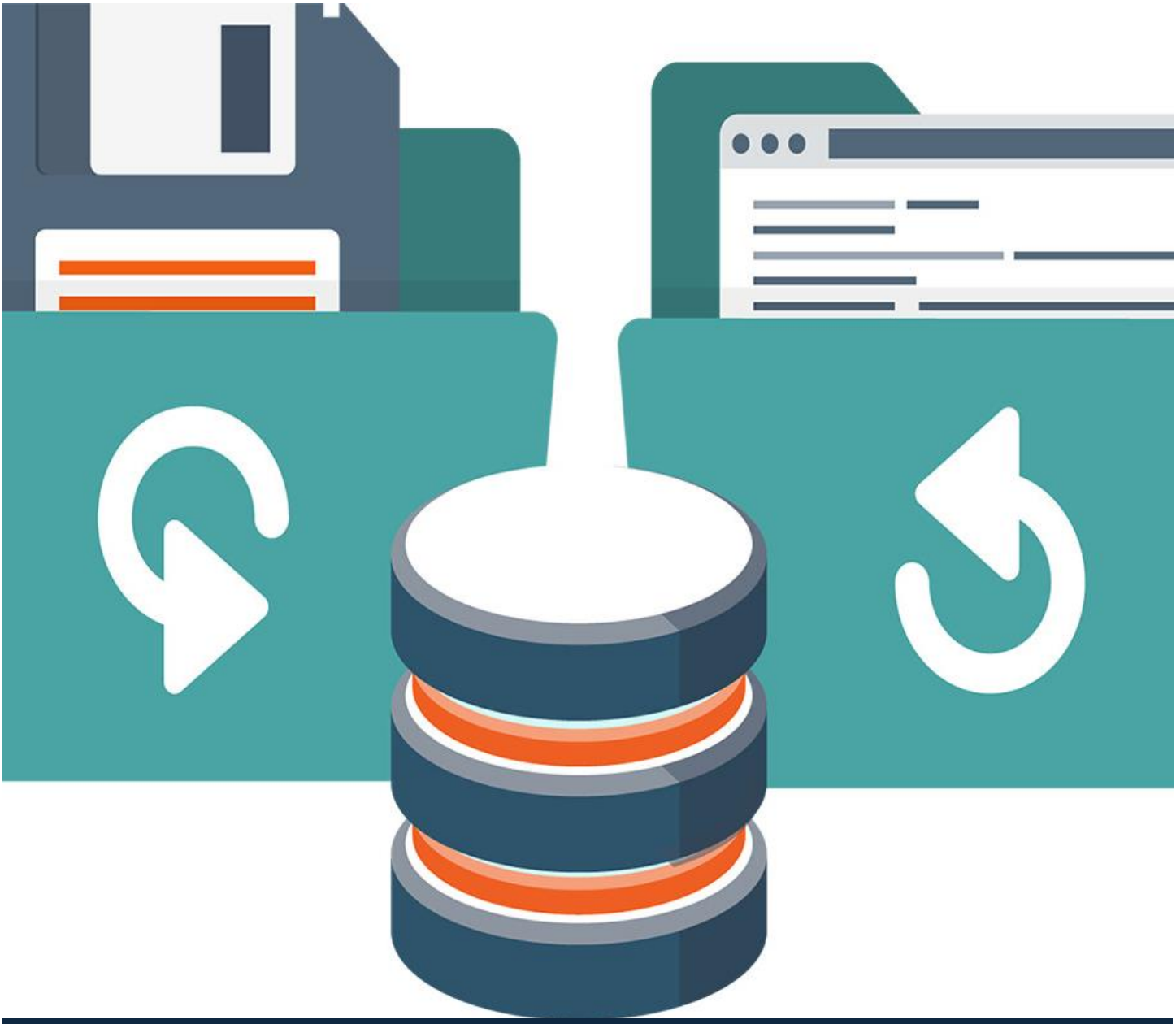


Documentation

Sauvegarder des dossiers et fichiers à l'aide du script

BET--TAFFARD Océane

29/11/24

Version 2

TABLE DES MATIERES

INTRODUCTION

Afin de pouvoir sauvegarder des dossiers et fichiers qui n'ont pas été modifiés depuis un certain temps, un script a été créé.

Dans ce script a été défini le seuil au bout duquel on souhaite que les fichiers non modifiés soient sauvegardés. Lorsqu'il s'exécute, le script regarde la date de la dernière modification des dossiers/fichiers et sauvegarde ceux dépassant le seuil en conservant les droits d'accès (les administrateurs gardent le contrôle total mais les utilisateurs passent en lecture et exécution seulement).

EXPLICATIONS

Fonctionnement du script de sauvegarde

Le script de sauvegarde a plusieurs fonctions : la copie des fichiers dépassant le seuil défini dans un serveur de sauvegarde, la conservation des droits d'accès et la restriction des droits des utilisateurs à lecture et exécution seulement.

