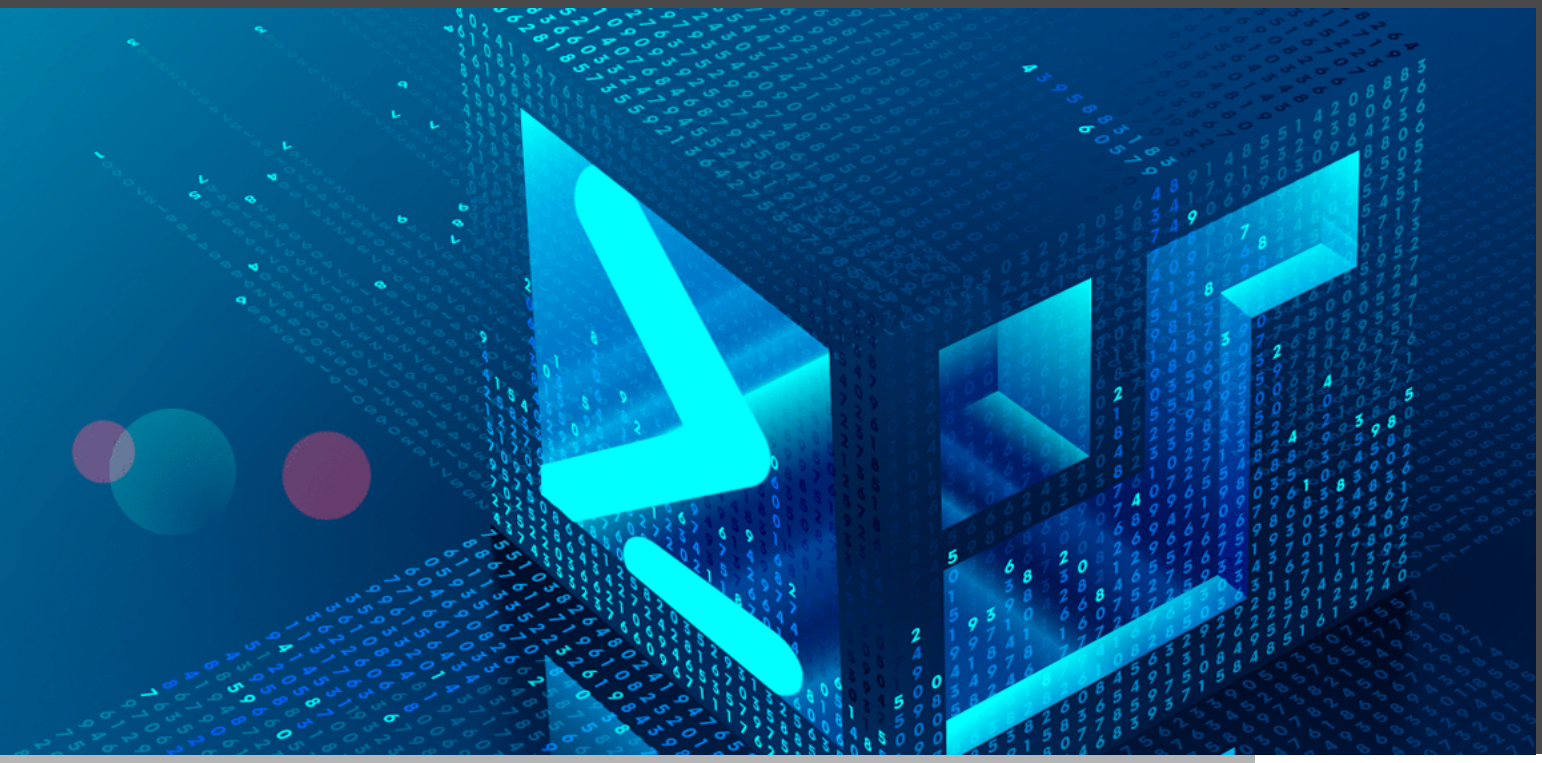
Devolutions

Référentiel local de modules PowerShell, sans serveur



C'EST ENCORE MIEUX SI ON LES COMBINE AVEC LES MODULES POWERSHELL

Les scripts PowerShell, ça fonctionne bien en soi, mais c'est encore mieux si on les combine avec les modules PowerShell. Avec PSGallery, le plus grand référentiel PowerShell, on peut trouver des modules pour faire à peu près n'importe quoi. Bien que les modules publiés soient assez faciles à installer et à importer, c'est loin d'être le cas pour les modules locaux non publiés. C'est même un défi quotidien quand on développe des modules PowerShell, mais il y a de l'espoir!

