.168.10**.52 |
| yann2k22srv2 | **192.168.10**.53 |

La carte réseau activée a 2 connexions : une pour que la machine soit connectée à internet et l’autre pour avoir un réseau LAN avec la machine physique.

Une image contenant texte, logiciel, Police, affichage

Description générée automatiquement

Figure - Protocole IPV6 désactivé

Le protocole IPV6 est désactivé afin d’éviter les problèmes d’incompatibilité qui pourraient survenir en utilisant winRM.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure - Activation du bureau à distance

Il a fallu activer le bureau à distance sur les serveurs physiques afin de pouvoir s’y connecter plus tard.

Une image contenant texte, Police, Bleu électrique, capture d’écran

Description générée automatiquement

Figure - Configuration de WinRM

Pour configurer celui-ci, il suffit d’ajouter les autres machines comme des hôtes de confiance (TrustedHosts avec PowerShell. Si le pare-feu n’est pas désactivé, cela ne fonctionnera pas. La valeur est une étoile pour autoriser n’importe quelle machine.

La prochaine étape est de créer un mot de passe d’application pour utiliser un compte Google Pour créer un mot de passe d’application, il faut accéder à l’A2F de Google dans les réglages du compte, puis sélectionner « mot de passes d’application ». Il suffit ensuite d’entrer un nom et d’appuyer sur « créer »

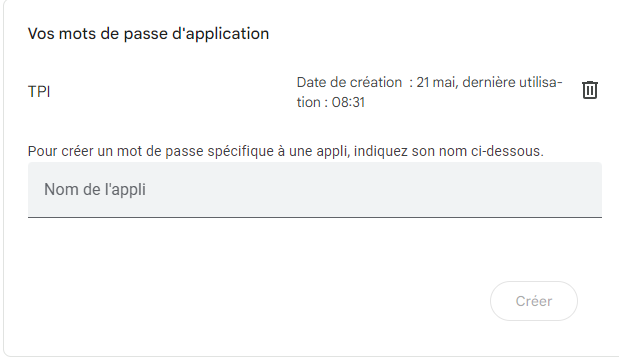


Figure - Page d'accueil de la page de création de mots de passe d'application Google

Google générera un mot de passe composé de 19 caractères en comptant les espaces comme celui ci-dessous.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure - Mot de passe généré

C’est un mot de passe comme celui-ci qu’il faut entrer dans le script pour que celui-ci puisse se connecter à Google.

### Réalisation du script de setup

Comme dit plus haut dans la partie analyse, le script 2 a été réalisé avant le premier, ce qui a causé un problème au moment de la réalisation du script 1 mais ce sera expliqué plus tard dans ce rapport. Le script a d’abord été fait avec le minimum requis. C’est-à-dire la copie d’un item dans une session, la création d’une tâche et des chemins en dur. J’ai eu des problèmes en réalisant la partie de la copie, car une erreur mentionnant un chemin vide était affichée. J’ai d’abord essayé de créer une liste d’arguments dans le invoke-cammand, mais ça ne fonctionnait pas non plus. Il s’est avéré que le problème venait du fait que la commande qui posait problème devait être en dehors du bloc d’invoke-cammand, et que je devais utiliser -ToSession pour faire la copie du fichier plutôt qu’essayer de me connecter à la session depuis cette-même session, vu que c’est comme ça qu’Invoke-Command fonctionne.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure - Le message d'erreur qui apparaissait avant de modifier copy-item