Étant donné que ce script peut être emmené à être utilisé dans plusieurs environnements différents, il y a quelques parties de son code qui peuvent nécessiter d'être modifiées :

```
param(
    [string]$sourceFolder = "$env:SystemDrive\Donnees", # Chemin vers le dossier source
    [string]$backupFolder = "\\192.168.1.202\Sauvegardes", # Chemin vers le dossier de sauvegarde
    [string]$logFile = "C:\Logs\backup_log.txt", # Chemin vers le fichier de log
    [string]$errorLogFile = "C:\Logs\backup_error_log.txt", # Chemin vers le fichier de log des erreurs
    [int]$daysThreshold = 700 # Seuil en jours
)
```

Le premier paramètre « $env:SystemDrive\Donnees » correspond au chemin vers le dossier source, c'est-à-dire le dossier à partir duquel les dossiers sont récupérés pour être sauvegardés. Il faudra donc remplacer ce chemin par le vôtre (en le mettant bien entre guillemets).

Le deuxième paramètre « \\192.168.1.202\Sauvegardes » correspond au chemin du dossier de sauvegarde, donc le dossier dans lequel seront sauvegardés les dossiers récupérés dans le dossier source. De même, il faudra remplacer ce chemin par le vôtre.

Le troisième paramètre « C:\Logs\backup_log.txt » correspond au chemin du fichier de logs (fichier dans lequel est écrit tout ce qui passe lors de l'exécution du script). Il ne nécessite pas obligatoirement d'être modifié étant donné que le script se charge lui-même de créer le fichier mais vous pouvez quand même le faire si vous souhaitez qu'il s'enregistre ailleurs.

Le quatrième paramètre « C:\Logs\backup_error_log.txt» correspond au chemin du fichier de log d'erreurs (fichier dans lequel sont recensées toutes les erreurs qu'il a pu y avoir lors de l'exécution du script). De même que pour le précédent, il ne nécessite pas obligatoirement d'être changé mais vous pouvez quand même le faire si vous souhaitez qu'il s'enregistre ailleurs.

Le dernier paramètre « $daysThreshold » correspond au seuil au bout duquel les dossiers et fichiers non modifiés doivent être sauvegardés. Pour fixer un autre seuil il suffit simplement de changer le « 700 ».

```powershell
# Créer le dossier de logs s'il n'existe pas
$logPath = Split-Path -Parent $logFile
if (-not (Test-Path $logPath)) {
    New-Item -ItemType Directory -Path $logPath -Force
}
```

Ce morceau de code vérifie si le dossier de Logs existe et, si tel n'est pas le cas, le créé.

```powershell
# Fichiers de log pour cette exécution
$timestamp = Get-Date -Format "yyyy-MM-dd-HH-mm"
$executionLogFile = "C:\Logs\backup_log_$timestamp.txt"
$executionErrorLogFile = "C:\Logs\backup_error_log_$timestamp.txt"
```

Cette partie du code sert à récupérer la date et l'heure actuelles, à définir les chemins des fichiers et à créer leurs noms en y incluant la date et l'heure.

```powershell
# Fonction de log
function Write-Log {
    param($Message, [switch]$IsError)
    $logMessage = "$(Get-Date -Format 'dd-MM-yyyy HH:mm:ss'): $Message"
    Add-Content -Path $executionLogFile -Value $logMessage
    Write-Host $logMessage
    if ($IsError) {
        Add-Content -Path $executionErrorLogFile -Value $logMessage
    }
}
```

Cette fonction sert à écrire les messages de logs, avec la date et l'heure, dans les deux fichiers de logs.

```powershell
# Fonction pour obtenir les permissions d'un chemin
function Get-PathPermissions {
    param ([string]$Path)
    $permissions = @{}
    try {
        $acl = Get-Acl -Path $Path
        if ($null -ne $acl) {
            foreach ($access in $acl.Access) {
                $identity = $access.IdentityReference.Value
                if (!$permissions.ContainsKey($identity)) {
                    $permissions[$identity] = $access.FileSystemRights
                } else {
                    $permissions[$identity] = $permissions[$identity] -bor $access.FileSystemRights
                }
            }
        }
        return $permissions
    } catch {
        Write-Log "Erreur lors de la récupération des permissions pour $Path : $_" -IsError
        throw
    }
}
```

Cette fonction permet de récupérer les permissions assignées aux dossiers et fichiers afin de les préserver lors du déplacement vers le serveur de sauvegarde.

```powershell
# Fonction pour vérifier si un utilisateur est administrateur
function Is-Admin {
    param ([string]$Identity)
    return $Identity -match "Administrateurs|Administrateur|Administrators|Système|Domain Admins"
}
```

