Link to download the script:

<https://github.com/ArnaudPain/CVAD-Service---Automation>

Script Code: #Requires -Version 5.0

## GRL Do you really need version 5.0 minimum? Far too many environments don't get PowerShell updated

#This File is in Unicode format. Do not edit in an ASCII editor. Notepad++ UTF-8-BOM

#region help text

<#

.SYNOPSIS

Deploy 2 Windows Server 2016 on Azure, domain join and deploy Citrix Cloud Connector in Azure

.DESCRIPTION

The script will create Resource Location in Citrix Cloud.

It will deploy 2 VMs in Azure based on Windows Server 2016 Template (if you want to use Windows Server 2019, just change line 46 and replace 2016 with 2019

When VMs are deployed they will be domain-joined

Then Citrix Cloud Connector software is downloaded from my ShareFile FTP and deployed

Requires -RunAsAdministrator (or elevated PowerShell session)

Requires existing domain controller (powered on!)

Requires a Citrix Cloud API key see --> https://docs.citrix.com/en-us/citrix-cloud/citrix-cloud-management/identity-access-management.html

Requires Active Directory on Azure or Site to Site connectivity between Azure and on-premises

.NOTES

NAME: Citrix-Azure-CC-v1.0.ps1

VERSION: 1.00

AUTHOR: Arnaud Pain

LASTEDIT: September 3, 2020

#>

#endregion

# VARIABLES TO BE SET BEFORE RUNNING THE SCRIPT

## GRL Have all these in a Param() block then the user doesn't need to edit the script. If they want to hard code values, these can be set as default values for each parameter

# Citrix Cloud credentials

$CustomerID = "2g60i8n7zhxd" #Read-Host "1/11 Please provide your Customer ID"

$ClientID = "d03d1473-cc68-4486-9bf3-02d7c7f455a0" #Read-Host "2/11 Please provide your Client ID"

$ClientSecret = "t8m5YuezfTxyo8JvBppuCw==" #Read-Host "3/11 Please provide your Client Secret"

$AzureResourceGroupLocation = "eastus2" #Read-Host "4/11 Please provide the Azure Region you want to use"

$AzureVNetName = "CTX-VNET" #Read-Host "5/11 Please provide the Azure Virtual Network you want to use (must have a domain controller on it or site-to-site connection with om-premises)"

$AzureResourceGroupName = "CTX-RG" #Read-Host "6/11 Please provide the Azure Resource Group you want to use (will be created if not existing)"

$AzureVNetResourceGroupName = $AzureResourceGroupName #Read-Host "6/12 Please provide the Azure Resource Group for the Virtual Network you want to use (will be created if not existing)"

$AzureDiagnosticsStorageAccountName = -join ((48..57) + (97..122) | Get-Random -Count 24 | % {[char]$\_}) #Read-Host "8/15 Please provide the name of the Azure Storage account to be used for diagnostics (will be created if not existing)"

$AzureDiagnosticResourceGroupName = $AzureResourceGroupName #Read-Host "7/12 Please provide the name of the Resource Group for the Azure Storage account (will be created if not existing)"0

$DomainName = "ctx.local" #Read-Host "7/11 Please provide the name of the domain (fqdn like domain.local)"

$CTX\_Resource\_Location\_Name = "EAST-US-2" #Read-Host "8/11 Please provide the name of Resource Location to be created in Citrix Cloud"

$AzureStorageAccountName = -join ((48..57) + (97..122) | Get-Random -Count 24 | % {[char]$\_})# Read-Host "12/15 Please provide the name of the Azure Storage account to be used (will be created if not existing)"

$OSType = "2016" #Read-Host "9/12 Please provide the Windows version you want to use (2016 or 2019)"

$CloudConnectorMachineName1 = "AZ-CC-01" #Read-Host "10/11 Please provide the name for the 1st Cloud Connector"

$CloudConnectorMachineName2 = "AZ-CC-02" #Read-Host "11/11 Please provide the name for the 2nd Cloud Connector"

