

Windows 2022 Box avec Packer et Vagrant

I. Présentation

Dans ce tutoriel, nous allons apprendre à créer une box Vagrant sous Windows Server 2022 à partir de l'outil Packer afin d'avoir un modèle de machine prêt à l'emploi. Cette démonstration est réalisée sous Windows 11, avec la solution VMware Workstation Pro 17 pour la virtualisation.

II. Infrastructure as Code : Vagrant et Packer

L'**Infrastructure as Code** (IAC) est une approche pour gérer son **infrastructure par l'intermédiaire de fichiers de configuration déclaratifs et de scripts** qui vont nous permettre d'automatiser le **provisionnement**, la **configuration** et le **déploiement** de nouveaux systèmes et services. Par exemple, la création d'une nouvelle machine virtuelle avec une configuration bien spécifique, de façon automatique, grâce aux directives présentes dans le fichier déclaratif.

Pour faire de l'Infrastructure as Code, on va pouvoir s'appuyer sur de nombreux outils dont ceux de l'éditeur HashiCorp parmi lesquels on retrouve **Packer**, **Vagrant** ou encore **Terraform**.

A. Qu'est-ce que Vagrant ?

Libre et open source, Vagrant a pour objectif de **faciliter la création d'environnements de développement** (pas pour de la production). Pour créer un nouvel environnement de développement, Vagrant va déployer ce que l'on appelle des box, c'est-à-dire des images prêtes à l'emploi (Linux, Windows, etc.).

Grâce à des fichiers déclaratifs, il va être possible de déployer une ou plusieurs machines virtuelles très rapidement, de manière à disposer d'un nouvel environnement propre et prêt à l'emploi. Il est compatible avec de nombreux environnements tels que **VirtualBox**, **Hyper-V**, **VMware** ainsi que **Docker**, grâce à ce que l'on appelle *des providers*.

Note : il existe des box Vagrant prêtes à l'emploi et disponibles sur [cette page](#).

B. Qu'est-ce que Packer ?

Packer est libre également et il est aussi proposé par HashiCorp ! Il a pour objectif de **faciliter la création d'images de systèmes d'exploitation**, notamment pour du Windows ou du Linux, tout en permettant la création de machines virtuelles ou de containers.

Comme Vagrant, Packer est compatible avec de nombreux environnements et il peut préparer ces images système sur plusieurs environnements en même temps. En effet, on peut utiliser Packer pour créer une image sur **Hyper-V**, **VMware ESXi**, **VMware Workstation**, ou encore **VirtualBox**, mais on peut aussi travailler sur des fournisseurs Cloud comme **Microsoft Azure** ou **AWS**.

En production, Packer peut être utilisée pour déployer une nouvelle machine virtuelle avec une configuration spécifique, des outils de base et les dernières mises à jour.

C. Cas pratique

Dans ce cas pratique, Packer va nous servir à **créer une image Windows Server 2022 pour VMware Workstation**, avec la dernière version de PowerShell et les VMware Tools. Une fois que cette image sera créée, **Packer va en créer une box Vagrant**.

Puis, Packer passera le relais à Vagrant.

En effet, on utilisera Vagrant pour créer une nouvelle machine virtuelle basée sur notre box, cette dernière étant utilisée comme source pour déployer la nouvelle VM (sans altérer la box qui est statique).

Même si Packer et Vagrant sont fréquemment utilisés sur Linux, ils sont compatibles avec Windows. Ici, on utilise une machine Windows 11 tout au long de la procédure.

III. Installer Packer sous Windows 11

Comme l'explique la documentation officielle de Packer

<https://developer.hashicorp.com/packer/install> , l'outil peut être installé sur Windows à l'aide de Chocolatey (gestionnaire de paquets gratuit et open source).