Cette fonction sert à vérifier si l'utilisateur est un administrateur afin de, si tel est le cas, ne pas lui enlever le contrôle total sur les dossiers et fichiers sauvegardés. Elle peut nécessiter d'être modifiée ou complétée si vous avez un administrateur spécifique qui n'apparaît pas dans cette liste. Dans ce cas-là il faut séparer chaque nom d'administrateur par « | ».

```powershell
# Fonction pour appliquer les permissions sur un dossier
function Set-FolderPermissions {
    param ([string]$Path, [hashtable]$Permissions)
    try {
        $acl = Get-Acl -Path $Path
        if ($null -ne $acl) {
            $acl.SetAccessRuleProtection($true, $false)
            $acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) | Out-Null }
            foreach ($identity in $Permissions.Keys) {
                $rights = $Permissions[$identity]
                if (-not (Is-Admin -Identity $identity)) {
                    $rights = [System.Security.AccessControl.FileSystemRights]::ReadAndExecute
                }
                $rule = New-Object System.Security.AccessControl.FileSystemAccessRule(
                    $identity,
                    $rights,
                    "ContainerInherit,ObjectInherit",
                    "None",
                    [System.Security.AccessControl.AccessControlType]::Allow
                )
                $acl.AddAccessRule($rule)
            }
            Set-Acl -Path $Path -AclObject $acl
            Write-Log "Permissions appliquées sur le dossier : $Path"
        }
    } catch {
        Write-Log "Erreur lors de l'application des permissions sur le dossier : $_" -IsError
        throw
    }
}
```

Cette fonction attribue les droits d'accès à chaque dossier lorsqu'ils sont copiés en vérifiant si l'utilisateur à qui il faut mettre les permissions est un administrateur ou un simple utilisateur.

```powershell
# Fonction pour appliquer les permissions sur un fichier
function Set-FilePermissions {
    param ([string]$Path, [hashtable]$Permissions)
    try {
        $acl = Get-Acl -Path $Path
        if ($null -ne $acl) {
            $acl.SetAccessRuleProtection($true, $false)
            $acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) | Out-Null }
            foreach ($identity in $Permissions.Keys) {
                $rights = $Permissions[$identity]
                if (-not (Is-Admin -Identity $identity)) {
                    $rights = [System.Security.AccessControl.FileSystemRights]::ReadAndExecute
                }
                $rule = New-Object System.Security.AccessControl.FileSystemAccessRule(
                    $identity,
                    $rights,
                    "None",
                    "None",
                    [System.Security.AccessControl.AccessControlType]::Allow
                )
                $acl.AddAccessRule($rule)
            }
            Set-Acl -Path $Path -AclObject $acl
            try {
                Set-ItemProperty -Path $Path -Name IsReadOnly -Value $true
                Write-Log "Propriété IsReadOnly appliquée : $Path"
            } catch {
                Write-Log "Erreur lors de la modification de la propriété IsReadOnly pour le fichier : $Path - $_" -IsError
            }
            Write-Log "Permissions appliquées sur le fichier : $Path"
        }
    } catch {
        Write-Log "Erreur lors de l'application des permissions sur le fichier : $_" -IsError
        throw
    }
}
```

Cette fonction fait la même chose que la précédente mais cette fois sur les fichiers. Elle vérifie si l'identité à qui il faut assigner les droits est un utilisateur ou un administrateur et les applique en fonction.

```powershell
# Fonction pour fusionner les permissions de deux hashtables
function Merge-Permissions {
    param ([hashtable]$Permissions1, [hashtable]$Permissions2)
    $mergedPermissions = @{}
    $allIdentities = @($Permissions1.Keys) + @($Permissions2.Keys) | Select-Object -Unique
    foreach ($identity in $allIdentities) {
        if ($Permissions1.ContainsKey($identity) -and $Permissions2.ContainsKey($identity)) {
            $mergedPermissions[$identity] = $Permissions1[$identity] -bor $Permissions2[$identity]
        } elseif ($Permissions1.ContainsKey($identity)) {
            $mergedPermissions[$identity] = $Permissions1[$identity]
        } else {
            $mergedPermissions[$identity] = $Permissions2[$identity]
        }
    }
    return $mergedPermissions
}
```