$CloudConnectorMachineType = "Standard\_DS2\_v2"

$CloudConnectorDiskType = "Premium\_LRS"

$AzureSubnetName = "default"

# FTP Variables

$Username = "arnaudpain/ftp@arnaud.biz"

## GRL Bad idea to have plain text passwords in or passed to scripts, use PSCredential objects instead, or Secure Strings. Clear text password can be retrieved by GetNetworkCredential().Password to pass to things which need clear texts passwords

$Password = "9$<rZK-k"

# Citrix Delivery controller

$DDC= "" # To be filled before running the script

# Miscellaneous

$LocalTempFolder = "C:\Temp"

$UsersGroupName = "Domain Users"

# PREREQUISITES

# Setup script running time

$ScriptStopWatch = [System.Diagnostics.StopWatch]::StartNew()

# Check if user is admin and script is running elevated

$CurrentPrincipal = New-Object Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())

if (!($CurrentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))) {

## GRL Use Write-Error not Write-Host

Write-Host "User does not have admin rights. Are you running this in an elevated session?" -ForegroundColor Red

Write-Host "Stopping script." -ForegroundColor Red

Return

}

# Enable TLS 1.2 required for Citrix Bearer Token

[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

## GRL Has this code come from somewhere/someone else? If so, put a reference to it in a comment as it is polite

# DEFINE FUNCTIONS

Function Add-JDAzureRMVMToDomain {

param(

[Parameter(Mandatory = $true)]

[string]$DomainName,

[Parameter(Mandatory = $false)]

[System.Management.Automation.PSCredential]$Credentials = $ADCredentials,

[Parameter(Mandatory = $true, ValueFromPipeline = $true, ValueFromPipelineByPropertyName = $true)]

[Alias('VMName')]

[string]$Name,

[Parameter(Mandatory = $true, ValueFromPipeline = $true, ValueFromPipelineByPropertyName = $true)]

[ValidateScript( { Get-AzureRmResourceGroup -Name $\_ })]

[string]$ResourceGroupName

)

begin {

# Define domain join settings (username/domain/password)

$Settings = @{

Name = $DomainName

User = $Credentials.UserName

Restart = "true"

Options = 3

}

$ProtectedSettings = @{

Password = $Credentials.GetNetworkCredential().Password

}

Write-Verbose -Message "Domainname is: $DomainName"

}

process {

try {

$RG = Get-AzureRmResourceGroup -Name $ResourceGroupName

$JoinDomainHt = @{

ResourceGroupName = $RG.ResourceGroupName

ExtensionType = 'JsonADDomainExtension'

Name = 'joindomain'

Publisher = 'Microsoft.Compute'

TypeHandlerVersion = '1.0'

Settings = $Settings

VMName = $Name

ProtectedSettings = $ProtectedSettings

Location = $RG.Location

}

Write-Verbose -Message "Joining $Name to $DomainName"

Set-AzureRMVMExtension @JoinDomainHt

}

catch {

Write-Warning $\_

}

}

end { }

}

## GRL PowerShell functions should be verb-noun where verb is from the approved list

Function RegisterRP {

Param(

[string]$ResourceProviderNamespace

)

## GRL No need for semi-colons

Write-Host "Registering Azure resource provider '$ResourceProviderNamespace'";

Register-AzureRmResourceProvider -ProviderNamespace $ResourceProviderNamespace;

## GRL Error handling/reporting?

}

# IMPORT MODULES

# Azure - Import necessary modules

Write-Host "1. Import necessary PowerShell modules - Part 1" -ForegroundColor Green

# Azure Resource Manager module

if (Get-Module -ListAvailable -Name AzureRM) {

## GRL Use Write-Warning or Write-Output not Write-Host as you can't redirect it, e.g. to silence output

Write-Host "Azure RM module already available, importing..." -ForegroundColor Yellow