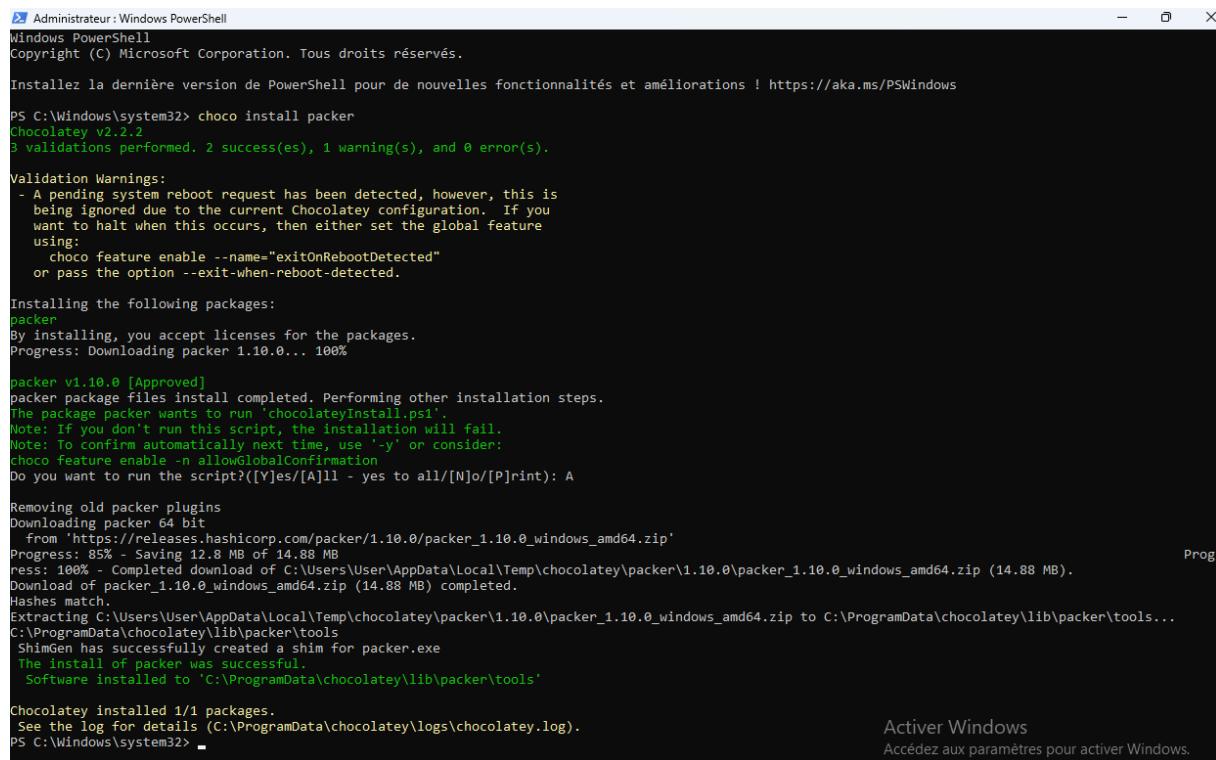
Dans une console PowerShell exécutée en tant qu'administrateur, on va pouvoir commencer par installer Chocolatey via la commande officielle :

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-
Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/inst
all.ps1'))
```

Puis, nous allons installer Packer via Chocolatey avec cette commande :

```
choco install packer
```

DevOps Packer - Vagrant



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Windows\system32> choco install packer
chocolatey v2.2.2
3 validations performed. 2 success(es), 1 warning(s), and 0 error(s).

Validation Warnings:
- A pending system reboot request has been detected, however, this is
being ignored due to the current Chocolatey configuration. If you
want to halt when this occurs, then either set the global feature
using:
  choco feature enable --name="exitOnRebootDetected"
or pass the option --exit-when-reboot-detected.

Installing the following packages:
packer
By installing, you accept licenses for the packages.
Progress: Downloading packer 1.10.0... 100%

packer v1.10.0 [Approved]
packer package files install completed. Performing other installation steps.
The package packer wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable --allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N)o/[P]rint): A

Removing old packer plugins
Downloading packer 64 bit
  from 'https://releases.hashicorp.com/packer/1.10.0/packer_1.10.0_windows_amd64.zip'
Progress: 85% - Saving 12.8 MB of 14.88 MB
ress: 100% - Completed download of C:\Users\user\AppData\Local\Temp\chocolatey\packer\1.10.0\packer_1.10.0_windows_amd64.zip (14.88 MB).
Download of packer_1.10.0_windows_amd64.zip (14.88 MB) completed.
Hashes match.
Extracting C:\Users\user\AppData\Local\Temp\chocolatey\packer\1.10.0\packer_1.10.0_windows_amd64.zip to C:\ProgramData\chocolatey\lib\packer\tools...
C:\ProgramData\chocolatey\lib\packer\tools
ShimGen has successfully created a shim for packer.exe
The install of packer was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\packer\tools'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\Windows\system32>
```

Activer Windows
Accédez aux paramètres pour activer Windows.

