

Sauvegarde de la configuration de Switch en auto

Table des matières

1.Archive.....	2
2.Script PowerShell pour déployer archive sur tous les Switchs.....	2
3.Script PowerShell pour trier les fichiers de conf (sur le server)	6
4.Planifier le script a l'aide du planificateur de tâche	8
1.Vérifier si le script PowerShell fonctionne seul.....	8
2.Autoriser l'exécution des scripts	8
3.Ouvrir le planificateur de tâches	8
4.Créer une nouvelle tâche	8
1.Créer une tâche	8
2.Onglet Général	8
3.Onglet Déclencheurs	8
4.Onglet Actions	9
5.Onglet Conditions	9
6.Onglet Paramètres	9
7.Tester.....	9

Pour réaliser ce travail, nous avons besoin de plusieurs outils, archive qui sera déployé sur les switchs via un script PowerShell en utilisant un csv, un script PowerShell de tri des versions qui s'exécutera via une tâche planifiée sur le serveur de stockage pour éviter un surnombre de copie des configurations.

1.Archive

La fonction archive va envoyer le fichier de configuration au serveur tftp à chaque write memory effectué sur le Switch

```
conf t
archive
path tftp://<ip du server>/$h$t-config
wr
End
wr
```

- \$h = nom du switch(hostname)
- \$t = horodatage du fichier de conf

2.Script PowerShell pour déployer archive sur tous les Switchs

```
<#
.SYNOPSIS
    Déploiement de configuration d'archive sur des switches Cisco via SSH
.DESCRIPTION
    Ce script lit un fichier CSV contenant les informations des switches et déploie
    la configuration d'archive via Posh-SSH
.PARAMETER CsvPath
    Chemin vers le fichier CSV contenant les informations des switches
.PARAMETER TftpServerIp
    Adresse IP du serveur TFTP
.EXAMPLE
    .\Deploy-SwitchArchive.ps1 -CsvPath "C:\switches.csv" -TftpServerIp "192.168.1.100"
#>

param(
    [Parameter(Mandatory=$true)]
    [string]$CsvPath,
    [Parameter(Mandatory=$true)]
    [string]$TftpServerIp,
    [Parameter(Mandatory=$false)]
    [int]$SshTimeout = 30
)

# Vérifier que Posh-SSH est installé
if (!([Get-Module -ListAvailable -Name Posh-SSH)) {
    Write-Host "Installation de Posh-SSH..." -ForegroundColor Yellow
    Install-Module -Name Posh-SSH -Force -Scope CurrentUser
}
```

```

Import-Module Posh-SSH

# Fonction pour exécuter les commandes sur un switch
function Deploy-ArchiveConfig {
    param(
        [string]$SwitchIP,
        [string]$Username,
        [string]$Password,
        [string]$TftpIP
    )

    try {
        Write-Host "`n[INFO] Connexion à $SwitchIP..." -ForegroundColor Cyan

        # Créer les credentials
        $SecPassword = ConvertTo-SecureString $Password -AsPlainText -Force
        $Credential = New-Object System.Management.Automation.PSCredential($Username, $SecPassword)

        # Établir la connexion SSH
        $Session = New-SSHSession -ComputerName $SwitchIP -Credential $Credential -AcceptKey -
        ConnectionTimeout $SshTimeout

        if ($Session.Connected) {
            Write-Host "[OK] Connecté à $SwitchIP" -ForegroundColor Green

            # Créer un stream SSH
            $Stream = New-SSHShellStream -SSHSession $Session
            Start-Sleep -Seconds 2

            # Lire le buffer initial
            $Stream.Read() | Out-Null

            # Commandes à exécuter
            $Commands = @(
                "conf t",
                "archive",
                "path tftp://$TftpIP/$h$t-config",
                "wr",
                "exit",
                "end",
                "wr"
            )

            # Exécuter chaque commande
            foreach ($Command in $Commands) {
                Write-Host " -> Exécution: $Command" -ForegroundColor Gray
                $Stream.WriteLine($Command)
                Start-Sleep -Milliseconds 500

                # Lire la sortie
                $Output = $Stream.Read()

                # Vérifier les erreurs
                if ($Output -match "Invalid input|Error|Failed") {
                    Write-Host "[ERREUR] Erreur lors de l'exécution de '$Command'" -ForegroundColor Red
                    Write-Host $Output -ForegroundColor Red
                }
            }
        }
    }
}