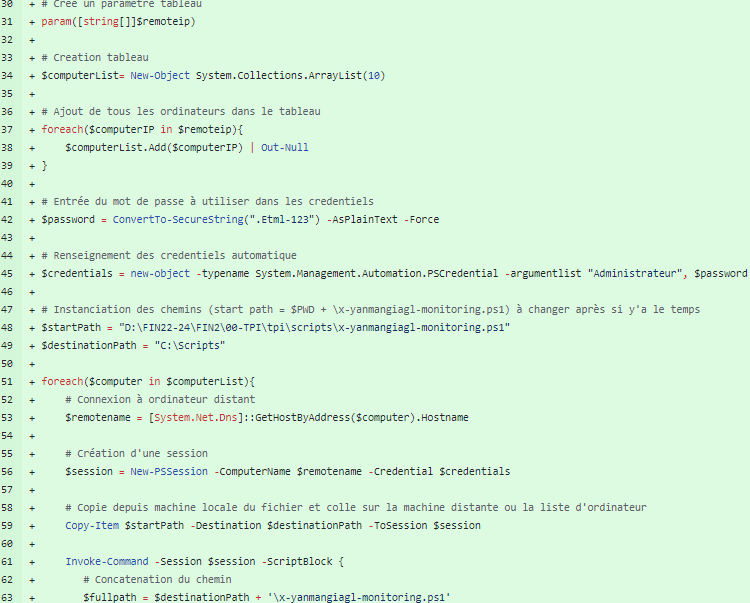


Figure - Première partie du code source original

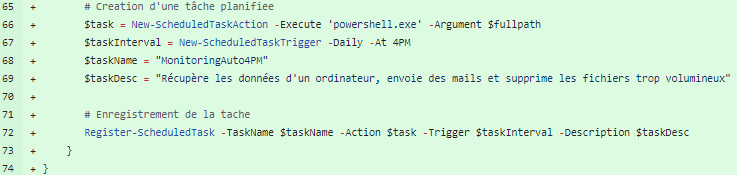


Figure - Seconde partie du code source original

On peut voir dans cette première version plusieurs problèmes ; les chemins en dur ne respectent pas le cahier des charges et il n’y a aucune gestion d’erreur. Les exemples aussi sont faux, c’est la première chose qui est modifiée.

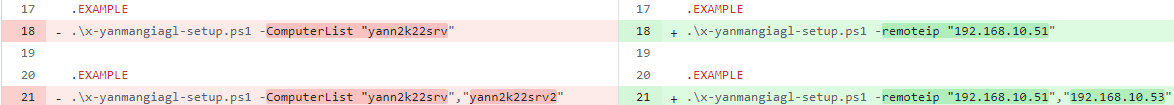


Figure - Modification des exemples

Cependant, un problème inattendu s’est déclaré qui a forcé une nouvelle version du script. En effet, le script 1 était bien ajouté en argument de la tâche planifiée, mais elle n’avait pas le bon format. Celui-ci devait non seulement contenir le chemin entier, mais en plus il devait contenir un argument nommé -File pour que le script s’exécute bien correctement.



Figure - Modification de l'argument du script 2

J’ai su que je devais faire ça en créant une tâche planifiée manuellement sur yann2k22srv, et dans les options du déclencheur de tâche, voilà ce qui était montré :

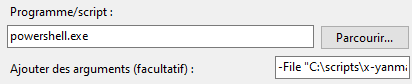


Figure - Résultat voulu de $fullpath

Une fois cette erreur corrigée, j’en ai profité pour mettre en place la gestion d’erreurs et les limites du script 2.

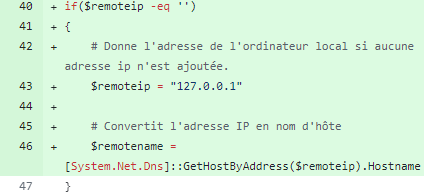


Figure - Ajout de gestion d'erreurs pour la conversion DNS

Si aucune adresse IP n’est spécifiée, le diagnostic est fait sur la machine locale. Attention cependant, cela ne fonctionnera pas sur la machine locale de l’école INF-A13-M215 car elle reste gérée par l’école qui bloque la requête du script.