Cette fonction est utilisée pour combiner les permissions du fichier source et du fichier de destination lors de la copie, assurant que tous les droits originaux sont préservés.

```powershell
# Fonction pour la sauvegarde des fichiers
function Start-FileBackUp {
    Write-Log "Début du processus de sauvegarde"

    # Vérification du dossier source
    if (-not (Test-Path $sourceFolder)) {
        Write-Log "Erreur: Dossier source inaccessible ou inexistant: $sourceFolder" -IsError
        return
    }
    Write-Log "Dossier source trouvé: $sourceFolder"

    #Vérification/création du dossier de sauvegarde
    if (-not (Test-Path $backupFolder)) {
        try {
            New-Item -ItemType Directory -Path $backupFolder -Force | Out-Null
            Write-Log "Dossier de sauvegarde créé : $backupFolder"
        } catch {
            Write-Log "Erreur lors de la création du dossier de sauvegarde: $backupFolder - $_" -IsError
            return
        }
    }
    try {
        # Récupération de TOUS les fichiers dans le dossier source
        $filesToMove = Get-ChildItem -Path $sourceFolder -Recurse -File -Force |
            Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-$daysThreshold) }

        $totalFiles = $filesToMove.Count
        Write-Log "Nombre total de fichiers à traiter : $totalFiles"

        $processedFiles = 0
        foreach ($file in $filesToMove) {
            $processedFiles++
            Write-Log "[$processedFiles/$totalFiles] Traitement du fichier: $($file.FullName)"

            # Calcul du chemin relatif pour préserver la structure
            $relativePath = $file.FullName.Substring($sourceFolder.Length)
            $destinationPath = Join-Path $backupFolder $relativePath
            $destinationDir = Split-Path -Parent $destinationPath

            try {
                $sourcePermissions = Get-PathPermissions -Path $file.FullName
                if ($null -ne $sourcePermissions) {
                    if (-not (Test-Path $destinationDir)) {
                        New-Item -ItemType Directory -Path $destinationDir -Force | Out-Null
                        Write-Log "Dossier de destination créé: $destinationDir"
                    }

                    $existingPermissions = @{}
                    if (Test-Path $destinationPath) {
                        $existingPermissions = Get-PathPermissions -Path $destinationPath
                    }

                    $mergedPermissions = Merge-Permissions -Permissions1 $sourcePermissions -Permissions2 $existingPermissions

                    # Application des permissions sur le dossier de destination
                    Set-FolderPermissions -Path $destinationDir -Permissions $mergedPermissions

                    # Avant de copier, désactivez l'attribut readonly
                    $file | Clear-ItemProperty -Name Attributes -Force

                    # Copie du fichier
                    Copy-Item -Path $file.FullName -Destination $destinationPath -Force
                    Write-Log "Fichier copié: $destinationPath"

                    # Application des permissions sur le fichier
                    Set-FilePermissions -Path $destinationPath -Permissions $mergedPermissions

                } else {
                    Write-Log "Aucune permission trouvée pour le fichier: $($file.FullName)" -IsError
                }
            } catch {
                Write-Log "Erreur lors du traitement du fichier: $_" -IsError
            }
        }

        Write-Log "Traitement terminé. $processedFiles fichiers traités sur $totalFiles"
    } catch {
        Write-Log "Erreur lors du processus de sauvegarde: $_" -IsError
    }
}
```

Cette fonction sert à sauvegarder les fichiers, pour cela elle procède en plusieurs étapes :

•Elle copie les fichiers dans un dossier de sauvegarde en préservant leur structure originale ;

• Gère et conserve les permissions des fichiers ;

• Enregistre et trace toutes les étapes du processus.

```powershell
# Exécution du script
Write-Log "=== Démarrage du script ==="
Write-Log "Version PowerShell: $($PSVersionTable.PSVersion)"
Write-Log "Utilisateur actuel: $([System.Security.Principal.WindowsIdentity]::GetCurrent().Name)"
Write-Log "Dossier source configuré: $sourceFolder"
Write-Log "Dossier de sauvegarde configuré: $backupFolder"
Write-Log "Seuil de sauvegarde: $daysThreshold jours"
Start-FileBackUp
Write-Log "=== Fin du script ==="
```

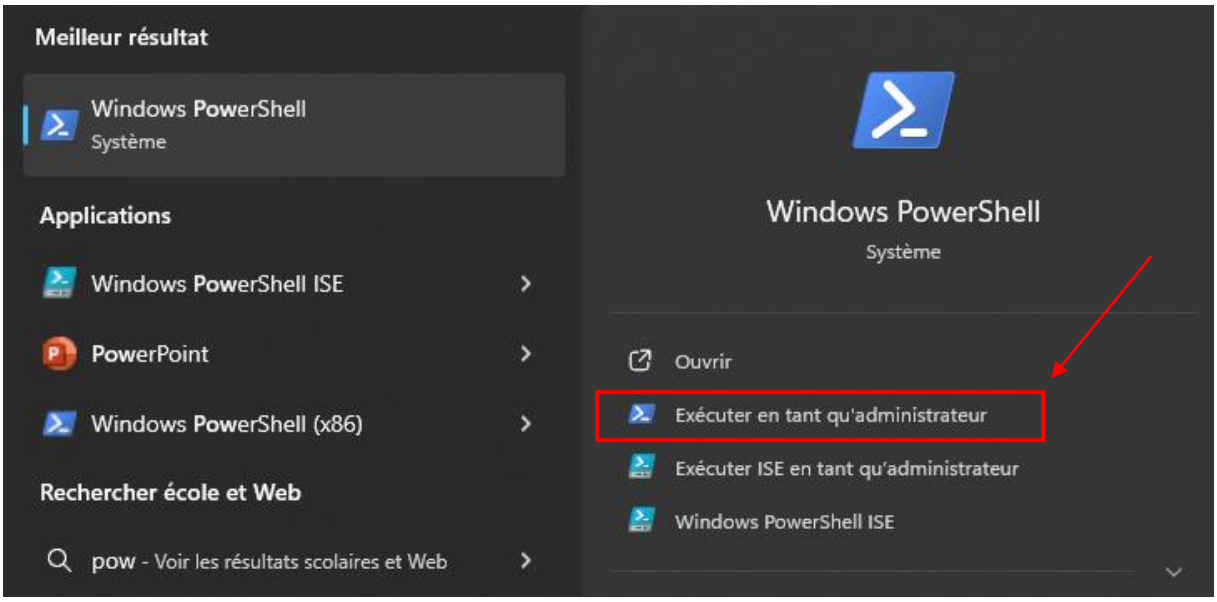
Ce dernier morceau de code sert à afficher certaines informations avant l'exécution du script tel que la version PowerShell utilisée, l'utilisateur qui exécute le script ainsi que les configurations du dossier source, du dossier de sauvegarde et du seuil de sauvegarde. Il lance ensuite le processus de sauvegarde tout en enregistrant les détails de l'exécution.