```

```

        }

# Attendre la fin de l'écriture
Start-Sleep -Seconds 3
$FinalOutput = $Stream.Read()

# Fermer le stream et la session
$Stream.Dispose()
Remove-SSHSession -SSHSession $Session | Out-Null

Write-Host "[OK] Configuration déployée avec succès sur $SwitchIP" -ForegroundColor Green
return $true
}
else {
    Write-Host "[ERREUR] Impossible de se connecter à $SwitchIP" -ForegroundColor Red
    return $false
}
}

catch {
    Write-Host "[ERREUR] Exception sur $SwitchIP : $($_.Exception.Message)" -ForegroundColor Red

# Nettoyer la session si elle existe
if ($Session) {
    Remove-SSHSession -SSHSession $Session -ErrorAction SilentlyContinue | Out-Null
}
return $false
}

# Programme principal
Write-Host "===== -ForegroundColor Cyan
Write-Host " Déploiement Configuration Archive" -ForegroundColor Cyan
Write-Host "===== -ForegroundColor Cyan

# Vérifier l'existence du fichier CSV
if (!(Test-Path $CsvPath)) {
    Write-Host "[ERREUR] Le fichier CSV '$CsvPath' n'existe pas!" -ForegroundColor Red
    exit 1
}

# Lire le fichier CSV
try {
    $Switches = Import-Csv -Path $CsvPath -Delimiter ','
    Write-Host "`n[INFO] $($Switches.Count) switch(es) trouvé(s) dans le CSV" -ForegroundColor Cyan
}
catch {
    Write-Host "[ERREUR] Impossible de lire le fichier CSV: $($_.Exception.Message)" -ForegroundColor Red
    exit 1
}

# Vérifier les colonnes requises
$RequiredColumns = @('IP', 'Username', 'Password')
$CsvColumns = $Switches[0].PSObject.Properties.Name

foreach ($Column in $RequiredColumns) {
    if ($Column -notin $CsvColumns) {

```

```

Write-Host "[ERREUR] Colonne '$Column' manquante dans le CSV!" -ForegroundColor Red
Write-Host "Colonnes requises: $($RequiredColumns -join ', ')" -ForegroundColor Yellow
exit 1
}

# Statistiques
$SuccessCount = 0
$FailCount = 0
$Results = @()

# Déployer sur chaque switch
foreach ($Switch in $Switches) {
    $Result = Deploy-ArchiveConfig -SwitchIP $Switch.IP `

        -Username $Switch.Username `

        -Password $Switch.Password `

        -TftpIP $TftpServerIp

    if ($Result) {
        $SuccessCount++
        $Status = "Succès"
    }
    else {
        $FailCount++
        $Status = "Échec"
    }

    $Results += [PSCustomObject]@{
        IP = $Switch.IP
        Username = $Switch.Username
        Status = $Status
        Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    }
}

# Résumé
Write-Host "`n======" -ForegroundColor Cyan
Write-Host "      RÉSUMÉ" -ForegroundColor Cyan
Write-Host "======" -ForegroundColor Cyan
Write-Host "Total: $($Switches.Count) | Succès: $SuccessCount | Échec: $FailCount" -ForegroundColor White

# Exporter les résultats
$LogPath = ".\Deployment-Log-$((Get-Date -Format 'yyyyMMdd-HHmmss')).csv"
$Results | Export-Csv -Path $LogPath -NoTypeInformation -Encoding UTF8
Write-Host "`n[INFO] Log exporté vers: $LogPath" -ForegroundColor Cyan

# Afficher les échecs s'il y en a
if ($FailCount -gt 0) {
    Write-Host "`nSwitches en échec:" -ForegroundColor Yellow
    $Results | Where-Object {$_ .Status -eq "Échec"} | Format-Table -AutoSize
}