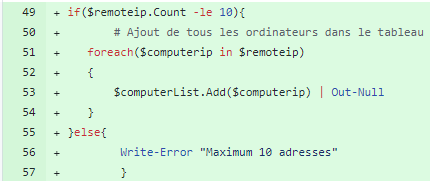


Figure - Ajout d'une limite de 10 machines

Comme il est demandé dans le cahier des charges que le nombre d’adresses maximum soit de 10 machines, le script limite le tableau d’adresses IP à 10 éléments maximum.

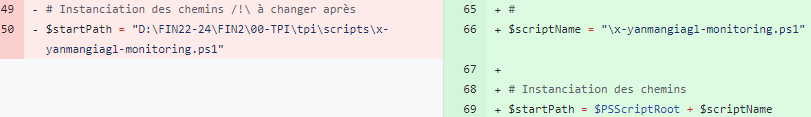


Figure - Modification d'un chemin en dur en chemin automatique

Ici, le chemin a été changé pour être automatique avec la variable $PSScriptRoot qui récupère le chemin duquel le script est exécuté, ce qui permet de se débarrasser d’un chemin en dur.



Figure - Ajout d'une gestion d'erreurs pendant la connexion

Une gestion d’erreur est effectuée avec un try catch pour sécuriser tout le code, même si cela ralentira un peu l’exécution du code.

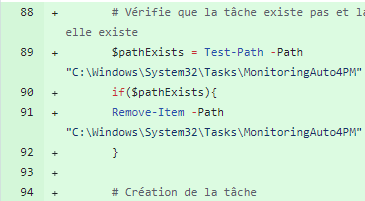


Figure - Vérification de si la tâche existe déjà, et la supprime si c'est le cas

Avant de créer la tâche, j’ai incorporé un test permettant de checker si la tâche existe déjà en me basant sur le chemin par défaut de Windows.

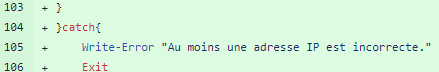


Figure – Bloc de code si une adresse IP est incorrecte

Si le script a une erreur, par exemple une adresse IP non valide, et bien un message d’erreur a été rajouté. Je pourrais améliorer ça en faisant en sorte que le script mette de côté cette adresse IP et fasse quand même les adresses IP valides plutôt que juste quitter le programme. Le script est donc fonctionnel comme démontré ci-dessous :

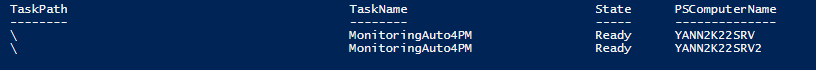


Figure - Sortie du script une fois les tâches créées

### Réalisation du script de maintenance

Le script de maintenance est le script qui se charge de vérifier le stockage et de récupérer les variables qui seront envoyées par mail si le stockage atteint 80%. Il y’a eu pas mal de problèmes au début pour faire fonctionner l’envoi de mail qui m’a bloqué pendant quasiment une journée complète. En effet, j’avais réalisé en amont du projet un script qui envoyait un mail sécurisé avec MimeKit et MailKit en utilisant Nuget, qui est un gestionnaire de package pour .NET [[1]](#footnote-1). Nuget a donc été installé avec MailKit et MimeKit.

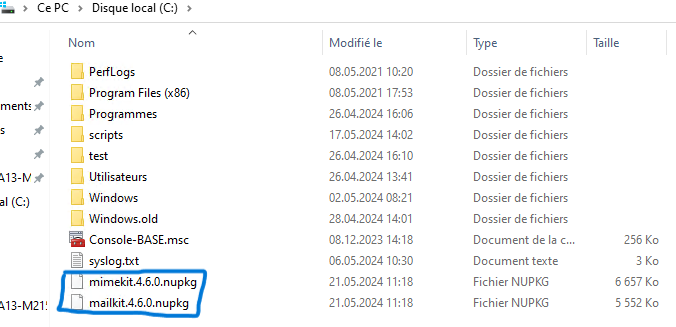


Figure - Installation de MimeKit et MailKit

Cependant, une erreur proclamait qu’il manquait des dépendances et le script s’arrêtait. Il a donc été décidé que Send-MailMessage serait utilisé, mais ce n’était pas plus facile pour autant.

