



30/05/2022

Projet PowerShell

Documentation technique

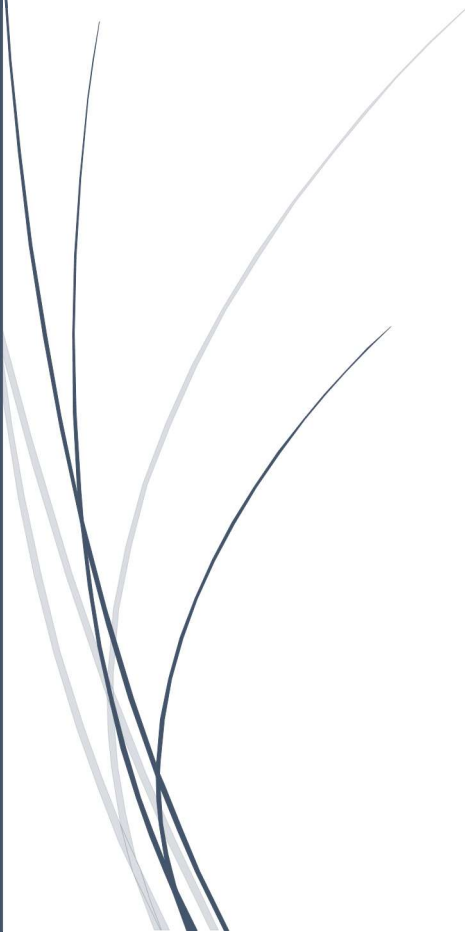
- 
- Jonathan CHAUMONT
 - Nathan GAY
 - Gaultier GAUCHON
 - Benjamin BRUNCO
 - Victor BOHN

Table des matières

I) Déploiement ARM	2
1) Script de déploiement :	2
2) Template Azure Resources Manager :	5
A. Les ressources :	5
B. Les paramètres :	6
C. Les variables :	7
D. Ressource virtualMachines/extension type custom script extension :	8
II) Supervision	9
1) Les réglages	9
2) Les fonctions	11
III) Installation des services :	16
1) Installation des services via poste local	16
A. Vérification de l'état du service WinRM	16
B. Installation & Configuration du service Active Directory	17
C. Installation & Configuration du service DHCP	18
D. Menu du script	19
2) Installation des services AD, DNS, DHCP via template ARM	20
A. Installation du service ActiveDirectory	20
B. Paramétrage de la tâche planifiée pour provisioning AD	21
C. Installation du service DHCP	21
3) Installation du serveur WEB + Supervision	22
A) Installation du service IIS	22
B) Paramétrage de la tâche planifiée (Supervision)	23
C) Enregistrement dans le domaine	23
IV) Création UO et utilisateurs	24
1. Création des unités d'organisations	24
2. Création des utilisateurs via import csv	25

I) Déploiement ARM

1) Script de déploiement :

```
1 function PPS2022VerificationModuleAz {  
2     #Définir les règles d'exécution des modules#  
3     Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser  
4  
5     #Installer le module Azure Powershell#  
6     Install-Module -Name Az -AllowClobber -Scope AllUsers -Force  
7 }  
8  
9
```

La fonction PPS2022VerificationModuleAz, permet de vérifier si les modules signés peuvent être installé pour l'utilisateur actuel grâce à la commande « Set-ExecutionPolicy », ainsi que les paramètres « -ExecutionPolicy » et « -Scope ».

Également cette fonction va installer le module Az, qui permet d'utiliser les commandes PowerShell relative à Azure.

```
10 function PPS2022CreationRG {  
11     #Demande des identifiants  
12     $credential = Get-Credential -Message "Veuillez entrer les identifiants de votre compte Azure"  
13  
14     # Connexion à la console Azure  
15     Login-AzAccount -Credential $credential  
16  
17     #Créer un groupe de ressources#  
18     New-AzResourceGroup -Name "PPS2022-RG" -Location 'Switzerland North'  
19 }  
20  
21  
22
```

Pour la fonction PPS2022CreationRG, une demande d'identifiants est faite via la commande « Get-Credential » et les informations sont stocké dans la variable \$credential en tant que « System.Security.SecureString ».

Par la suite, la commande PowerShell « Login-AzAccount », fournie par le module Az, permet la connexion au compte Azure grâce au paramètre « -Credential » et la variable \$credential avec les informations demandées précédemment.

```
23 function PPS2022DeploiementARM {  
24     #Utilisation des templates ARM#  
25     $TemplateARM = ".\PPS2022-Template-ARM-DC+IIS.json"  
26     New-AzResourceGroupDeployment -Name "PPS2022-ARM-DC" -ResourceGroupName "PPS2022-RG" -TemplateFile $TemplateARM  
27 }  
28  
29
```

La fonction PPS2022DeploiementARM est la fonction qui va permettre la création du groupe de ressources ainsi que le déploiement des ressources en faisant appel à une template « Azure Ressources Manager ».

La variable \$TemplateARM va permettre d'indiquer le chemin d'accès à la template, qui est au même emplacement que ce script lorsque le dépôt github a été cloné.

Ensuite encore une fois avec une commande, « New-AzResourceGroupDeployment », fourni par le module Az, le groupe de ressource « PPS2022-RG » est créé. Puis la template ARM est appelé via le paramètre « -TemplateFile ».

```
30 function PPS2022SuppressionRessources {  
31     #Suppression du groupe de ressources et donc de chaque ressources crée#  
32     Remove-AzResourceGroup -Name "PPS2022-RG"  
33     Remove-AzResourceGroup -Name "NetworkWatcherRG"  
34 }  
35  
36
```

La fonction PPS2022SuppressionRessources permet de supprimer toutes les ressources créer par la template ARM en supprimant le groupe de ressources, grâce à la commande « Remove-AzResourceGroup » et la paramètre « -Name ».

```

37 function PPS2022ChoixMenu {
38     $continue = $true
39     while ($continue){
40         write-host "-----SCRIPT de Déploiement-----"
41         write-host "1. Verifier le module Az sur le poste local"
42         write-host "2. Connexion à Azure et création RG"
43         write-host "3. Déploiement ARM"
44         write-host "4. Supression des ressources"
45         write-host "5. Exit"
46         write-host "-----"
47         $choix = read-host "Faire un choix"
48         switch ($choix){
49             1{PPS2022VerificationModuleAz}
50             2{PPS2022CreationRG}
51             3{PPS2022DeploiementARM}
52             4{PPS2022SuppressionRessources}
53             5{$continue = $false}
54
55             default {Write-Host "Choix invalide"-ForegroundColor Red}
56         }
57     }
58 }
59 PPS2022ChoixMenu

```

Enfin, la fonction PPS2022ChoixMenu est la fonction qui permet de choisir les actions désirées.