## GRL No need to pipe result of Import-Module unless you use -passThru

Import-Module AzureRM | Out-Null

} else {

## GRL Use Write-Warning or Write-Output not Write-Host as you can't redirect it, e.g. to silence output

Write-Host "Azure RM module not yet available, installing..." -ForegroundColor Yellow

Install-Module -Name AzureRM -scope AllUsers -Confirm:$false -force

## GRL No need to pipe result of Import-Module unless you use -passThru

Import-Module AzureRM | Out-Null

}

# AUTHENTICATION

## GRL I'm not a fan of Write-Host but in this context, getting user input, it's ok

Write-Host "2. Ask user for credentials" -ForegroundColor Green

# Azure

Write-Host "\*\*\* Azure login \*\*\*" -ForegroundColor Yellow

Login-AzureRmAccount

# Virtual machines - Local administrator

Write-Host "\*\*\* Virtual machine - Local administrator \*\*\*" -ForegroundColor Yellow

Write-Host "Please enter the Windows administrator credentials to be set on the Cloud Connector" -ForegroundColor Yellow

$CloudConnectorAdminUsername = Read-Host "Username"

$CloudConnectorAdminPassword = Read-Host "Password" -AsSecureString

# Virtual machines - Domain join

Write-Host "\*\*\* Virtual machine - Domain join \*\*\*" -ForegroundColor Yellow

Write-Host "Enter the credentials for a user that is allowed to join machines to the domain" -ForegroundColor Yellow

## GRL I always use the -Message to Get-Credential so it doesn't look like a possibly random credential prompt which may be phishing

$ADCredentials = Get-Credential

## GRL do you need to test if Get-Credential was cancelled (e.g. escape key) - would need to be in a try/catch block to hide the error from the user

# Azure - Import necessary modules

Write-Host "3. Import necessary PowerShell modules - Part 2" -ForegroundColor Green

# Azure Active Directory module

if (Get-Module -ListAvailable -Name AzureAD) {

Write-Host "Azure AD module already available, importing..." -ForegroundColor Yellow

Import-Module AzureAD | Out-Null

} else {

Write-Host "Azure AD module not yet available, installing..." -ForegroundColor Yellow

Install-Module -Name AzureAD -Scope AllUsers -Confirm:$false -Force

## GRL no need for pipe. If you do have things that you need to throw away, assign it to $null as it is more efficient

Import-Module AzureAD | Out-Null

}

# MAIN SCRIPT

# Citrix Cloud - Bearer Token

Write-Host "4. Citrix Cloud - Get Bearer Token" -ForegroundColor Green

$Body = @{

"ClientId" = $ClientID;

"ClientSecret" = $ClientSecret

}

$PostHeaders = @{

"Content-Type" = "application/json"

}

## GRL Better to have this as a parameter with this as default or right at the top of the script so easy to change

$TrustURL = "https://trust.citrixworkspacesapi.net/root/tokens/clients"

$Response = Invoke-RestMethod -Uri $TrustURL -Method POST -Body (ConvertTo-Json -InputObject $Body) -Headers $PostHeaders

## GRL Error checking, what if $Response is $null or other errors

$BearerToken = $Response.token

$Token = "CwsAuth Bearer=" + $BearerToken

# Citrix Cloud - Create Resource Location

Write-Host "5. Citrix Cloud - Create Resource Location" -ForegroundColor Green

$Body = @{

"Name" = $CTX\_Resource\_Location\_Name

}

$Headers = @{

"Accept" = "application/json";

"Authorization" = $Token;

"Content-Type" = "application/json"

}

$Json = ConvertTo-Json -InputObject $Body

$ResourceURL = "https://registry-eastus-release-b.citrixworkspacesapi.net/" + $CustomerID + "/resourcelocations"

$Resource = Invoke-WebRequest -Method POST -uri $ResourceURL -body $json -Headers $headers -UseBasicParsing

$CTXCloudResourceID = ($Resource.Content | ConvertFrom-Json).ID

# Create Azure storage account

Write-Host "6. Azure - Create Azure resource groups (if needed)" -ForegroundColor Green