EXECUTER LE SCRIPT

Pré-requis :

- Être connecté en tant qu'administrateur
- Avoir PowerShell d'installer sur son ordinateur

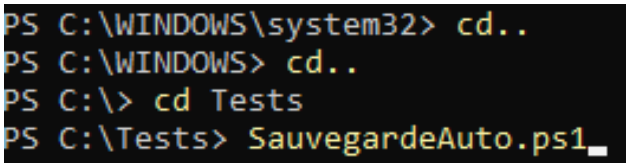
Pour exécuter le script de sauvegarde il faut d'abord ouvrir PowerShell en Administrateur.



Pour lancer le script il y a deux méthodes :

Méthode n°1 :

Dans PowerShell, se déplacer dans le dossier contenant le script puis l'exécuter.



Méthode n°2 :

Toujours dans PowerShell, Spécifier le chemin complet du script pour l'exécuter.

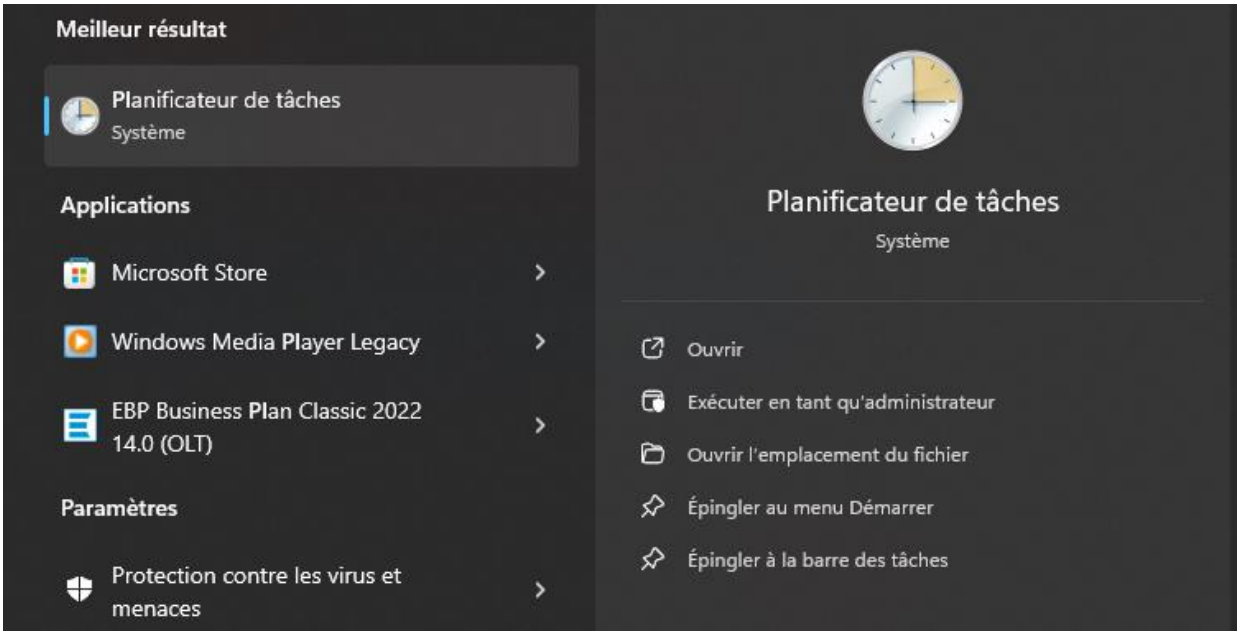
/!\ Important /!

Lorsque vous essaierez d'exécuter le script, il est possible que celui-ci ne le veuille pas parce que vous avez interdit l'exécution des scripts. Pour cela il faut exécuter la commande « Set-ExecutionPolicy RemoteSigned » et cliquer sur « oui » lorsque la notification s'affiche.