Choisir un module PowerShell de référence

Dans ce cas-ci, nous avons choisi le module Devolutions.Hub à titre de référence. Vous pouvez suivre ces étapes même si vous utilisez un nom et une version de module différents. Par contre, PowerShell 7 sera nécessaire pour charger ce module multiplateforme, donc Windows PowerShell 5.1 (intégré) ne pourra pas être utilisé. Pour plus de détails, vous pouvez vous référer au guide d'installation officiel de PowerShell ou utiliser cette commande sous Windows :

```
iex «& { $(irm https://aka.ms/install-powershell.ps1) } -UseMSI -Quiet»
```

Une fois PowerShell 7 (pwsh.exe) installé, assurez-vous qu'il soit toujours utilisé à la place de l'ancien Windows PowerShell (powershell.exe). Ça vous évitera bien des maux de tête!

Importer des modules par nom ou par chemin

Les modules publiés peuvent être installés avec la commande **Install-Module**, après quoi ils pourront être importés avec la commande **Import-Module** en utilisant le nom du module :

```
Install-Module -Name 'Devolutions.Hub'
Import-Module -Name 'Devolutions.Hub'
```

Les modules pourront être chargés en utilisant le chemin d'accès au fichier .psm1 :

```
$ModuleBase = `
Get-Module -Name Devolutions.Hub -ListAvailable | `
Select-Object -First 1 -ExpandProperty ModuleBase
Import-Module «$ModuleBase\Devolutions.Hub.psm1»
```

S'il porte le même nom que le fichier .psm1 qu'il contient, vous pourrez charger les modules en utilisant le chemin d'accès du répertoire du module. Étant donné que la commande **Install-Module** vient créer un sous-répertoire

selon la version du module (“Devolutions.Hub\2021.1.0\Devolutions.Hub.psm1”), vous devrez faire une copie temporaire pour obtenir la structure désirée (“Devolutions.Hub\Devolutions.Hub.psm1”) :

```
$TempPath = [System.IO.Path]::GetTempPath()
$TempModulePath = Join-Path $TempPath «Devolutions.Hub»
Copy-Item $ModuleBase $TempModulePath -Recurse
Import-Module $TempModulePath
```

Une fois importé, il n’y a aucune différence. L’importation par nom s’apparente au lancement d’un programme ayant été installé globalement avec un installateur .msi, tandis que l’importation par chemin équivaut au lancement d’un .exe utilisant son chemin complet. Au final, ce n’est pas SI important. Par contre, ça pourrait devenir embêtant si les scripts sont en attente d’installation de modules.

Enregistrer un référentiel local

Pour régler le problème d’installation du module, vous pouvez créer un référentiel PowerShell sur le système local de fichiers. Au lieu d’importer des modules par chemin, les modules peuvent être publiés et installés localement, sans devoir passer par PSGallery, un serveur NuGet ou encore un partage réseau. Et oui : peu de gens le savent, mais contrairement à la croyance populaire, la commande **Register-PSRepository** accepte les chemins locaux sur le système de fichiers, et pas seulement les dossiers partagés sur le réseau!

Créez le répertoire “~/psrepo” puis nommez-le **Register-PSRepository** pour créer le référentiel ‘local’ :

```
$RepoPath = «~/psrepo»
New-Item -Path $RepoPath -ItemType ‘Directory’ -Force | Out-Null
Register-PSRepository -Name ‘local’ `
   -SourceLocation «$(Resolve-Path $RepoPath)» `
   -PublishLocation «$(Resolve-Path $RepoPath)» `
   -InstallationPolicy ‘Trusted’
```

Vérifiez que le nouveau référentiel ait été correctement créé avec la commande **Get-PSRepository** :

```
PS > Get-PSRepository
```

```
Name                InstallationPolicy   SourceLocation
----                ------------------   --------------
local               Trusted              C:\Users\wayk\psrepo
PSGallery           Untrusted            https://www.powershellgallery.com/api/v2
```

Par défaut, PSGallery ne sera pas reconnu comme étant fiable, parce que son contenu n'est pas organisé. Dans ce cas-ci, nous allons marquer le référentiel local comme étant fiable pour la simple raison qu'il est local, donc non contrôlé par quelqu'un d'autre.

Enregistrer un module PowerShell

Pour éviter d'avoir à créer un module PowerShell à partir de zéro, prenons-en un qui se trouve sur PSGallery. Pour récupérer un module et le copier dans un chemin local sans l'installer, vous pouvez utiliser la commande **Save-Module**. N'importe quel module peut être utilisé tant et aussi longtemps qu'il est désinstallé avant de suivre les instructions. Inutile de s'inquiéter : le module pourra ensuite être réinstallé proprement. Voici un exemple pour mieux comprendre le principe :

```
$ModuleName = 'Devolutions.Hub'
$ModuleVersion = '2021.1.0'
New-Item -Path «~/modules» -ItemType 'Directory' -Force | Out-Null
Save-Module -Name $ModuleName -RequiredVersion $ModuleVersion `
  -Repository 'PSGallery' -Path «~/modules»
$ModulePath = «~/modules/${ModuleName}/${ModuleVersion}»
```

Les lignes de code ci-dessus enregistrent le module Devolutions.Hub de PSGallery dans "~/modules/Devolutions.Hub/2021.1.0". Ce répertoire doit contenir le fichier manifeste du module (.psd1).