Le texte expliquant les choix possibles est affiché grâce plusieurs commande Write-Host à la suite.

Le choix de l'utilisateur est récupéré avec la commande Read-Host et stocké dans la variable \$choix.

Un switch permet d'appeler les fonctions du script vu précédemment en fonction du choix de l'utilisateur.

2) Template Azure Resources Manager :

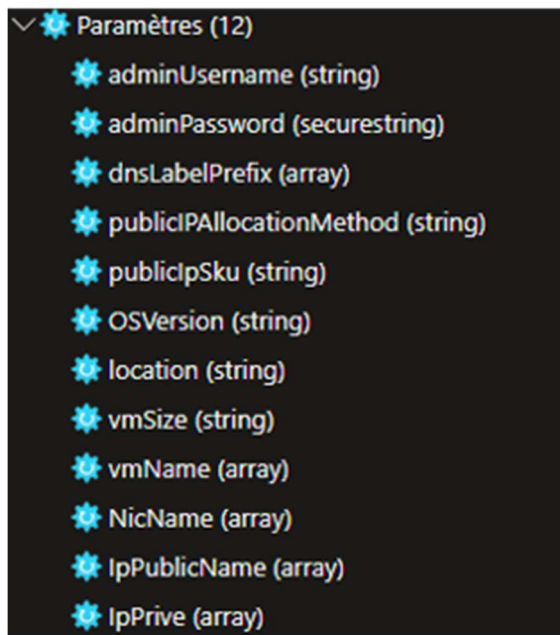
A. Les ressources :



Deux machines virtuelles sont déployées, un disque virtuel par machine, une carte réseau par machine, une IP publique par machine, un seul réseau virtuel, un storage account pour stocker les scripts de post déploiement, et un network security group.

Le Network security group est paramétré pour autoriser les flux RDP (3389), HTTP (80), PSSesison (5986)

B. Les paramètres :



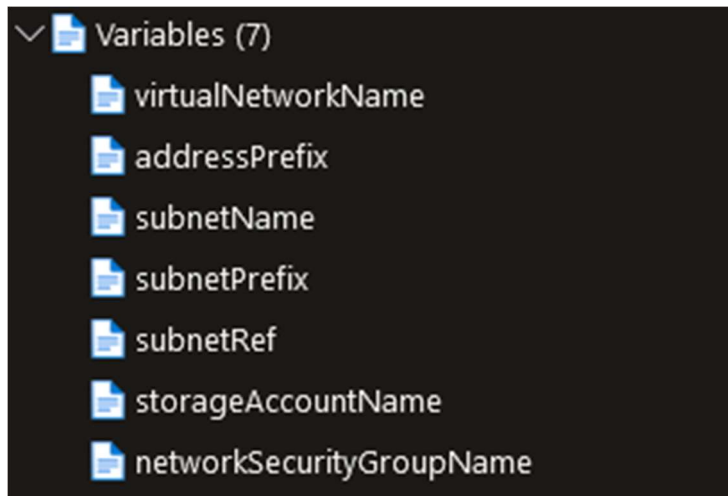
Le paramètre adminpassword n'est pas renseigné dans la template et son type est securestring, cela permet une demande du mot de passe au lancement du script de déploiement, et ce dernier ne sera pas affiché et stocké en clair.

Les paramètres ayant pour type array, permettent de stocker plusieurs informations qui sont utilisées par les boucles CopyIndex() ; Copy() et Count() à l'intérieur des ressources déclarées dans la template

OSVersion permet de renseigner le système d'exploitation voulu pour la VM, ici Windows Server 2019 Datacenter Core gen 2

vmSize permet d'indiquer le type d'instance Azure voulu, ici Standard_D2s_v3, 2 cpu 8 Go de RAM

C. Les variables :



virtualNetworkName : PPS2022-VNET

addressPrefix : 192.168.1.0/16

subnetName : PPS2022-Subnet

subnetPrefix : 192.168.1.0/24

storageAccountName : pps2022storageaccount

networkSecurityGroupName : PPS2022-NSG

D. Ressource virtualMachines/extension type custom script extension :

Cette ressource est utilisée pour déployer automatiquement les scripts de post déploiement.

La ligne `commandToExecute` permet d'indiquer le script voulu et de l'exécuter avec PowerShell.

Elle permet également de passer des variables ou paramètres présent dans la template ARM et de les récupérer en paramètres dans la VM, et les utiliser dans un script PowerShell

La ligne `fileUri` permet de renseigner le lien pour accéder à ce script, ici sur github.

Deux ressources de ce type sont utilisées. Une pour déployer les services AD, DNS, DHCP sur le contrôleur de domaine en utilisant le script `PPS2022-InstallServices-AD-DNS-DHCP.ps1`

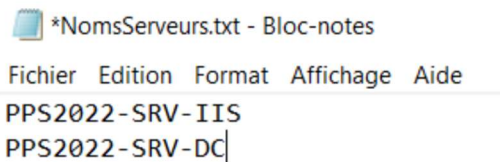
Une pour déployer le services IIS sur la machine utilisée en tant que serveur web, en utilisant le script `PPS2022-InstallService-IIS.ps1`.

```
"type": "Microsoft.Compute/virtualMachines/extensions",
"apiVersion": "2018-06-01",
"name": "PPS2022-SRV-IIS/config-app",
"location": "[resourceGroup().location]",
"tags": {
  "displayName": "PPS2022-InstallServices"
},
"properties": {
  "publisher": "Microsoft.Compute",
  "type": "CustomScriptExtension",
  "typeHandlerVersion": "1.10",
  "autoUpgradeMinorVersion": true,
  "settings": {
    "timestamp": 123456789
  },
  "protectedSettings": {
    "storageAccountName": "[variables('storageAccountName')]",
    "storageAccountKey": "[listKeys(variables('storageAccountName'), '2019-04-01').keys[0].value]",
    "commandToExecute": "[concat('powershell -ExecutionPolicy Unrestricted -File PPS2022-InstallService-IIS.ps1', ' -DomainPassword ', parameters('adminPassword'))]",
    "fileUri": [
      "https://raw.githubusercontent.com/Jonathan28260/ProjetAnnuel_Powershell/main/PPS2022-InstallService-IIS.ps1"
    ]
  }
}
```