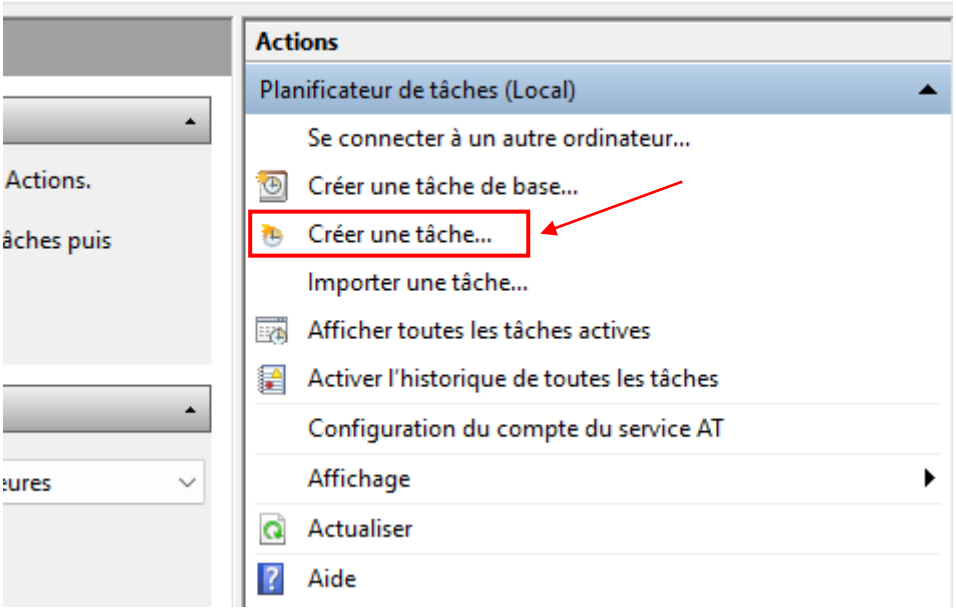
CREER LA TACHE D'AUTOMATISATION

Pour automatiser l'exécution du script de sauvegarde il faut créer une tâche dans le planificateur de tâche de Windows.

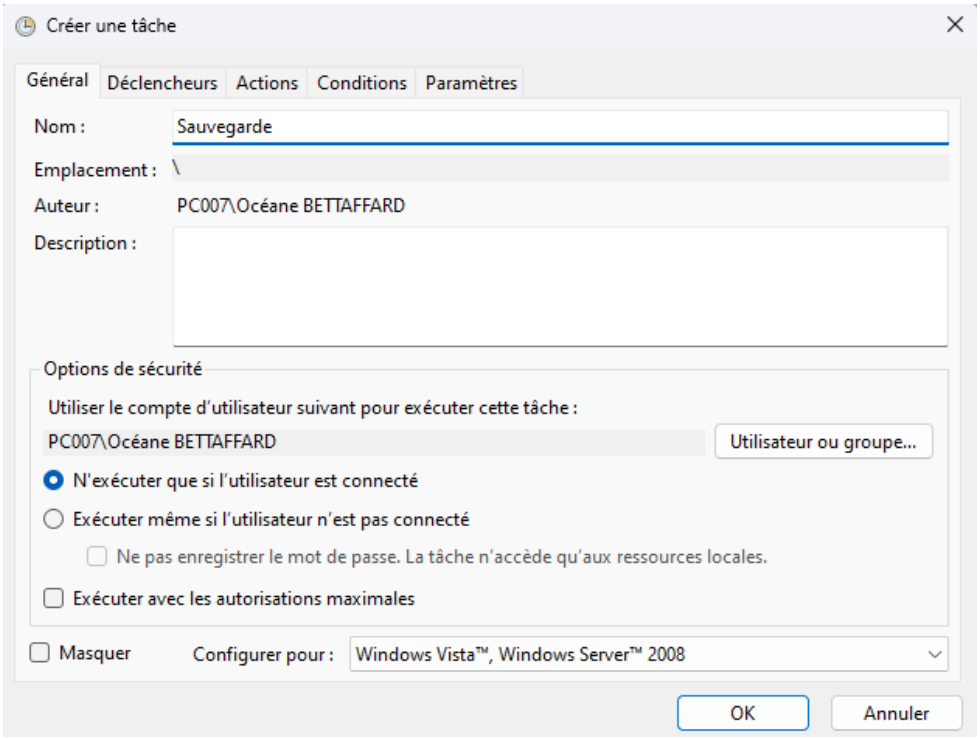
Tout d'abord, cherchez et ouvrez le planificateur de tâche (Task Scheduler si votre ordinateur est en anglais).