Une fois l'installation effectuée, vérifiez que Packer soit bien installé en essayant d'obtenir des informations sur la version installée :

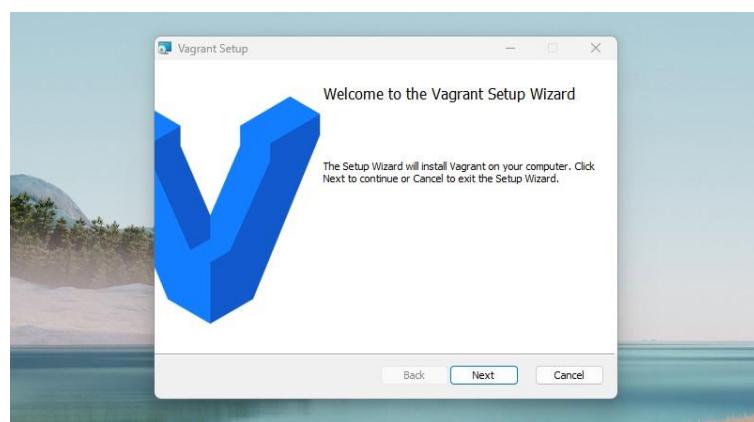
```
packer version
```

IV. Installer Vagrant sous Windows 11

Pour installer Vagrant sur Windows 11, il suffit de télécharger un package MSI sur le site officiel de HashiCorp ! Accédez au lien ci-dessous, cliquez sur l'onglet "Windows" et téléchargez le package MSI via le bouton "**Download**".

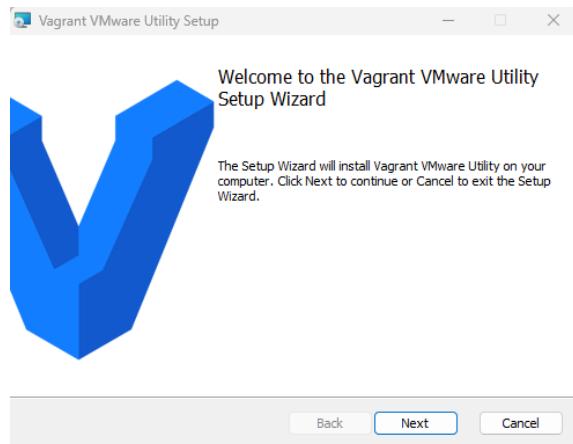
- [Télécharger Vagrant pour Windows](#)

Effectuez l'installation en suivant l'installation : ce n'est qu'une question de quelques clics et d'un redémarrage de machine.



En complément de Vagrant en lui-même, vous devez installer "**Vagrant VMware Utility**" pour la prise en charge de VMware Workstation. Là encore, c'est un package à télécharger depuis le site de HashiCorp et à installer en quelques clics : 

- [Télécharger Vagrant VMware Utility](#)