II) Supervision

1) Les réglages

On commence avec les noms de serveurs qu'on souhaite superviser. Ils sont notés à la ligne dans un document texte appelé NomsServeur.txt



Ensuite grâce à la commande Get-Content on prend le contenu de ce fichier texte et on le met dans la variable \$NomsServeurs

```
#Le fichier d'entrée avec les noms des windows core
$NomsServeurs = Get-Content 'C:\Scripts\NomsServeurs.txt'
```

On précise la variable \$HTMLSupervision avec le fichier HTML de sortie qui sera notre page IIS

```
#Le fichier HTML de sortie qui sera la page web de supervision
$HTMLSupervision = 'C:\inetpub\wwwroot\Index.html'
```

Sur cette partie-là on va faire les réglages du HTML, comme la partie auto refresh de 13 secondes. On va importer le module w3-css qui nous permettra de mettre de la couleur et de créer l'entête de HTML, avec le titre, la date et l'heure, les colonnes etc. Ensuite on exporte ça dans le HTML.

```
#Ajout du module w3 css et réglages de la barre superieur du tableau HTML
'<meta http-equiv="refresh" content="13"
<Head>
<title>Supervision</title>
</Head>
<body class="w3-tiny">
<div class="w3-responsive w3-tiny">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<h4 class="w3-indigo w3-center">Resultats du '+ (Get-date -Format "dd/MM/yyyy HH:mm") +'</h4>
<table class="w3-table w3-centered w3-bordered w3-card-4 w3-centered w3-hoverable">
<tr class="w3-indigo">
  <th class="w3-border w3-tiny">Nom du serveur</th>
  <th class="w3-border w3-tiny">Adresse(s) IP</th>
  <th class="w3-border w3-tiny">Version de Windows</th>
  <th class="w3-border w3-tiny">Numero de build</th>
  <th class="w3-border w3-tiny">Utilisation CPU</th>
  <th class="w3-border w3-tiny">Utilisation RAM</th>
  <th class="w3-border w3-tiny">Statut</th>
' | Out-File $HTMLSupervision utf8
#fin des réglages
```

2) Les fonctions

La fonction Supervision adresse IP nous permet, via une résolution DNS (commande Resolve-DNSName) de connaître l'IP du serveur en ne connaissant que son nom. On ajoute cette valeur à la variable \$IP. Enfin on ajoute cette valeur ainsi que celle de couleur de statut (explication plus loin) dans mon fichier HTML de sortie.

```
function PPS2022SupervisionAdresseIP #Le but de cette fonction est de connaître l'IP en connaissant le nom DNS
{
    $IP = (Resolve-DNSName $Nom -Type A).IPAddress -join " - " #Resolution DNS pour avoir IP, separation de multiresultats par un tiret
    "<td class=" + $ColorStatut + ">" + $IP + "</td>" | Out-File $HTMLSupervision -Append utf8 #J'ajoute ma variable IP au HTML
}
```

La fonction Supervision OS information nous permet de connaître le système installé sur le serveur via la commande **Get-WmiObject Win32_OperatingSystem).Caption** dont on exporte la valeur dans une variable \$OSVersion et de connaître le build installé via la commande **Get-WmiObject Win32_OperatingSystem).BuildNumber** donc on exporte la valeur dans une variable \$OSBuild. Ensuite on exporte ces valeurs ainsi que la valeur de couleur de statut au fichier HTML de sortie.

```
function PPS2022SupervisionOSInformation
{
    $OSVersion = (Get-WmiObject Win32_OperatingSystem -ComputerName $Nom).Caption #Commande pour connaître la version de l'OS
    $OSBuild = (Get-WmiObject Win32_OperatingSystem -ComputerName $Nom).BuildNumber #Commande pour connaître le numero de build de l'OS
    "<td class=" + $ColorStatut + ">" + $OSVersion + "</td>" | Out-File $HTMLSupervision -Append utf8 #J'ajoute la variable OSVersion au HTML
    "<td class=" + $ColorStatut + ">" + $OSBuild + "</td>" | Out-File $HTMLSupervision -Append utf8 #J'ajoute la variable OSBuild au HTML
}
```

La fonction Supervision CPU est là pour qu'on connaisse le pourcentage d'utilisation du processeur et aussi en déduire un code couleur suivant le résultat.

La variable \$CPUUsage est scoper « script » pour que l'on puisse l'utiliser en dehors de la fonction (pour la fonction color statut on verra plus bas)

Ensuite grâce à la commande :

```
(Get-WmiObject -ComputerName $Nom -Class win32_processor -ErrorAction Stop | Measure-Object  
-Property LoadPercentage -Average | Select-Object Average).Average
```

Nous récupérons dans la variable \$CPUUsage le pourcentage d'utilisation du processeur.