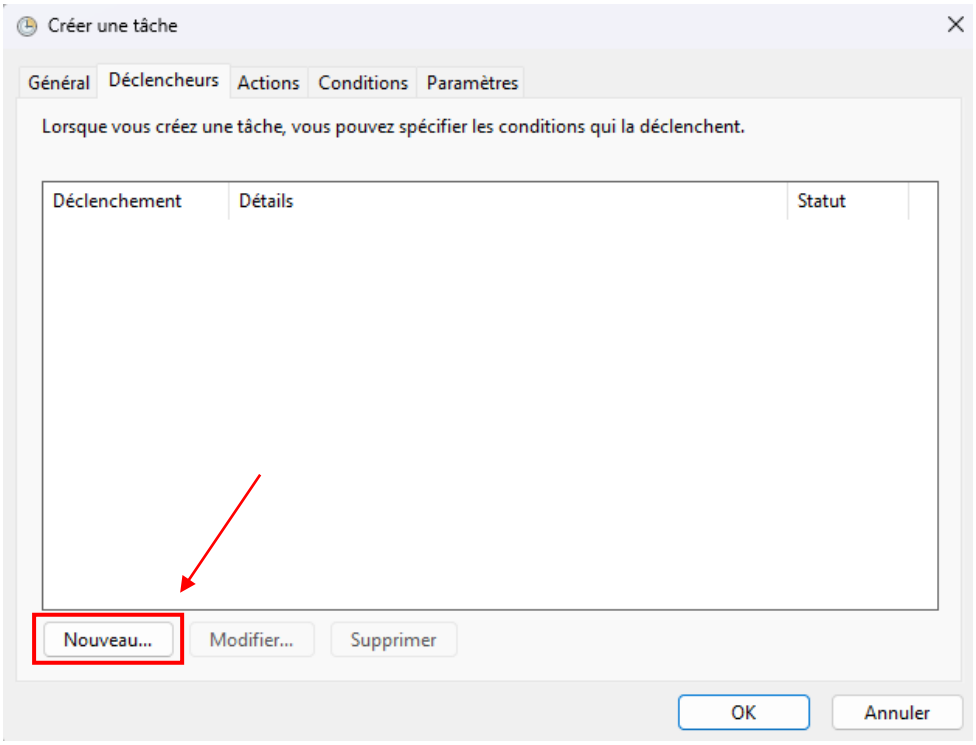
Une fois ouvert, cliquez sur « Créer une tâche » dans le menu de droite.



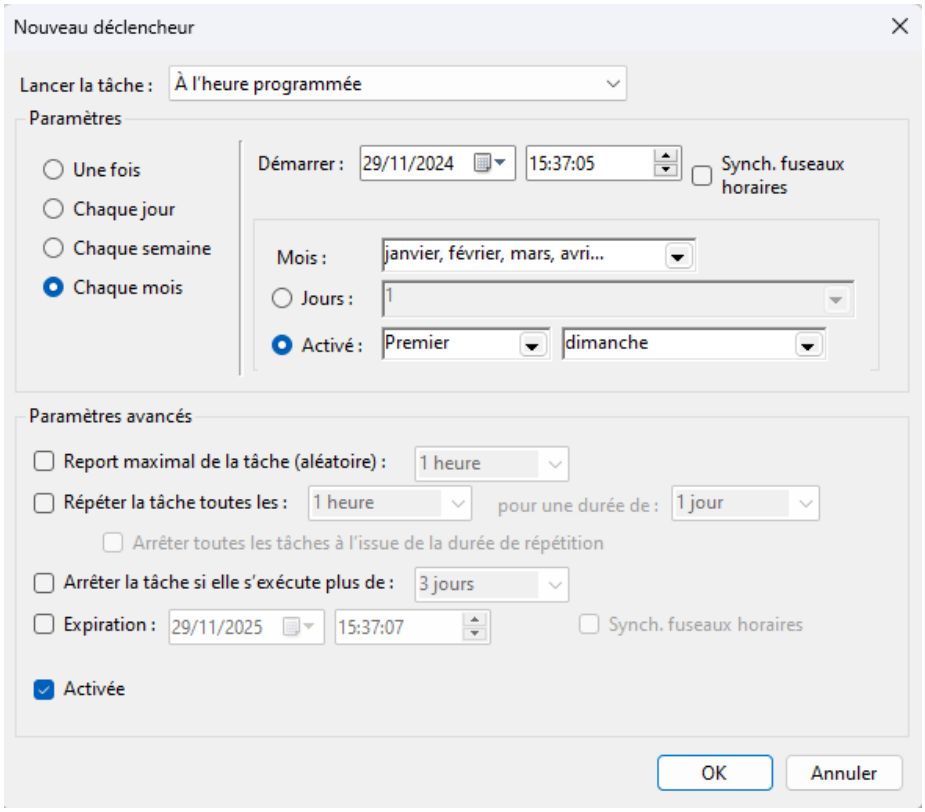
Donnez le nom que vous souhaitez à la tâche.



Ensuite allez dans l'onglet « Déclencheurs » et sélectionnez « Nouveau ».