# Check for existing resource group and create new one if needed

$AzureResourceGroup = Get-AzureRmResourceGroup -Name $AzureResourceGroupName -ErrorAction SilentlyContinue

if (!$AzureResourceGroup) {

Write-Host "Resource group '$AzureResourceGroupName' does not exist yet" -ForegroundColor Yellow

Write-Host "Creating resource group '$AzureResourceGroupName' in location '$AzureResourceGroupLocation'" -ForegroundColor Yellow

New-AzureRmResourceGroup -Name $AzureResourceGroupName -Location $AzureResourceGroupLocation

} else {

Write-Host "Using existing resource group '$AzureResourceGroupName'" -ForegroundColor Yellow

}

if ($AzureVNetResourceGroupName -ne $AzureResourceGroupName) {

Write-Host "Different resource group specified for Virtual Networks" -ForegroundColor Yellow

$AzureVNetResourceGroup = Get-AzureRmResourceGroup -Name $AzureVNetResourceGroupName -ErrorAction SilentlyContinue

if (!$AzureVNetResourcegroup) {

Write-Host "Virtual network resource group '$AzureVNetResourceGroupName' does not exist yet" -ForegroundColor Yellow

Write-Host "Creating virtual network resource group '$AzureVNetResourceGroupName' in location '$AzureResourceGroupLocation'" -ForegroundColor Yellow

New-AzureRmResourceGroup -Name $AzureVNetResourceGroupName -Location $AzureResourceGroupLocation

## GRL what if this failed?

} else {

Write-Host "Using existing virtual network resource group '$AzureVNetResourceGroupName'" -ForegroundColor Yellow

}

} else {

Write-Host "Specified virtual network resource group is identical to the VM resource group" -ForegroundColor Yellow

}

if ($AzureDiagnosticResourceGroupName -ne $AzureResourceGroupName) {

Write-Host "Different resource group specified for diagnostic information" -ForegroundColor Yellow

$AzureDiagnosticResourceGroup = Get-AzureRmResourceGroup -Name $AzureDiagnosticResourceGroupName -ErrorAction SilentlyContinue

if (!$AzureDiagnosticResourceGroup) {

Write-Host "Diagnostic resource group '$AzureDiagnosticResourceGroupName' does not exist yet" -ForegroundColor Yellow

Write-Host "Creating diagnostic resource group '$AzureDiagnosticResourceGroupName' in location '$AzureResourceGroupLocation'" -ForegroundColor Yellow

New-AzureRmResourceGroup -Name $AzureDiagnosticResourceGroupName -Location $AzureResourceGroupLocation

## GRL what if this failed?

} else {

Write-Host "Using existing diagnostic virtual network resource group '$AzureDiagnosticResourceGroupName'" -ForegroundColor Yellow

}

} else {

Write-Host "Specified diagnostic resource group is identical to the VM resource group" -ForegroundColor Yellow

}

# Create Azure storage account

Write-Host "7. Azure - Create Azure storage accounts (if needed)" -ForegroundColor Green

# Check for existing storage accounts and create new ones if needed

if ($AzureStorageAccount = (Get-AzureRmStorageAccount -ResourceGroupName $AzureResourceGroupName -Name $AzureStorageAccountName -ErrorAction SilentlyContinue).StorageAccountName) {

Write-Host "Azure storage account already exists" -ForegroundColor Yellow

} else {

Write-Host "Azure storage account does not exist yet, creating..." -ForegroundColor Yellow

$AzureStorageAccount = (New-AzureRmStorageAccount -ResourceGroupName $AzureResourceGroupName -Name $AzureStorageAccountName -SkuName Standard\_GRS -Location $AzureResourceGroupLocation).StorageAccountName

}

if (Get-AzureRmStorageAccount -ResourceGroupName $AzureDiagnosticResourceGroupName -Name $AzureDiagnosticsStorageAccountName -ErrorAction SilentlyContinue) {

Write-Host "Azure diagnostics storage account already exists" -ForegroundColor Yellow