Ensuite avec une boucle IF on en déduit la variable \$ColorCPU en fonction de l'utilisation, vert si moins de 70% d'utilisation, jaune si entre 70% et 85% et rouge au-delà.

```
function PPS2022SupervisionCPUUsage  
{  
    $script:CPUUsage = (Get-WmiObject -ComputerName $Nom -Class win32_processor -ErrorAction Stop | Measure-Object -Property LoadPercentage -Average | Select-Object Average).Average  
    if ($CPUUsage -le 70) # Si l'usage CPU est en dessous de 70%  
    {  
        $script:ColorCPU = 'w3-green' #La variable ColorCPU prend la valeur vert  
    }  
  
    elseif(($CPUUsage -gt 70) -and ($CPUUsage -le 85)) #Sinon si l'usage CPU est entre de 70% et 85%  
    {  
        $script:ColorCPU = 'w3-yellow' #La variable ColorCPU prend la valeur jaune  
    }  
  
    else #Sinon  
    {  
        $script:ColorCPU = 'w3-red' #La variable ColorCPU prend la valeur rouge  
    }  
}
```

La fonction Supervision RAM Usage se base en partie sur le même concept que la CPU Usage.
Les deux principales différences c'est qu'en plus de la commande pour récupérer les informations techniques :

```
Get-WmiObject -Class WIN32_OperatingSystem -ComputerName $Nom
```

Il y a une commande pour en faire une valeur exploitable sur 100, un pourcentage d'utilisation. Ici on utilise également [math]::round pour avoir une valeur arrondie au decimal :

$$[math]::round(((\$RAMInfos.TotalVisibleMemorySize - \$RAMInfos.FreePhysicalMemory)*100)/\$RAMInfos.TotalVisibleMemorySize)$$

On met ensuite ces informations dans une variable \$RAMUsage

Enfin de la même manière que la CPU Usage, avec une boucle IF je déduit la variable \$ColorRAM en fonction de l'utilisation, vert si moins de 70% d'utilisation, jaune si entre 70% et 85% et rouge au-delà.

```
function PPS2022SupervisionRAMUsage
{
    $RAMInfos = Get-WmiObject -Class WIN32_OperatingSystem -ComputerName $Nom #Je recupere les informations hardware
    $script:RAMUsage = [math]::round((($RAMInfos.TotalVisibleMemorySize - $RAMInfos.FreePhysicalMemory)*100)/ $RAMInfos.TotalVisibleMemorySize)

    if ($RAMUsage -le 70) # Si l'usage RAM est en dessous de 70%
    {
        $script:ColorRAM= 'w3-green' #La variable ColorRAM prend la valeur vert - Les variable Color sont de portées globale pour pouvoir fonction
    }

    elseif(($RAMUsage -gt 70) -and ($RAMUsage -le 85)) #Sinon si l'usage RAM est entre de 70% et 85%
    {
        $script:ColorRAM = 'w3-yellow' #La variable ColorRAM prend la valeur jaune
    }

    else #Sinon
    {
        $script:ColorRAM = 'w3-red' #La variable ColorRAM prend la valeur rouge
    }
}
```

La variable Supervision Statut, elle, nous permet d'avoir une idée globale de l'état de la machine.

Si le CPU et la RAM sont en vert, alors c'est vert et la variable statut prend la valeur Good.

Si les deux sont en rouge, alors c'est rouge et la variable statut prend comme valeur KO.

Sinon la couleur est jaune et la variable statut prend la valeur Warning.

Ensuite on exporte le résultat dans le html pour la colonne statut.

Aussi la couleur de la variable \$ColorStatut est mise sur toute les cases de la ligne comme vu dans les fonctions précédentes pour avoir une vue rapide sur l'état général des serveurs.

```
function PPS2022Supervisionstatut
{
    if(($ColorCPU -eq 'w3-green') -and ($ColorRAM -eq 'w3-green')) #Si la RAM et le CPU sont au vert
    {
        $script:Statut = 'Good' #La variable statut vaut "Good"
        $script:ColorStatut = 'w3-pale-green' #La variable ColorStatut sera verte
    }

    elseif(($ColorCPU -eq 'w3-red') -and ($ColorRAM -eq 'w3-red')) #Sinon si la RAM et le CPU sont au rouge
    {
        $script:Statut = 'KO' #La variable statut vaut "KO"
        $script:ColorStatut = 'w3-pale-red' #La variable ColorStatut sera rouge
    }

    else #Sinon
    {
        $script:Statut = 'Warning' #La variable statut vaut "Warning"
        $script:ColorStatut = 'w3-pale-yellow' #La variable ColorStatut sera jaune
    }
}
```

Enfin la fonction Supervision qui fait office de fonction main. Avec une boucle foreach de chaque serveur dans le fichier texte elle va exécuter les fonctions une à une pour avoir les différentes informations et les ajouter dans le HTML de sortie.

On voit que qu'on exécute les fonctions CPU et RAM deux fois, pour une question d'ordre d'exécution.

Une première qui nous sert à avoir la couleur que nous mettons ensuite dans chaque case du tableau et une deuxième pour la valeur en pourcentage au moment venu.

```
function PPS2022Supervision #Fonction qui contient toutes les autres fonctions pour la supervision
{
    foreach($Nom in $NomsServeurs)
    {
        PPS2022SupervisionCPUUsage
        PPS2022SupervisionRAMUsage
        PPS2022SupervisionStatut
        "<tr class=" + $ColorStatut + "><td class>" + $Nom + "</td>" | Out-File $HTMLSupervision -Append utf8 #3
        PPS2022SupervisionAdresseIP
        PPS2022SupervisionOSInformation
        PPS2022SupervisionCPUUsage
        "<td class=" + $ColorCPU + ">" + $CPUUsage + " %" + "</td>" | Out-File $HTMLSupervision -Append utf8 #3
        PPS2022SupervisionRAMUsage
        "<td class=" + $ColorRAM + ">" + $RAMUsage + " %" + "</td>" | Out-File $HTMLSupervision -Append utf8 #3
        PPS2022SupervisionStatut
        "<td class=" + $ColorStatut + ">" + $Statut + "</td>" | Out-File $HTMLSupervision -Append utf8 #3
    }
}
```

Améliorations envisageables :

Sur la partie supervision on pense qu'on pourrait améliorer ça :

- Éviter la double utilisation des fonctions CPU et RAM pour avoir la couleur, c'est dommage.
- Améliorer l'auto-refresh pour éviter les pages en partie blanche de manière aléatoire
- On aurait pu faire en sorte qu'au déploiement d'un serveur il aille inscrire son nom automatiquement dans le fichier texte des noms de serveurs.