```

Il faut installer Posh-SSH. Au moment de lancer le script, il faudra renseigner L'IP du TFTP et le chemin vers le fichier csv. (Csv au format **Name, IP,Username,Password**)

3. Script PowerShell pour trier les fichiers de conf (sur le server)

Ce script va permettre de ne pas garder trop de copies des configurations qui sont envoyés sur le serveur et de les renommer par heure d'arrivée. Dans cet exemple nous allons garder 3 copies de configuration par Switch.

```
$Dossier = "C:\Users\stagie\Documents\tftpd"
$Keep = 3

# Mois en anglais (comme ton exemple: Jan, Feb, Mar...)
$enUS = [System.Globalization.CultureInfo]::GetCultureInfo("en-US")

function Get-SwitchName([string]$filename) {
    if ($filename -match '^(<sw>[^-]+)-') { return $Matches.sw }
    return $null
}

# Déetecte si déjà au format voulu: sw13654-Jan-30-09-41-22.483-config-3
function Is-AlreadyInTargetFormat([string]$filename) {
    return ($filename -match '^[-]+[A-Za-z]{3}-\d{2}-\d{2}-\d{2}\.\d{3}-config-\d+$')
}

# Construit le nom cible à partir d'une date (heure locale Windows) + index 1..Keep
function Build-TargetName([string]$sw, [datetime]$dt, [int]$index) {
    $mon = $dt.ToString("MMM", $enUS)    # Jan
    $day = $dt.ToString("dd", $enUS)     # 30
    $hms = $dt.ToString("HH-mm-ss", $enUS) # 09-41-22
    $ms = $dt.ToString("fff", $enUS)      # 483
    return "$sw-$mon-$day-$hms.$ms-config-$index"
}

# 1) Renommage : on convertit les fichiers "bruts" (ou autres formats) en "sw-Mon-dd-HH-mm-ss.fff-config-X"
#   Ici on les renomme d'abord avec un index temporaire 0, puis on reindexera proprement à 1..Keep.
Get-ChildItem -Path $Dossier -File | ForEach-Object {
    $f = $_
    $sw = Get-SwitchName $f.Name
    if ($null -eq $sw) { return }

    if (Is-AlreadyInTargetFormat $f.BaseName) { return }

    $dt = $f.LastWriteTime # heure locale Windows
    $tempName = Build-TargetName $sw $dt 0 # index temporaire
    $newPath = Join-Path $Dossier $tempName

    # Evite collision si même milliseconde
    if (Test-Path -LiteralPath $newPath) {
        $i = 1
        do {
            $tempName2 = "$tempName-$(($i).ToString('00'))"
            $newPath2 = Join-Path $Dossier $tempName2
            $i++
        } while (Test-Path -LiteralPath $newPath2)
        $tempName = $tempName2
        $newPath = $newPath2
    }

    Rename-Item -LiteralPath $f.FullName -NewName $tempName
```

```

        Write-Host "RENOMME [$sw] $($f.Name) -> $tempName"
    }

# 2) Rétention + re-indexation config-1..config-$Keep (1 = plus récent)
#   On trie par LastWriteTime décroissant, on garde $Keep, et on renomme proprement avec l'index.
Get-ChildItem -Path $Dossier -File |
ForEach-Object {
    # On force un objet enrichi avec SwitchName
    $sw = Get-SwitchName $_.Name
    [PSCustomObject]@{ File = $_; Switch = $sw }
} |
Where-Object { $_.Switch -ne $null } |
Group-Object Switch |
ForEach-Object {

    $switch = $_.Name

    $sorted = $_.Group | Sort-Object { $_.File.LastWriteTime } -Descending

    # Ceux qu'on garde
    $keepFiles = $sorted | Select-Object -First $Keep

    # Re-indexe 1..$Keep
    for ($idx = 1; $idx -le $keepFiles.Count; $idx++) {
        $f = $keepFiles[$idx - 1].File
        $dt = $f.LastWriteTime
        $targetBase = Build-TargetName $switch $dt $idx

        if ($f.BaseName -ne $targetBase) {
            $targetPath = Join-Path $Dossier $targetBase

            # Evite collision si un fichier existe déjà avec ce nom
            if (Test-Path -LiteralPath $targetPath) {
                $j = 1
                do {
                    $targetBase2 = "$targetBase-$($j.ToString('00'))"
                    $targetPath2 = Join-Path $Dossier $targetBase2
                    $j++
                } while (Test-Path -LiteralPath $targetPath2)
                $targetBase = $targetBase2
                $targetPath = $targetPath2
            }
        }

        Rename-Item -LiteralPath $f.FullName -NewName $targetBase
        Write-Host "INDEXE [$switch] $($f.Name) -> $targetBase"
    }
}

# Ceux à supprimer au-delà de $Keep
$ToDelete = $sorted | Select-Object -Skip $Keep
foreach ($x in $ToDelete) {
    Remove-Item -LiteralPath $x.File.FullName -Force
    Write-Host "SUPPRIME [$switch] $($x.File.Name)"
}
}