} else {

Write-Host "Azure diagnostics storage account does not exist yet, creating..." -ForegroundColor Yellow

New-AzureRmStorageAccount -ResourceGroupName $AzureDiagnosticResourceGroupName -Name $AzureDiagnosticsStorageAccountName -SkuName Standard\_LRS -Location $AzureResourceGroupLocation

## GRL what if this failed?

}

# Check for existing storage keys and create new one if needed

if ($AzureStorageKeys = Get-AzureRMStorageAccountKey -ResourceGroupName $AzureResourceGroupName -Name $AzureStorageAccount -ErrorAction SilentlyContinue | Where-Object{$\_.KeyName -eq "Key1"}) {

Write-Host "Azure storage key already exists" -ForegroundColor Yellow

} else {

Write-Host "Azure storage key does not exist yet, creating..." -ForegroundColor Yellow

$AzureStorageKeys = New-AzureRmStorageAccountKey -ResourceGroupName $AzureResourceGroupName -Name $AzureStorageAccount -KeyName "Key1"

## GRL what if this failed?

}

## More efficient to have Where-Object KeyName -eq "Key1"

$AzureStorageSAKey = ($AzureStorageKeys | Where-Object {$\_.KeyName -eq "Key1"}).Value

# Various

## GRL No need for semi colon

$ResourceProviders = @("microsoft.resources", "microsoft.compute");

if ($ResourceProviders.Length) {

Write-Host "Registering Resource Providers" -ForegroundColor Yellow

foreach ($ResourceProvider in $ResourceProviders) {

RegisterRP($ResourceProvider);

}

}

if (!(Test-Path -Path $LocalTempFolder -ErrorAction SilentlyContinue)) {

## GRL Assign to $null otherwise newly created folder will be output on standard output. Even better, assign to a variable and check -ne $null in case folder creation failed

New-Item -ItemType Directory -Path $LocalTempFolder -Force

}

## GRL or in 1 line $AzureSubscriptionId = Get-AzureRmSubscription | Select-Object -First 1 -ExpandProperty Id

$AzureSubscription = Get-AzureRmSubscription | Select-Object -First 1

$AzureSubscriptionId = $AzureSubscription.Id

# Azure - Create Cloud Connector virtual machine

Write-Host "8. Azure - Create Cloud Connector Virtual Machine x 2" -ForegroundColor Green

# Various

## GRL what is an NI Name? Make variable names as meaningful as possible, even they are long as you can tab complete them

$CloudConnectorNIName1 = $CloudConnectorMachineName1 + "01"

$CloudConnectorNIName2 = $CloudConnectorMachineName2 + "02"

# Start the deployment CC 1

Write-Host "Starting Cloud Connector" $CloudConnectorMachineName1 "deployment. This can take few minutes..." -ForegroundColor Yellow

## GRL Better to have the TemplateUri string as a parameter with a default or in a section at the top of the script so easy to change if ever needed

New-AzureRmResourceGroupDeployment -ResourceGroupName $AzureResourceGroupName -Name "CloudConnector" -TemplateUri 'https://raw.githubusercontent.com/ArnaudPain/Citrix-Azure/master/Azure-Citrix-Cloud-Connector-Deployment-Template-2016.json' `

-Location $AzureResourceGroupLocation `

-NetworkInterfaceName $CloudConnectorNIName1 `

-SubnetName $AzureSubnetName `

-VirtualNetworkId "/subscriptions/$AzureSubscriptionId/resourceGroups/$AzureVNetResourceGroupName/providers/Microsoft.Network/virtualNetworks/$AzureVNetName" `

-VirtualMachineName $CloudConnectorMachineName1 `

-VirtualMachineRG $AzureResourceGroupName `

-OSDiskType $CloudConnectorDiskType `

-VirtualMachineSize $CloudConnectorMachineType `

-AdminUsername $CloudConnectorAdminUsername `

-AdminPassword $CloudConnectorAdminPassword `

-DiagnosticsStorageAccountName $AzureDiagnosticsStorageAccountName `

-DiagnosticsStorageAccountId "/subscriptions/$AzureSubscriptionId/resourceGroups/$AzureDiagnosticResourceGroupName/providers/Microsoft.Storage/storageAccounts/$AzureDiagnosticsStorageAccountName"