III) Installation des services :

1) Installation des services via poste local

A. Vérification de l'état du service WinRM

```

1  $ErrorActionPreference = "stop"
2
3  3 references
4  function PPS2022TestWinRM{
5      #On récupère le statut du service WinRM, s'il est éteint alors on démarre le service
6      if(Get-Service -Name Winrm | Where-Object {$_.status -NotLike "Running"})
7      {
8          try{
9              Start-Service WinRM
10             Write-Host -Object "Le service WinRM a été démarré" -ForegroundColor Yellow
11         }
12         catch{
13             #Si le try ne fonctionne pas alors on affiche l'erreur.
14             Write-Host $Error[0].Exception.Message -ForegroundColor Red
15         }
16     }
17
18     else
19     {
20         Write-Host -Object "Le service WinRM est déjà démarré" -ForegroundColor Green
21     }
22
23
24  Read-Host "Appuyez sur ENTREE pour continuer..."
25
26  }

```

- 1- On crée la fonction « PPS2022TestWinRM » qui nous servira à déterminer si le service WinRM est bien en exécution sur le poste local. Ce service est nécessaire pour pouvoir gérer des ordinateurs à distance.

On teste donc si le service WinRM est éteint.

- 2- Si le service WinRM est éteint alors on essaie de démarrer le service et on affiche que le service WinRM a été démarré.
- 3- Si l'étape d'avant renvoie une erreur alors on stop l'exécution et on affiche l'erreur.
- 4- Si le service est déjà lancé alors on affiche que le service est déjà en cours d'exécution.

B. Installation & Configuration du service Active Directory

```
function PPS2022InstallAD{
    #Initialisation de la session Powershell à distance
    Try{
        $sessionId = New-PSSession -ComputerName pps2022testdc.switzerlandnorth.cloudapp.azure.com -UseSSL -SessionOption (New-PSSessionOption -SkipCACheck -SkipCNCheck) -Credential Administrate
    }
    Catch{
        Write-Host $Error[0].Exception.Message -ForegroundColor Red
    }
}
```

Création de la fonction « **PPS2022InstallAD** ». Cette fonction a pour but d'installer le rôle Active Directory ainsi que la forêt « **PPS2022.local** ».

Dans un premier temps, on réalise donc un Try/Catch afin d'ouvrir une PSSession sur le serveur distant et on affiche l'erreur si le Try échoue.

```
#Si la PSSession a bien été initialisée alors on envoie les commandes suivantes sur le serveur distant
if($sessionId){
    #Récupération du SafePassword pour la forêt ActiveDirectory
    $safePassword = Read-Host "Entrer le SafePassword pour la forêt Active Directory" -AsSecureString

    Invoke-Command -Session $sessionId -ArgumentList $safePassword -Scriptblock {
        $verifAD = Get-ADDomainController -ErrorAction SilentlyContinue

        if(!$verifAD)
        {
            Install-WindowsFeature AD-Domain-Services -IncludeManagementTools
            Install-ADDSForest -DomainName PPS2022.local -DomainNetBiosName PPS2022 -InstallDns:$true -NoRebootOnCompletion:$false -SafeModeAdministratorPassword $args[0] -Force
        }
    }
}
```

- 1- Si l'initialisation de la PSSession a fonctionné, alors on demande à entrer le SafePassword dans une SecureString afin de cacher et chiffrer ce dernier.
- 2- Ensuite on envoie des commandes sur le serveur distant à l'aide de la Cmdlet « Invoke-Command », on lui passe en argument le SafePassword entré plus tôt.
 - On commence par vérifier si le service ActiveDirectory existe déjà sur le serveur, si ce n'est pas le cas alors on lance l'installation du service Active Directory.
 - Une fois l'installation terminée on configure une nouvelle forêt avec le nom « **PPS2022.local** ».
 - Nous installons le service DNS, nous passons l'argument « **NoRebootOnCompletion** » sur « **False** » afin que le serveur redémarre à la fin de la configuration et nous donnons le SafePassword passé en argument lors de notre « **Invoke-Command** ».

C. Installation & Configuration du service DHCP

```
Function PPS2022InstallDHCP{
    try {
        $sessionId = New-PSSession -ComputerName pps2022testdc.switzerlandnorth.cloudapp.azure.com -UseSSL -SessionOption (New-PSSessionOption -SkipCACheck -SkipCNCheck) -Credential "PPS2022\Adm..."
    }
    catch {
        Write-Host $Error[0].Exception.Message -ForegroundColor Red
    }

    if($sessionId){
        Invoke-Command -Session $sessionId -Scriptblock {
            Install-WindowsFeature DHCP -IncludeManagementTools
            Add-DHCPServerInDC -DNSName ([System.Net.Dns]::GetHostByName($env:computerName).HostName)
            Set-DHCPServerV4OptionValue -DNSServer 192.168.1.10 -DNSDomain PPS2022.local -Router 192.168.1.1
            Add-DHCPServerV4Scope -Name "Pool PPS2022" -StartRange 192.168.1.50 -EndRange 192.168.1.100 -SubnetMask 255.255.255.0 -Description "Plage DHCP pour projet PPS2022"
        }

        Remove-PSSession $sessionId
        Write-Host -Object "Le service DHCP a été installé"
    }
}
```

- 1) On commence par initialiser une PSSession qu'on stock dans la variable « **\$sessionId** » avec un Try/Catch. Si celle-ci ne fonctionne pas alors on renvoie le message d'erreur.
- 2) Si la variable « **\$sessionId** » a bien été (remplie) alors on envoie les commandes suivantes à l'aide du Cmdlet « **Invoke-Command** » :
 - « **Install-WindowsFeature DHCP -IncludeManagementTools** » pour installer le service DHCP sur le serveur.
 - « **Add-DHCPServerInDC -DNSName ([System.Net.Dns]::GetHostByName(\$env:computerName).HostName)** » pour ajouter le DHCP au DomainController. La valeur passée à l'argument « **-DNSName** » permet de récupérer le FQDN du serveur.
 - « **Set-DHCPServerV4OptionValue** » :
 - **-DNSServer** : On spécifie l'adresse du serveur DNS.
 - **-DNSDomain** : On passe en argument ici le nom de domaine (PPS2022.local)
 - **-Router** : On donne ici l'adresse IP de la passerelle (192.168.1.1)
 - « **Add-DHCPServerV4Scope** » :
 - **-Name** : Nom du pool DHCP.
 - **-StartRange** : Adresse IP de la première adresse IP du pool.
 - **-EndRange** : Adresse IP de la dernière adresse IP du pool.
 - **-SubnetMask** : Masque de sous-réseau pour le pool.
 - **-Description** : Description pour le pool DHCP.