Puis, terminez par installer le plugin VMware de Vagrant avec cette commande :

```
# Installer le plugin VMware
vagrant plugin install vagrant-vmware-desktop

# Lister les plugins Vagrant
vagrant plugin list

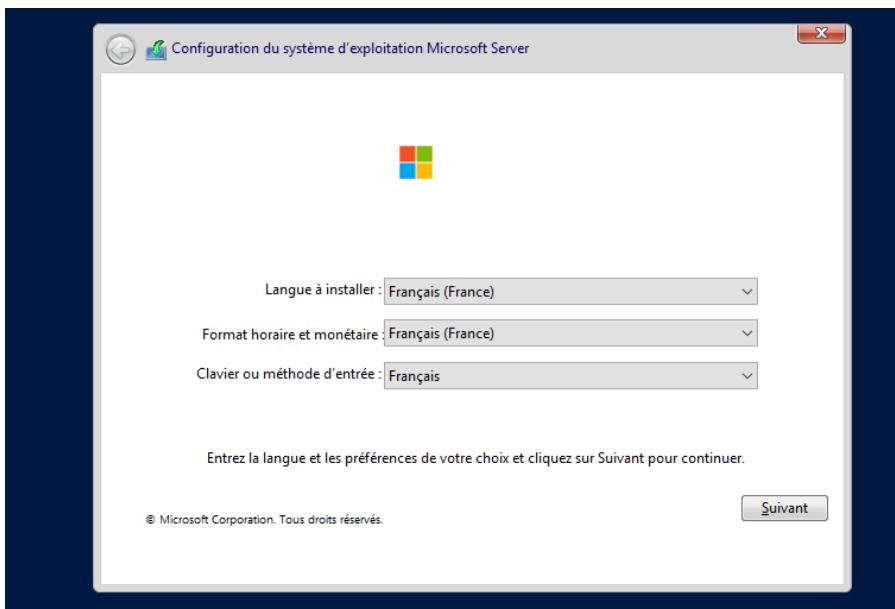
# Mettre à jour le plugin VMware
vagrant plugin update vagrant-vmware-desktop
```

Voilà, **Packer et Vagrant sont prêts à l'emploi** pour la suite des opérations !

V. Générer un fichier autounattend.xml de Windows Server 2022

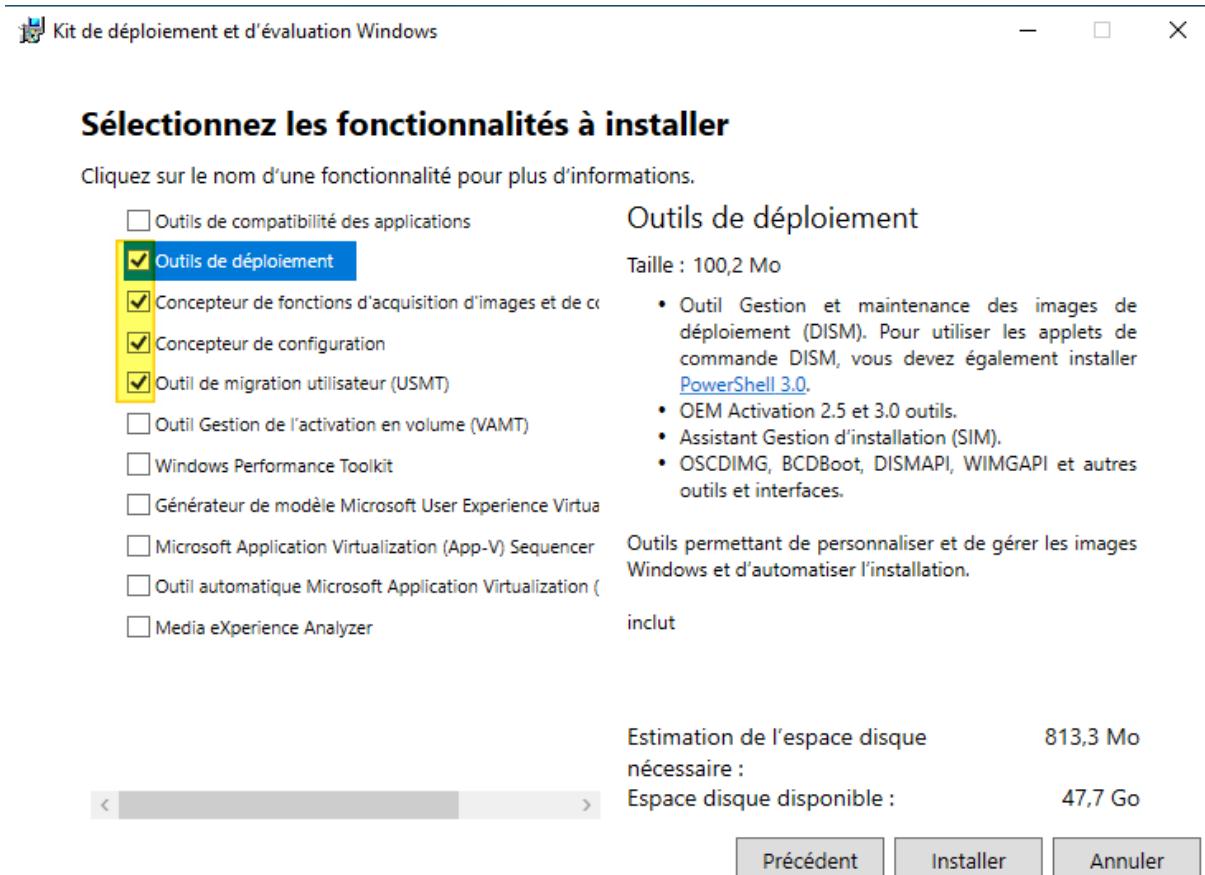
Pour générer un fichier de réponse pour Windows Server 2022 en partant de zéro, il est possible d'utiliser l'outil "**Windows System Image Manager**" ou en français "**Assistant gestion d'installation**". Il va permettre de charger une image WIM et d'**ajouter des instructions pour automatiser l'installation du système** : partitionnement, choix de la langue, clé de licence, mot de passe administrateur, etc... Tout ce qui est demandé lorsque l'on effectue l'installation à la main en fait.