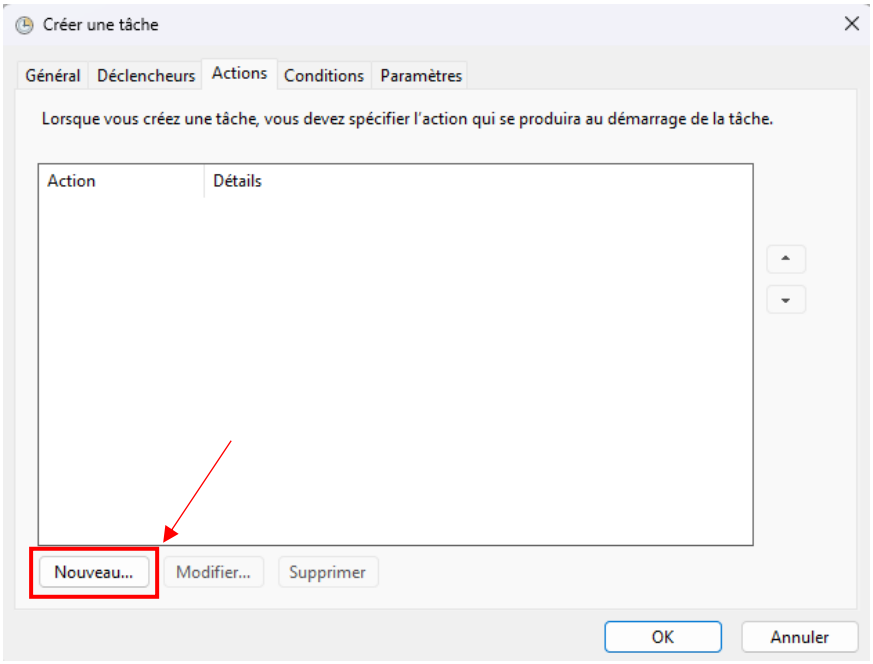


Paramétrez le déclencheur pour que le tâche s'active et se répète chaque fois que vous le voulez.

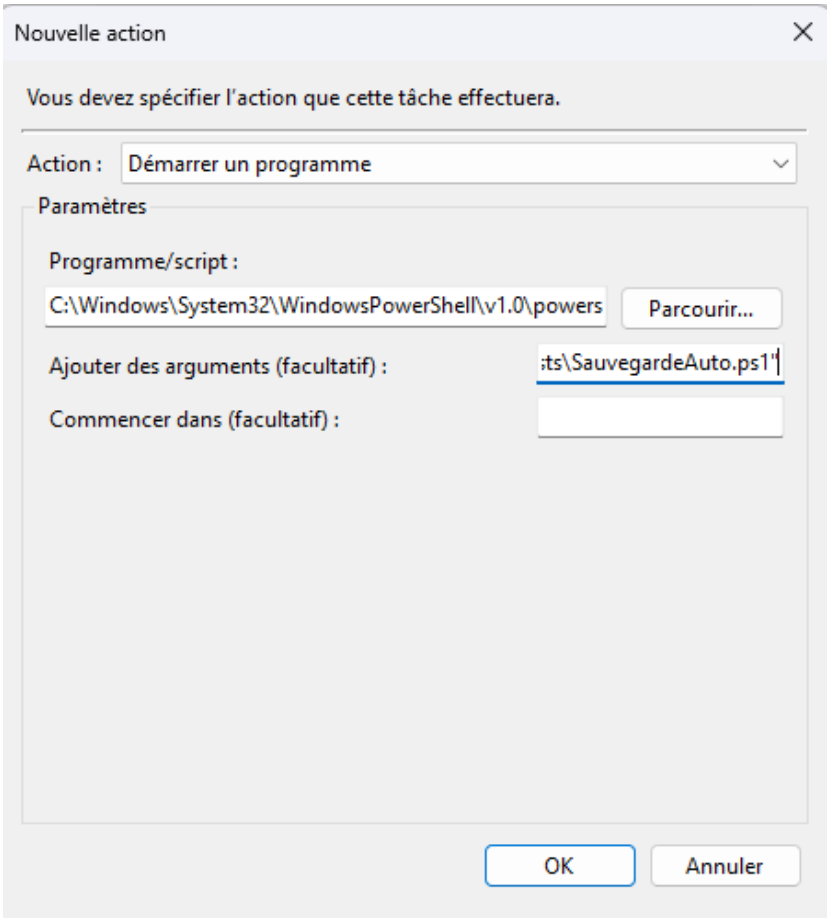


Puis cliquez sur « Ok ».

Déplacez-vous dans l'onglet « Actions » et sélectionnez « Nouveau ».



Dans le champ « Programme/script » mettez le chemin menant vers powershell.exe et dans le champ « Ajouter des arguments » le chemin vers le script de sauvegarde.



Sélectionnez « Ok » puis validez la création de la tâche en resélectionnant « Ok ».

Vous pouvez vérifier que la tâche a bien été créée en vous rendant dans l'onglet « Bibliothèque du planificateur de tâches » en haut à gauche.