Charger les modules depuis PSModulePath

Comment PowerShell arrive-t-il à trouver les modules quand ils sont importés par nom? En fait, il utilise PSModulePath, une variable d'environnement qui comprend une liste des répertoires à examiner. Son fonctionnement est très semblable à la variable d'environnement PATH qui contrôle les répertoires dans lesquels il faut rechercher les fichiers système exécutables. En d'autres mots, PSModulePath utilise le même caractère de séparation que PATH: ';' sur Windows, ':' sur des non-Windows.

En ajoutant "~/modules" à PSModulePath, on vient rendre les modules accessibles pour PowerShell. Pour éviter toute interférence avec un module précédemment installé, vous devez vous assurer que le module d'exemple est correctement désinstallé :

```
Uninstall-Module -Name $ModuleName -AllVersions
Get-Module -Name $ModuleName -ListAvailable -Refresh
```

Ensuite, modifiez temporairement PSModulePath, vérifiez que le module devient disponible, puis rétablissez PSModulePath à sa valeur originale :

```
$PSModulePath = $Env:PSModulePath
$Env:PSModulePath += «$([IO.Path]::PathSeparator)$(Resolve-Path «~/modules»)»
Get-Module -Name $ModuleName -ListAvailable -Refresh
$Env:PSModulePath = $PSModulePath
```

La commande **Get-Module** devrait lister l'exemple de module dans le répertoire ~/modules. Après avoir restauré PSModulePath à sa valeur originale, le module ne devrait plus être trouvé. C'est très important, parce que notre prochain objectif, c'est d'installer le module de manière à ce qu'il puisse être chargé sans modifier PSModulePath.

Publier le module localement

L'installation de modules ne peut être effectuée qu'à partir d'un référentiel. Nous allons donc publier le module dans le référentiel local :

```
$ModulePackage = «${RepoPath}/${ModuleName}.${ModuleVersion}.nupkg»
Remove-Item -Path $ModulePackage -ErrorAction 'SilentlyContinue'
```

```
Publish-Module -Path $ModulePath -Repository 'local'
```

Une des limitations d'utiliser **Publish-Module** avec un répertoire local, c'est que le paramètre -Force n'est pas suffisant pour écraser un module publié existant. Une solution pour contourner le problème serait de supprimer le fichier .nupkg au moment de la publication répétée d'un module de la même version. Ce fichier, c'est un paquet NuGet, qui est un fichier .zip contenant le module PowerShell et des métadonnées supplémentaires. Le paramètre -Verbose peut être ajouté à la commande **Publish-Module** pour en savoir plus sur le processus utilisé pour produire le fichier .nupkg.

Installer le module localement

À partir du référentiel 'local', utilisez la commande **Find** pour répertorier le module disponible pour l'installation :

```
PS > Find-Module -Name $ModuleName -Repository 'local' | select -Property Name,
Version
Name            Version
```

```
----            -------
Devolutions.Hub 2021.1.0
```

Finalement, faites la commande **Install-Module** pour installer le module à partir du référentiel 'local' :

```
Install-Module -Name $ModuleName -Repository 'local' -Scope CurrentUser
```

Assurez-vous que le module ait été correctement installé en utilisant la commande **Get-InstalledModule** :

```
PS > Get-InstalledModule -Name $ModuleName | select -Property Name, Version
```

```
Name            Version
----            -------
Devolutions.Hub 2021.1.0
```

Faire le ménage

Pour restaurer le système à son état d'origine, désinstallez le module de test, désinscrivez le référentiel 'local' et supprimez les répertoires "~/psrepo" and "~/modules" :

```
Uninstall-Module -Name $ModuleName
Unregister-PSRepository -Name 'local'
Remove-Item «~/psrepo» -Force -Recurse -ErrorAction 'SilentlyContinue'
Remove-Item «~/modules» -Force -Recurse -ErrorAction 'SilentlyContinue'
```

Sinon, gardez le référentiel 'local' en place et commencez à l'utiliser à la place d'un serveur NuGet local ou d'un référentiel hébergé sur un partage de réseau.

Le mot de la fin

Vous avez trouvé cet article utile? Partagez la bonne nouvelle! On aime lire vos commentaires et, surtout, on aimerait beaucoup en savoir plus sur votre expérience avec les modules PowerShell.

On est curieux : si vous aviez une baguette magique, qu'est-ce que vous aimeriez changer sur la distribution des modules PowerShell?