D. Menu du script

```
function DisplayMenu{
    $continue = $true
    while ($continue){
        write-host "-----SCRIPT INSTALLATION DES SERVICES -----"
        write-host "1. Verifier WinRM sur le poste local"
        write-host "2. Installer le service Active Directory"
        write-host "3. Installer le service DHCP"
        write-host "4. Exit"
        write-host "-----"
        $choix = read-host "Faire un choix"

        switch ($choix){
            1{PPS2022TestWinRM}
            2{PPS2022InstallAD}
            3{PPS2022InstallDHCP}
            4{$continue = $false}

            default {Write-Host "Choix invalide"-ForegroundColor Red}
        }
    }
}

DisplayMenu
```

- 1) On créer la fonction « **PPS2022DisplayMenu** », On commence par initialiser la variable « **\$continue** » sur « **\$true** ». Celle-ci nous servira pour afficher le menu.
On continue avec un « **while(\$continue)** » afin d’afficher le menu tant que la variable « **\$continue** » est sur la valeur « **\$true** ».
On affiche donc à l’aide des « **Write-Host** » notre menu qui sera affiché sur le terminal.
- 2) Pour réaliser notre menu, on se sert de la condition « **switch** » qui nous permettra d’exécuter un code selon la valeur entrée par l’utilisateur. Pour chaque valeur on exécute une fonction donnée.
Pour quitter le menu et donc le script, pour une valeur donnée on donne à la variable « **\$continue** » la valeur « **\$false** » afin que la boucle se coupe et que le code se termine.
- 3) On appelle la fonction « **PPS2022DisplayMenu** »

Difficultés rencontrées :

- Utilisation des PSSession sur les serveurs Azure,
- La gestion et l'affichage des erreurs ont quelquefois été problématiques (Arrêt du code, compréhension de l'erreur, source de l'erreur, etc.)
- Nous avons dû passer le GIT en public afin de récupérer les scripts automatiquement pour le déploiement avec Template ARM.
- Bien gérer le redémarrage de l'AD (Attente de disponibilité du service Powershell)

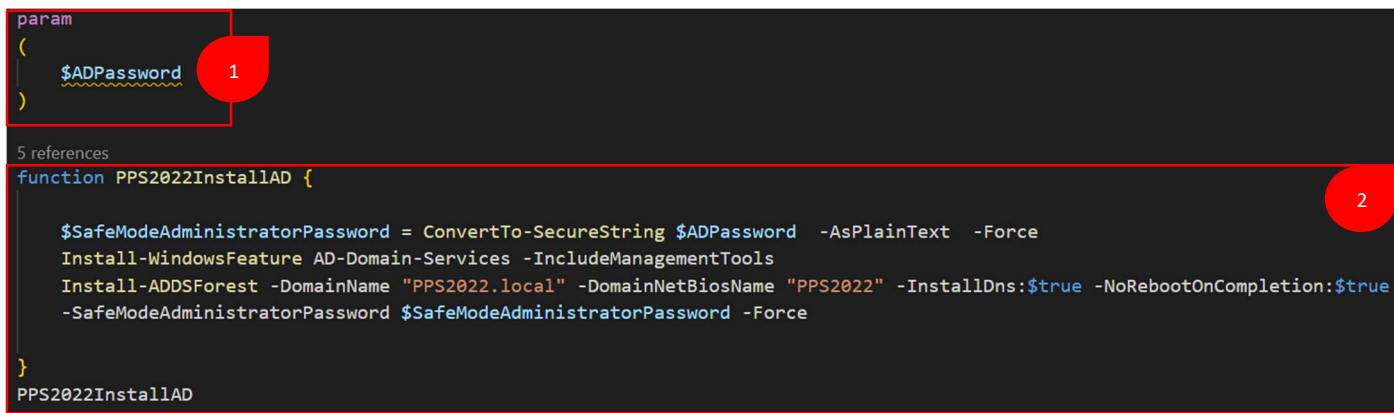
Améliorations à apporter :

- Nous sommes actuellement obligés de prendre la main sur le serveur afin de créer une règle de pare-feu et générer un certificat pour la connexion en PSSession. Il faudrait que l'on automatise cette partie.
- La gestion des erreurs n'est pas optimale.
- Mettre en place une vérification de l'existence de certains services ou paramètres avant de les installer / créer.

3) Installation des services AD, DNS, DHCP via template ARM

Le code suivant est totalement exécuté sur les serveurs créés via la template ARM.

A. Installation du service ActiveDirectory



```
param
(
    $ADPassword
)

5 references
function PPS2022InstallAD {

    $SafeModeAdministratorPassword = ConvertTo-SecureString $ADPassword -AsPlainText -Force
    Install-WindowsFeature AD-Domain-Services -IncludeManagementTools
    Install-ADDSForest -DomainName "PPS2022.local" -DomainNetBiosName "PPS2022" -InstallDns:$true -NoRebootOnCompletion:$true
    -SafeModeAdministratorPassword $SafeModeAdministratorPassword -Force
}
PPS2022InstallAD
```

1. On commence par récupérer le mot de passe ActiveDirectory envoyé en paramètre par la template ARM
2. On crée ensuite la fonction « **PPS2022InstallAD** » qui nous servira pour l'installation du service Active Directory ainsi que pour la configuration de la forêt « **PPS2022.local** ». On récupère le mot de passe ActiveDirectory qu'on convertit en SecureString et qu'on stocke dans « \$SafeModeAdministratorPassword ». On installe ensuite le service active Directory et la forêt « PPS2022.local », on donne en paramètre « -InstallDns:\$true » afin d'installer le service DNS, « -NoRebootOnCompletion:\$True » pour éviter le redémarrage automatique du poste à la fin de l'installation. On finit par le mot de passe du SafeMode. On appelle ensuite la fonction pour que celle-ci s'exécute.