## GRL Check for failure?

Restart-AzureRmVM -ResourceGroupName $AzureResourceGroupName -Name $CloudConnectorMachineName1

## GRL why sleep?? Just wastes time - if you're waiting for something to happen, check if it has happened every few seconds if you can't wait on it and at least make it adjustable via a parameter with a default of 60

Start-Sleep -Seconds 60

# Domain join CC 1

Write-Host "Cloud connector" $CloudConnectorMachineName1 "created, joining machine to domain and restarting" -ForegroundColor Yellow

Get-AzureRmVM -ResourceGroupName $AzureResourceGroupName | Where-Object { $\_.Name -like $CloudConnectorMachineName1 } | Add-JDAzureRMVMToDomain -DomainName $DomainName

## GRL failure?

## GRL what are we sleeping for? Wait for that instead

Start-Sleep -Seconds 30

Restart-AzureRmVM -ResourceGroupName $AzureResourceGroupName -Name $CloudConnectorMachineName1

# Start the deployment CC 2

## GRL very similar code to just above so consider putting it in a function with parameters so if anything needs changing, you only need to change it in one place

Write-Host "Starting Cloud Connector" $CloudConnectorMachineName2 "deployment. This can take few minutes..." -ForegroundColor Yellow

New-AzureRmResourceGroupDeployment -ResourceGroupName $AzureResourceGroupName -Name "CloudConnector" -TemplateUri 'https://raw.githubusercontent.com/ArnaudPain/Citrix-Azure/master/Azure-Citrix-Cloud-Connector-Deployment-Template-2016.json' `

-Location $AzureResourceGroupLocation `

-NetworkInterfaceName $CloudConnectorNIName2 `

-SubnetName $AzureSubnetName `

-VirtualNetworkId "/subscriptions/$AzureSubscriptionId/resourceGroups/$AzureVNetResourceGroupName/providers/Microsoft.Network/virtualNetworks/$AzureVNetName" `

-VirtualMachineName $CloudConnectorMachineName2 `

-VirtualMachineRG $AzureResourceGroupName `

-OSDiskType $CloudConnectorDiskType `

-VirtualMachineSize $CloudConnectorMachineType `

-AdminUsername $CloudConnectorAdminUsername `

-AdminPassword $CloudConnectorAdminPassword `

-DiagnosticsStorageAccountName $AzureDiagnosticsStorageAccountName `

-DiagnosticsStorageAccountId "/subscriptions/$AzureSubscriptionId/resourceGroups/$AzureDiagnosticResourceGroupName/providers/Microsoft.Storage/storageAccounts/$AzureDiagnosticsStorageAccountName"

Restart-AzureRmVM -ResourceGroupName $AzureResourceGroupName -Name $CloudConnectorMachineName2

Start-Sleep -Seconds 60

# Domain join CC2

Write-Host "Cloud connector" $CloudConnectorMachineName2 "created, joining machine to domain and restarting" -ForegroundColor Yellow

Get-AzureRmVM -ResourceGroupName $AzureResourceGroupName | Where-Object { $\_.Name -like $CloudConnectorMachineName2 } | Add-JDAzureRMVMToDomain -DomainName $DomainName

Start-Sleep -Seconds 30

Restart-AzureRmVM -ResourceGroupName $AzureResourceGroupName -Name $CloudConnectorMachineName2

Write-Host "9. Citrix Cloud/Azure - Deploy Cloud Connector software on Server 1" -ForegroundColor Green

$AzureStorageContainerName1 = "cloudconinstaller1"

# Create Cloud Connector deployment script

$DeployCloudConnectorScriptContent = "

`$CustomerID = `"$CustomerID`"

`$ClientID = `"$ClientID`"