Ce fichier est très important, car il va permettre à Packer d'effectuer l'installation du système à notre place, et donc de l'automatiser. Toutefois, l'utilisation d'un fichier de réponses n'est pas spécifique à l'utilisation de Packer : c'est une technique utile dans bien d'autres situations.



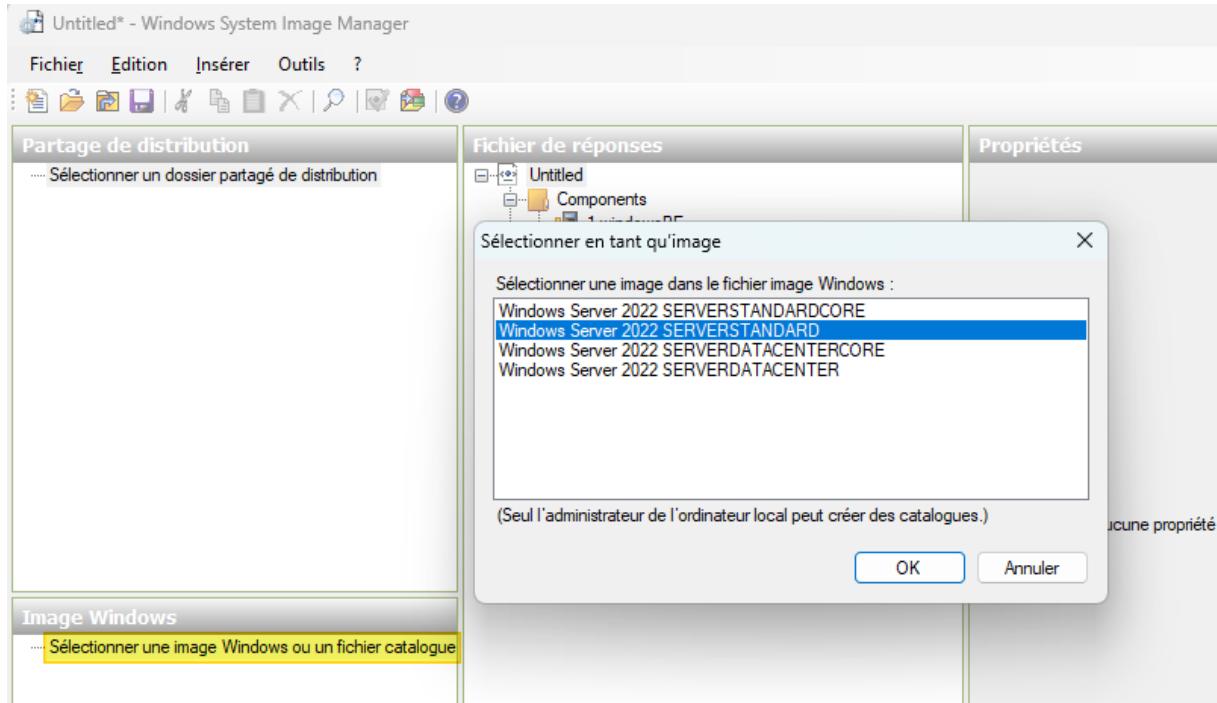
1 - Installez le kit Windows ADK (avec les mêmes options que pour [MDT](#))