```

4. Planifier le script à l'aide du planificateur de tâche

1. Vérifier si le script PowerShell fonctionne seul

Ouvrir PowerShell et tester :

```
cd « C:\chemin\vers\le\script »  
.\\le_script.ps1
```

Si ça fonctionne → c'est parfait
Sinon la prochaine étape permet de corriger cela

2. Autoriser l'exécution des scripts

Set-ExecutionPolicy Unrestricted

Il faut répondre O(oui)

3. Ouvrir le planificateur de tâches

Win+R

Saisir : taskschd.msc

4. Créer une nouvelle tâche

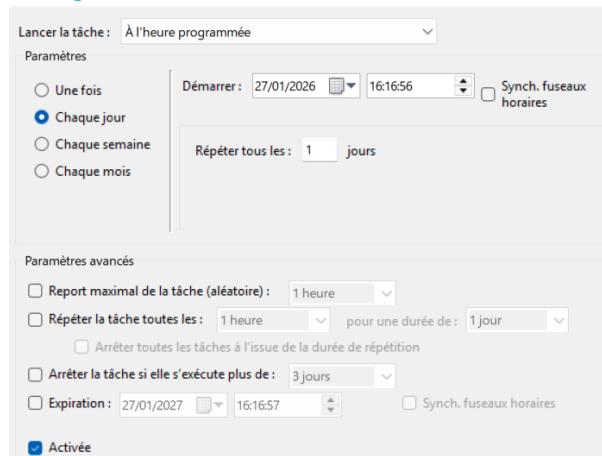
1. Créer une tâche

- **Créer une tâche** (pas "tâche de base")

2. Onglet Général

- Nom : Nettoyage backups Cisco
- Cocher :
 - Exécuter que l'utilisateur soit connecté ou non
 - Exécuter avec les autorisations maximales

3. Onglet Déclencheurs



A vous de choisir la date et l'heure

4.Onglet Actions

- **Nouvelle action**
 - Programme/script : powershell.exe
 - Ajouter des arguments :
 - -ExecutionPolicy Bypass -File "C:\chemin\vers\le\script\nomdu_script.ps1"
 - Démarrer dans : C:\ chemin\vers\le\script
-

5.Onglet Conditions

- décocher *Démarrer la tâche uniquement si l'ordinateur est sur secteur*
-

6.Onglet Paramètres

- Cocher :
 - Autoriser l'exécution de la tâche à la demande
 - Exécuter la tâche dès que possible après un démarrage planifié manqué
 - Décocher :
 - Arrêter la tâche si elle s'exécute plus de X heures (important)
-

7.Tester

- Clic droit sur la tâche → **Exécuter**