`$ClientSecret = `"$ClientSecret`"

`$CTXCloudResourceID = `"$CTXCloudResourceID`"

`$LocalFile = `"C:\cwcconnector.exe`"

`$downloadsUri = `"https://downloads.cloud.com/`" + `$CustomerID + `"/connector/cwcconnector.exe`"

`$FTPRequest = [System.Net.FtpWebRequest]::Create(`$RemoteFile)

`$FTPRequest.Credentials = New-Object System.Net.NetworkCredential(`$Username,`$Password)

`$FTPRequest.Method = [System.Net.WebRequestMethods+Ftp]::DownloadFile

`$FTPRequest.UseBinary = `$true

`$FTPRequest.KeepAlive = `$false

`$FTPResponse = `$FTPRequest.GetResponse()

`$ResponseStream = `$FTPResponse.GetResponseStream()

`$LocalFileFile = New-Object IO.FileStream (`$LocalFile,[IO.FileMode]::Create)

`[byte[]]`$ReadBuffer = New-Object byte[] 1024

`do {

`$ReadLength = `$ResponseStream.Read(`$ReadBuffer,0,1024)

`$LocalFileFile.Write(`$ReadBuffer,0,`$ReadLength)

`}

`while (`$ReadLength -ne 0)

`$LocalFileFile.Close()

`$Arguments = `"/q /customername:`$CustomerID /clientid:`$ClientID /clientsecret:`$ClientSecret /location:`$CTXCloudResourceID /acceptTermsofservice:true`"

Start-Process `$LocalFile `$Arguments -Wait"

## GRL Safer to generate a random name using $pid or similar

$ScriptFile = "InstallCloudCon1.ps1"

## Use Join-Path rather than string concatenation

$LocalScriptFile = "$LocalTempFolder\$ScriptFile"

## GRL what if the file already exists?

Set-Content -Path $LocalScriptFile -Value $DeployCloudConnectorScriptContent -Force

## GRL Could do these 3 lines in a single line for efficiency/speed

$TempScriptContent = Get-Content -Path $LocalTempFolder\$ScriptFile

$TempScriptContent = $TempScriptContent -Replace "\?", ""

Set-Content -Path $LocalScriptFile -Value $TempScriptContent -Force

# Upload Cloud Connector deployment script

$AzureStorageContext = New-AzureStorageContext -StorageAccountName $AzureStorageAccountname -StorageAccountKey $AzureStorageSAKey

Set-AzureRmCurrentStorageAccount -Context $AzureStorageContext

New-AzureStorageContainer -Name $AzureStorageContainerName1

Set-AzureStorageBlobContent -File $LocalScriptFile -container $AzureStorageContainerName1 -Force

Set-AzureRmVMCustomScriptExtension -Name 'Cloudcon-Installer' -ContainerName $AzureStorageContainerName1 -FileName $ScriptFile -StorageAccountName $AzureStorageAccountName -ResourceGroupName $AzureResourceGroupName -VMName $CloudConnectorMachineName1 -Run "installcloudcon1.ps1" -Location $AzureResourceGroupLocation

Start-Sleep -Seconds 10

Write-Host "Citrix Cloud Connector installation succesful, cleaning up..." -ForegroundColor Yellow

# Remove Extension and Script

Remove-AzureRmVMCustomScriptExtension -ResourceGroupName $AzureResourceGroupName -VMName $CloudConnectorMachineName1 -Name 'Cloudcon-Installer1' -Force

# Delete storage container

Remove-AzureStorageContainer -name $AzureStorageContainerName1 -Force

Write-Host "10. Citrix Cloud/Azure - Deploy Cloud Connector software on Server 2" -ForegroundColor Green

$AzureStorageContainerName2 = "cloudconinstaller2"

# Create Cloud Connector deployment script

## GRL This looks very similar to the previous definition! If you copy and paste more than a couple of lines, and more than once, you are doing something wrong/inefficently!