- [Télécharger Windows ADK pour Windows 11 22H2](#)

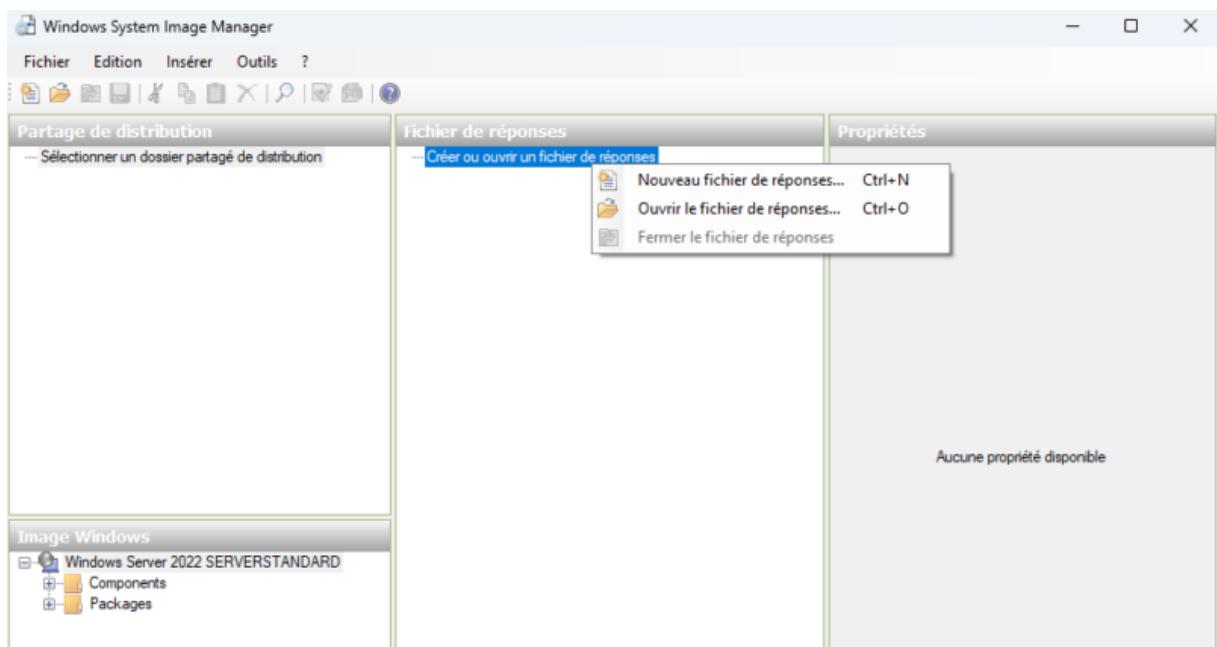


2 - Ouvrez Windows System Image Manager (WSIM) qui s'appelle "Assistant gestion d'installation" sur un système en français.