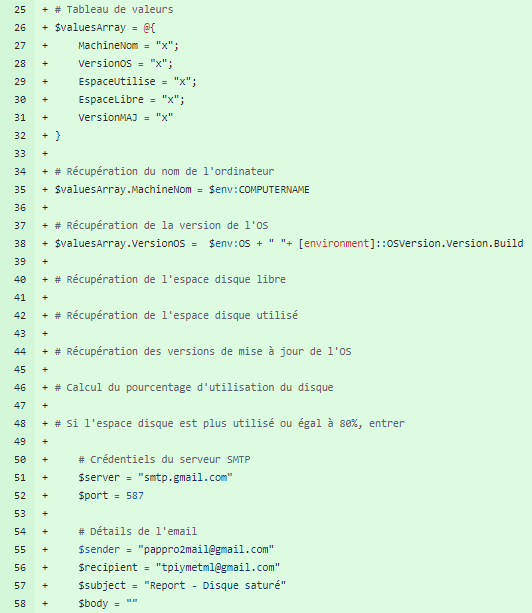


Figure - Première partie du script 1

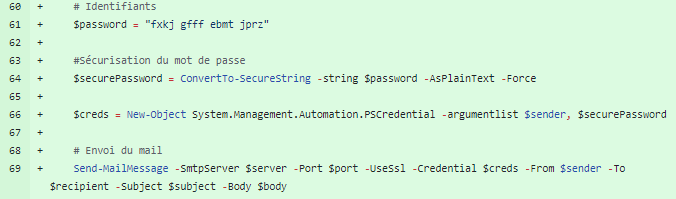


Figure - Seconde partie du script 1

Dans cette première version du script, seules les variables d’environnement et les valeurs nécessaires à l’envoi du mail sont réalisées, car c’est ce que j’ai eu le temps de faire à cause du blocage créé à cause de ma première idée avec NuGet, puis les problèmes qui ont suivi avec Send-MailMessage.



Figure - Capture d'écran du message d’erreur dû à Google

Ce message d’erreur a été un grand ralentissement dans ma progression, car j’avais renseigné le bon port, le bon mot de passe, le bon nom d’utilisateur, et spécifié le SSL, désactivé le pare-feu du serveur et la machine était connectée à Internet. Ce qui faisait bloquer le mail était qu’il fallait en faire un secure-string. J’avais pensé que l’instruction créait un objet identitaire le faisait nativement, mais non. Google ne considérait la connexion sans sécurité, même si elle était en SSL. J’ai donc dû continuer le script le jour suivant.



Figure - Enlèvement de la lettre du disque

J’ai commencé par enlever la lettre du disque car il n’y en a pas besoin dans l’immédiat, puis j’ai ajouté les valeurs qui manquaient pour l’espace libre et utilisé.

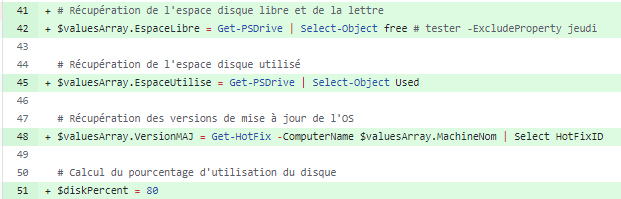


Figure - Ajout des valeurs dans le script

Je ne l’ai pas implémenté correctement, il fallait une variable ou préciser, par exemple $valuesArray.EspaceUtilise.Used, mais ça faisait répétitif et ce n’était pas très lisible. J’ai corrigé cela plus tard.



Figure - Création de la condition de 80%

Un début de code pour la suppression des fichiers volumineux est créé. C’est dans ce bloc que sera rajouté plus tard toutes les commandes de suppression.