B. Paramétrage de la tâche planifiée pour provisionning AD

```
function PPS2022ScheduledTask {
    New-item -Path "C:\\" -Name "Scripts" -ItemType "directory"
    cd C:\Scripts
    Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/Jonathan28260/ProjetAnnuel_Powershell/main/PPS2022-CreationUoUtilisateurs.ps1" -OutFile C:\Scripts\PPS2022-CreationUoUtilisateurs.ps1
    $action = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '-File "PPS2022-CreationUoUtilisateurs.ps1"'
    $trigger = New-ScheduledTaskTrigger -AtStartup -RandomDelay 00:00:30
    $principal = New-ScheduledTaskPrincipal -GroupId "BUILTIN\Administrators" -RunLevel Highest
    $definition = New-ScheduledTask -Action $action -Principal $principal -Trigger $trigger -Description "Run CreateUserAD at startup"
    Register-ScheduledTask -TaskName "CreateUserAD" -InputObject $definition
}
PPS2022ScheduledTask
```

1. On crée un nouveau dossier « **Scripts** » à la racine « **C :** » et on se place dans ce nouveau dossier.
2. On réalise un « **Invoke-WebRequest** » afin de récupérer le code sur le GitHub et de le placer dans le fichier « **C:\Scripts\PPS2022-CreationUoUtilisateurs.ps1** »
3. On définit l'action que notre tâche exécutera, dans notre cas ce sera l'exécution du script précédemment créé.
On lui spécifie la condition de démarrage (Au démarrage de la machine)
On donne ensuite le niveau de droit pour l'exécution du script en spécifiant le groupe « **BUILTIN\Administrators** ».
On finit par enregistrer et activer notre tâche planifiée.
4. On appelle la fonction précédemment créée.

C. Installation du service DHCP

```
function PPS2022InstallDHCP {
    Install-WindowsFeature DHCP -IncludeManagementTools
    Add-DHCPServerInDC -DNSName SRV-AD.PPS2022.local
    Set-DHCPServerv4OptionValue -DNSServer 192.168.1.10 -DNSDomain PPS2022.local -Router 192.168.1.1
    Add-DHCPServerv4Scope -Name "Pool PPS2022" -StartRange 192.168.1.50 -EndRange 192.168.1.100 -SubnetMask 255.255.255.0 -Description "Plage DHCP pour projet PPS2022"
    Restart-Computer -Force
}
PPS2022InstallDHCP
```

1. On lance l'installation du service DHCP.
On lie notre DHCP à notre domaine créé précédemment.
2. On spécifie les paramètres de notre serveur DHCP (Serveur DNS, Nom de domaine, IP de la passerelle)
On ajoute notre pool DHCP « **Pool PPS2022** » avec le range et le masque de sous réseau.
Une fois toutes les étapes réalisées on peut redémarrer notre serveur afin de finaliser les installations.

3) Installation du serveur WEB + Supervision

A) Installation du service IIS

```
param(
    $DomainPassword
)

1 reference
function PPS2022installIIS{
    #Installer IIS et les composants
    Install-WindowsFeature web-server -IncludeManagementTools
    Install-WindowsFeature web-mgmt-service
    Set-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\WebManagement\Server -Name EnableRemoteManagement -Value 1
    install-windowsfeature Web-Asp-Net45
    install-windowsfeature Web-ISAPI-Ext
    install-windowsfeature web-windows-auth
    Net Stop WMSVC
    Net Start WMSVC
    New-NetFirewallRule -Name "IIS80" -DisplayName "IIS80" -Enabled True -Profile Any -Action Allow -Direction Inbound -LocalPort 80 -Protocol TCP
}
PPS2022installIIS
```

1. On commence par récupérer le mot de passe AD fournit par la template ARM.
2. On crée ensuite la fonction « **PPS2022installIIS** »
On commence par installer les services « **web-server** » et « **web-mgmt-service** »
3. On modifie une valeur de clé de registre afin d'autoriser la gestion à distance.
4. On installe les paquets « **Web-Asp-Net45** », « **Web-ISAPI-Ext** », « **web-windows-auth** »
5. On redémarre le service « **WMSVC** »
6. On ajoute une règle de firewall pour autoriser le trafic entrant sur le port 80 (http)

B) Paramétrage de la tâche planifiée (Supervision)

```
function PPS2022ScheduledTask{
    New-item -Path "C:\\" -Name "Scripts" -ItemType "directory"
    ADD-content -path "C:\Scripts\PPS2022-Loop-Supervision.ps1" -value 'while($true){ C:\Scripts\PPS2022-Supervision.ps1
    sleep -seconds 10 }'
    ADD-content -path "C:\Scripts\NomsServeurs.txt" -value "PPS2022-SRV-DC`nPPS2022-SRV-IIS"
    cd C:\Scripts
    Invoke-WebRequest -UseBasicParsing -Uri "https://raw.githubusercontent.com/Jonathan28260/ProjetAnnuel_Powershell/main/PPS2022-Supervision.ps1"
    -OutFile C:\Scripts\PPS2022-Supervision.ps1
    $action = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '-File "C:\Scripts\PPS2022-Loop-Supervision.ps1"'
    $trigger = New-ScheduledTaskTrigger -AtStartup -RandomDelay 00:00:30
    $principal = New-ScheduledTaskPrincipal -GroupId "BUILTIN\Administrators" -RunLevel Highest
    $definition = New-ScheduledTask -Action $action -Principal $principal -Trigger $trigger -Description "Run Supervision at startup"
    Register-ScheduledTask -TaskName "Supervision" -InputObject $definition
}
PPS2022ScheduledTask
```

1. On commence par créer le dossier « **Scripts** » à la racine « **C :** ».

On crée ensuite un script PowerShell qui exécutera notre code de supervision toutes les 10 secondes dans le dossier précédemment créé.