$DeployCloudConnectorScriptContent = "

`$CustomerID = `"$CustomerID`"

`$ClientID = `"$ClientID`"

`$ClientSecret = `"$ClientSecret`"

`$CTXCloudResourceID = `"$CTXCloudResourceID`"

`$Username = `"arnaudpain/ftp@arnaud.biz`"

`$Password = `"9$<rZK-k`"

`$LocalFile = `"C:\cwcconnector.exe`"

`$RemoteFile = `"ftp://arnaudpain.sharefileftp.com/automation/cwcconnector.exe`"

`$FTPRequest = [System.Net.FtpWebRequest]::Create(`$RemoteFile)

`$FTPRequest.Credentials = New-Object System.Net.NetworkCredential(`$Username,`$Password)

`$FTPRequest.Method = [System.Net.WebRequestMethods+Ftp]::DownloadFile

`$FTPRequest.UseBinary = `$true

`$FTPRequest.KeepAlive = `$false

`$FTPResponse = `$FTPRequest.GetResponse()

`$ResponseStream = `$FTPResponse.GetResponseStream()

`$LocalFileFile = New-Object IO.FileStream (`$LocalFile,[IO.FileMode]::Create)

`[byte[]]`$ReadBuffer = New-Object byte[] 1024

`do {

`$ReadLength = `$ResponseStream.Read(`$ReadBuffer,0,1024)

`$LocalFileFile.Write(`$ReadBuffer,0,`$ReadLength)

`}

`while (`$ReadLength -ne 0)

`$LocalFileFile.Close()

`$Arguments = `"/q /customername:`$CustomerID /clientid:`$ClientID /clientsecret:`$ClientSecret /location:`$CTXCloudResourceID /acceptTermsofservice:true`"

Start-Process `$LocalFile `$Arguments -Wait"

$ScriptFile = "InstallCloudCon2.ps1"

$LocalScriptFile = "$LocalTempFolder\$ScriptFile"

Set-Content -Path $LocalScriptFile -Value $DeployCloudConnectorScriptContent -Force

$TempScriptContent = Get-Content -Path $LocalTempFolder\$ScriptFile

$TempScriptContent = $TempScriptContent -Replace "\?", ""

Set-Content -Path $LocalScriptFile -Value $TempScriptContent -Force

# Upload Cloud Connector deployment script

$AzureStorageContext = New-AzureStorageContext -StorageAccountName $AzureStorageAccountname -StorageAccountKey $AzureStorageSAKey

Set-AzureRmCurrentStorageAccount -Context $AzureStorageContext

New-AzureStorageContainer -Name $AzureStorageContainerName2

Set-AzureStorageBlobContent -File $LocalScriptFile -container $AzureStorageContainerName2 -Force

Set-AzureRmVMCustomScriptExtension -Name 'Cloudcon-Installer' -ContainerName $AzureStorageContainerName2 -FileName $ScriptFile -StorageAccountName $AzureStorageAccountName -ResourceGroupName $AzureResourceGroupName -VMName $CloudConnectorMachineName2 -Run "installcloudcon2.ps1" -Location $AzureResourceGroupLocation

Start-Sleep -Seconds 10

Write-Host "Citrix Cloud Connector installation succesful, cleaning up..." -ForegroundColor Yellow

# Remove Extension and Script

Remove-AzureRmVMCustomScriptExtension -ResourceGroupName $AzureResourceGroupName -VMName $CloudConnectorMachineName2 -Name 'Cloudcon-Installer2' -Force

# Delete storage container

Remove-AzureStorageContainer -name $AzureStorageContainerName2 -Force

# Present timing

$ScriptStopWatch.Stop()

$ScriptRunningTime = [math]::Round($ScriptStopWatch.Elapsed.TotalMinutes,1)

Write-Host "Script ran for" $ScriptRunningTime "Minutes" -ForegroundColor Magenta

# END OF SCRIPT

## GRL can be useful to have a variable counting the number of errors and warnings so you can report at the end so if there is a lot of output, a user can look at the line reporting the error & warning counts and see if they need to investigate