Figure - Création du corps du mail

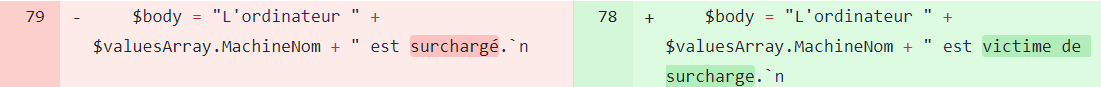
Le corps du mail est rempli pour ne plus être vide, et on y met déjà les valeurs qu’on veut recevoir dans le mail.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure - Mails de tests reçus

Dans ce mail cependant, on peut remarquer un problème d’encodage. Pour contourner le problème, j’ai décidé de reformuler mon texte pour ne pas avoir à me soucier de ce problème handicapant, comme dans cet exemple.



Les autres changements sont plus importants ; L’ajout des versions de mise à jour et la suppression des fichiers OVA de plus de 3 Gb [[2]](#footnote-2).





L’espace libre et l’utilisation de l’espace sont bien notés d’une manière qui fonctionne, et plus lisible qu’avant.



La commande cherche dans tous les dossiers que contient C : ainsi que C : lui-même tous les fichiers dont l’extension est .ova, et qui pèsent plus de 3Gb, puis les supprime. Le -Force est là pour effacer les .ova qui pourraient être cachés.

Cette partie permet de reproduire ou reprendre le projet par un tiers.

Pour chaque étape, il faut décrire sa mise en œuvre. Typiquement :

Versions des outils logiciels utilisés (OS, applications, pilotes, librairies, etc.)

Configurations spéciales des outils (Equipements, PC, machines, outillage, etc.)

Code source commenté des éléments logiciels développés.

Modèle physique d’une base de données.

Arborescences des documents produits.

Il faut décrire le parcours de réalisation et justifier les choix.

## Modifications

Historique des modifications demandées (ou nécessaires) aux spécifications détaillées.

Date, raison, description, etc.

# Tests

## Dossier des tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Numéro | Description | Procédure | Résultat attendu | Résultat obtenu |
| 1 | Vérification synchronisation GitHub – GitHub desktop | Faire un commit GitHub en utilisant GitHub Desktop afin de voir si le commit passe bien sur GitHub | OK | OK |
| 2 | Lancement d’un script depuis son chemin à l’intérieur d’un autre script | Entrer la commande ./x-pappromail.ps1 pour voir si le script s’exécute | KO | OK |
| 3 | Affichage de l’aide (script 2) | Dans l’en-tête des scripts, renseigner .SYNOPSIS, .DESCRIPTION, .EXAMPLE et .PARAMETER, puis faire les commandes  Get-Help x-yanmangiagl-setup.ps1 et  Get-Help yanmangiagl-setup.ps1 -Examples | L’aide s’affiche sans problème | L’aide s’affiche sans problème |
| 4 | Déplacement des variables (script 2) | Faire bouger les variables $startPath et $destinationPath dans le scriptblock d’invoke command pour ne plus avoir de x dite $null alors qu’elle ne l’est pas | Les variables sont instanciées et permettent au script de passer à la suite | Les 2 variables ont l’air de fonctionner maintenant mais $session est $null |
| 5 | Supprimer d’ invokecommand copy-item |  | OK | OK |
| 6 | Tenter 192.168.10.53 sans l’autre comme paramètre du script 2 (.51 fonctionne) | Restart-Service winrm | Marche pour toujours | Fonctionne seulement une fois après un redémarrage du serveur si la commande est faite assez vite |
| 8 | Désactiver pare-feu serveur 2 | La premier serveur a un pare-feu éteint et pas le deuxième, tester de le désactiver pour voir si ça fonctionne | Marche pour toujours | Marche pour toujours |
| 9 | Envoi de mail avec Send-MailMessage | Lancer la commande permettant d’envoyer un mail | La commande peut s’exécuter sans problème | La commande ne s’exécute pas et retourne une erreur |
| 10 | Encoder avec Powershell | Réaliser ce qui est fait sur [cette page](https://stackoverflow.com/questions/23256911/encode-a-string-in-utf-8) | Affichage de éà dans le mail | Affichage de ?? dans le mail |
| 11 | Test du script 1 fini | Mise en commentaire de la commande mettant le vrai pourcentage et lancement du script 1 depuis la machine locale | Le script fonctionne comme prévu et le mail est envoyé | Le script fonctionne comme prévu et le mail est envoyé |
| 12 | Exécution du script depuis la tâche planifiée | Même test mais sur yann2k22srv | Le script fonctionne comme prévu et le mail est envoyé | Le mail n’arrive pas > Problème vient sûrement de la création de la tâche dans script 2 |
| 13 | Création d’une tâche manuelle | Création d’une tâche qui exécute le script manuellement pour comprendre d’où vient le problème en comparant les 2 tâches après | Dans les arguments on voit c:\scripts… | Dans les arguments on voit -File c:\scripts… |
| 14 | Connecter serveur à internet | Réactiver une des cartes réseau | OK | OK |
| 15 | Lancer un mail depuis les 2 machines en même temps | Lancer le script 2 avec les 2 machines en paramètres avec le script de debug pour voir comment se lance les mails | Chaque machine fait son propre mail | Chaque machine fait son propre mail |
| 16 | Test taille réelle | OVA de 3 Go, OVA de 2 Go, deux serveurs prêts à accueillir les scripts | OVA de 3go suppr, ova de 2 go gardé, mail envoyé | -Path est null pour le truc que j’ai réglé dans les 5 min, tout le reste a fonctionné |