On finit par créer un document texte dans lequel on vient ajouter le nom de nos serveurs à superviser.
2. On va récupérer le contenu du code « **PPS2022-Supervision.ps1** » se situant sur le Github et on vient l'intégrer dans le script « **PPS2022-Supervision.ps1** » en local sur le serveur.
3. On paramètre ensuite notre tâche planifiée « **Supervision** » qui exécutera donc le script « **PPS2022-Loop-Supervision.ps1** » au démarrage de la machine avec un délai de lancement aléatoire de 30 secondes et des droits d'administrateur (**BUILTIN\Administrators**).

C) Enregistrement dans le domaine

```
function JoinDomain{
    #Ajout du DC en DNS pour rejoindre le domaine#
    Set-DnsClientServerAddress -InterfaceAlias "Ethernet" -ServerAddresses "192.168.1.10"

    #Joindre le domaine#
    $Domain = "PPS2022.local"
    $username = "PPS2022\Administrateur"
    $Password = ConvertTo-SecureString $DomainPassword -AsPlainText -Force
    $credential = new-object -typename System.Management.Automation.PSCredential -argumentlist $username, $password
    Add-Computer -DomainName $Domain -Credential $credential
    Restart-Computer -Force
}
JoinDomain
```

1. On commence par paramétrer notre serveur AD/DHCP/DNS précédemment créé en DNS sur notre serveur IIS.
2. On stocke dans des variables : le nom de domaine, le nom d'utilisateur ayant les droits sur le domaine, le mot de passe récupéré en début de script et convertit en **SecureString**.
3. On crée un objet pour stocker les credentials et on rejoint le domaine. Après redémarrage notre serveur est bien dans le domaine.

Problématique rencontrée :

La tâche planifiée pour l'exécution du script PPS2022-LoopSupervision.ps1 ne se lance pas automatiquement au démarrage de la machine mais fonctionne bien lorsqu'elle est lancée manuellement.

IV) Création UO et utilisateurs

1. Création des unités d'organisations

Cette fonctionnalité a pour but de créer une hiérarchie « par défaut » de trois unités d'organisations : Direction, Compta, Marketing. Ainsi que de trois groupes rattachés à ces unités d'organisations afin d'y rajouter par la suite les utilisateurs.

```
Configuration / Paramétrage AD

0 references
function PPS2022CreationUO {
    New-ADOrganizationalUnit "Direction" -Path "OU=DC=PPS2022,DC=local"
    New-ADGroup -Name "Direction Users" -GroupeScope Global -Path "OU=Direction,DC=PPS2022,DC=local"

    New-ADOrganizationalUnit "Marketing" -Path "OU=DC=PPS2022,DC=local"
    New-ADGroup -Name "Marketing Users" -GroupeScope Global -Path "OU=Marketing,DC=PPS2022,DC=local"

    New-ADOrganizationalUnit "Compta" -Path "OU=DC=PPS2022,DC=local"
    New-ADGroup -Name "Compta Users" -GroupeScope Global -Path "OU=Compta,DC=PPS2022,DC=local"
}
```

1. Création de la fonction ainsi que sur la première ligne l'unité d'organisation.
2. Création des groupes rattaché à l'unité d'organisation en question.

2. Création des utilisateurs via import csv

Ce script a pour but de créer des utilisateurs dans l'Active Directory depuis une importation d'un fichier CSV. Il est possible de personnaliser les informations/détails de l'utilisateur dans le fichier .csv. A noter que le fichier csv est encodé en UTF8.

```

24 function PPS2022CreationUtilisateurs{
25     $UtilisateurAD = Import-Csv C:\Scripts\NouveauxUtilisateurs.CSV -Delimiter ";"
26     Invoke-WebRequest -Uri "https://raw.githubusercontent.com/Jonathan28260/ProjetAnnee1_Powershell/main/NouveauxUtilisateurs.csv" -OutFile C:\Scripts\NouveauxUtilisateurs.csv
27     foreach ($Utilisateur in $UtilisateurAD)
28     {
29         $Username = $Utilisateur.identifiant
30         $Password = $Utilisateur.motdepasse
31         $Firstname = $Utilisateur.prenom
32         $Lastname = $Utilisateur.nom
33         $OU = $Utilisateur.ou
34     }
35
36     # Nous allons vérifier si l'utilisateur existe déjà et renvoyer un message dans le cas où il existe
37     # S'il n'existe pas il sera créé
38     # Le compte sera créé dans l'unité d'organisation indiquée dans la variable $OU du fichier CSV
39
40     if (Get-ADUser -F {SamAccountName -eq $Username})
41     {
42         Write-Warning "L'utilisateur $Username existe déjà dans l'Active Directory."
43     }
44     else
45     {
46         New-ADUser `
47             -SamAccountName $Username `
48             -UserPrincipalName "$Username@PPS2022.local" `
49             -Name "$Firstname $Lastname" `
50             -GivenName $Firstname `
51             -Surname $Lastname `
52             -Enabled $True `
53             -ChangePasswordAtLogon $True `
54             -DisplayName "$Lastname, $Firstname" `
55             -Path "OU=$OU,DC=PPS2022,DC=local" `
56             -AccountPassword (Convertto-SecureString $Password -AsPlainText -Force)
57     }
58 }

```

1. Création de la fonction et stockage du fichier csv dans la variable « utilisateurad » afin de récupérer le fichier nous faisons appel à une commande « webrequest » qui va aller chercher le fichier csv.
2. Ensuite nous allons stocker dans différentes variables prédéfinies par les colonnes du fichier csv les données dont nous avons besoin pour créer l'utilisateur.
3. Une fois les données récupérées nous allons vérifier que l'utilisateur en question ne soit pas déjà présent en récupérant l'identifiant qui est stocké dans la variable « Username » et qui est comparé à « SamAccountName » qui correspond à l'identifiant de l'utilisateur dans l'AD.
4. Si l'utilisateur n'existe pas alors il sera créé grâce aux informations renseignées dans le fichier csv puis stocké dans les variables. Des paramétrages ont été rajoutés comme l'activation de l'utilisateur ainsi que le changement de mot de passe lors de la première connexion.

Problématiques :

Le fichier Excel de base n'était pas conforme car il n'était pas encodé en UTF8

Comment exporter/déployer le csv.

Axes d'améliorations :

Créer l'UO si celle renseignée dans le fichier csv n'existe pas.