On dresse le bilan des tests effectués (qui, quand, avec quelles données…) sous forme de procédure. Lorsque cela est possible, fournir un tableau des tests effectués avec les résultats obtenus et les actions à entreprendre en conséquence (et une estimation de leur durée).

Si des tests prévus dans la stratégie n'ont pas pu être effectués :

raison, décisions, etc.

Liste des bugs répertoriés avec la date de découverte et leur état:

Corrigé, date de correction, corrigé par, etc.

# Conclusion

## Bilan des fonctionnalités demandées

Il s’agit de reprendre point par point les fonctionnalités décrites dans les spécifications de départ et de définir si elles sont atteintes ou pas, et pourquoi.

Si ce n’est pas le cas, estimer en « % » ou en « temps supplémentaire » le travail qu’il reste à accomplir pour terminer le tout.

## Bilan de la planification

Distinguer et expliquer les tâches qui ont généré des retards ou de l'avance dans la gestion du projet. Indiquer les différence entre les planifications initiales et détaillées avec le journal de travail.

## Bilan personnel

Si c’était à refaire:

Qu’est-ce qu’il faudrait garder ? Les plus et les moins ?

Qu’est-ce qu’il faudrait gérer, réaliser ou traiter différemment ?

Qu’est que ce projet m’a appris ?

Suite à donner, améliorations souhaitables, …

Remerciements, signature, etc.

# Divers

## Journal de travail

Date, activité (description qui permet de reproduire le cheminement du projet), durée, liens et références sur des documents externes. Lorsqu’une activité de recherches a été entreprise, il convient d’énumérer ce qui a été trouvé, avec les références.

## Bibliographie

Références des livres, revues et publications utilisés durant le projet.

## Webographie

Références des sites Internet consultés durant le projet.

# Annexes

Listing du code source (partiel ou, plus rarement complet)

Guide(s) d’utilisation et/ou guide de l’administrateur

Etat ou « dump » de la configuration des équipements (routeur, switch, robot, etc.).

Extraits de catalogue, documentation de fabricant, etc.

1. Citation venant de Microsoft Learn [↑](#footnote-ref-1)
2. Instruction de supression générée par Bing Copilot et modifiée pour convenir au contexte [↑](#footnote-ref-2)