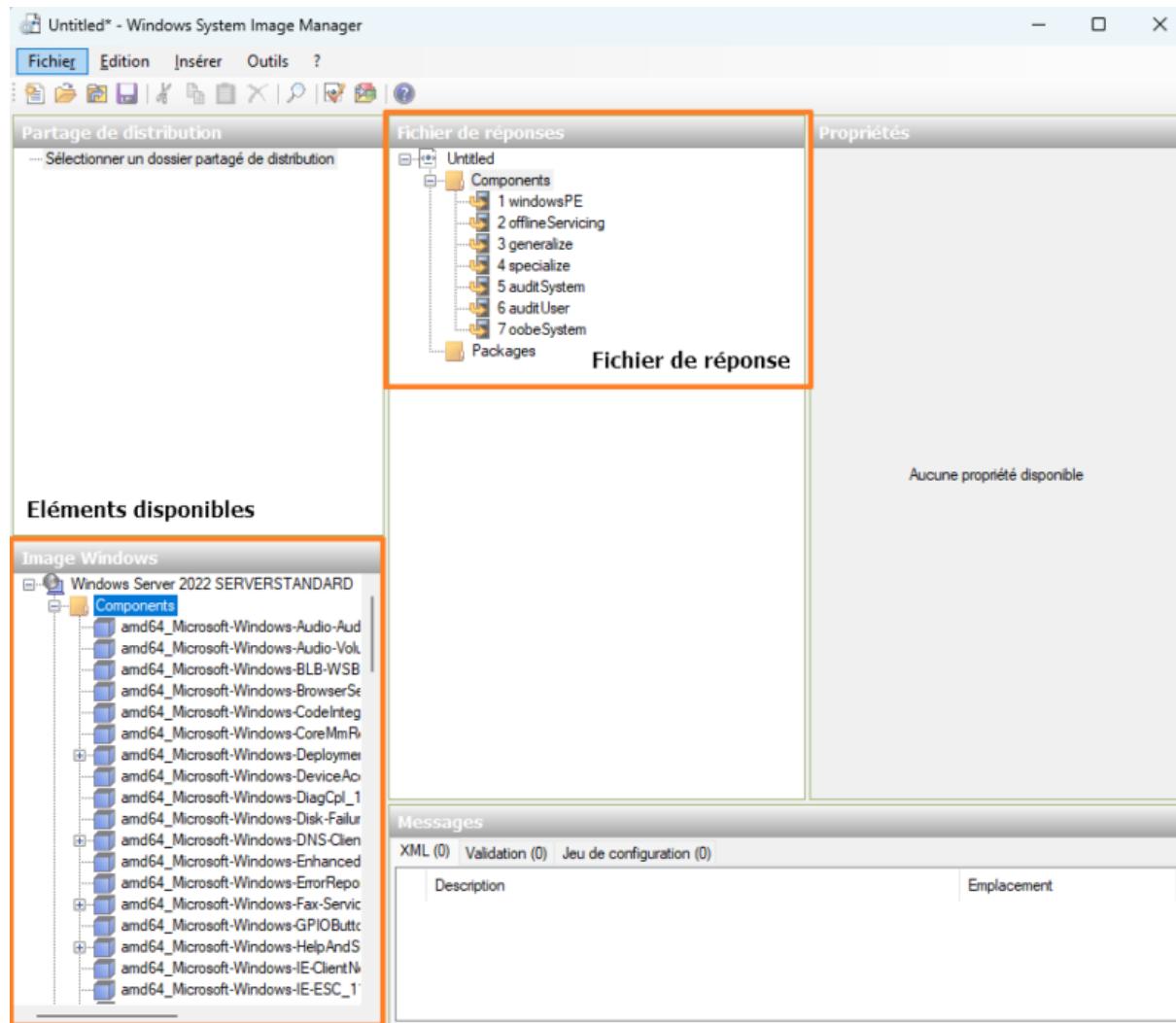
3 - Ajoutez l'image "install.wim" de Windows Server 2022 dans l'application, ce qui implique de charger une image "install.wim" à partir d'un support accessible en écriture. Pour cela, montez une image ISO sur votre machine et copiez le fichier "install.wim" de l'image ISO vers votre disque local. Chargez ce fichier dans WSIM et sélectionner l'image de votre choix, ici ce sera "Windows Server 2022 SERVERSTANDARD**".**



4 - Créez un nouveau fichier de réponse avec un clic droit dans la partie centrale puis "Nouveau fichier de réponses".



5 - Construisez le fichier de réponses de Windows Server 2022. Suite à l'ajout de l'image et à la création d'un fichier de réponse vierge, il y a une liste d'éléments disponibles en bas à gauche, tandis que la partie centrale correspond au contenu du fichier de réponses. Pour le moment, ce fichier est vide.

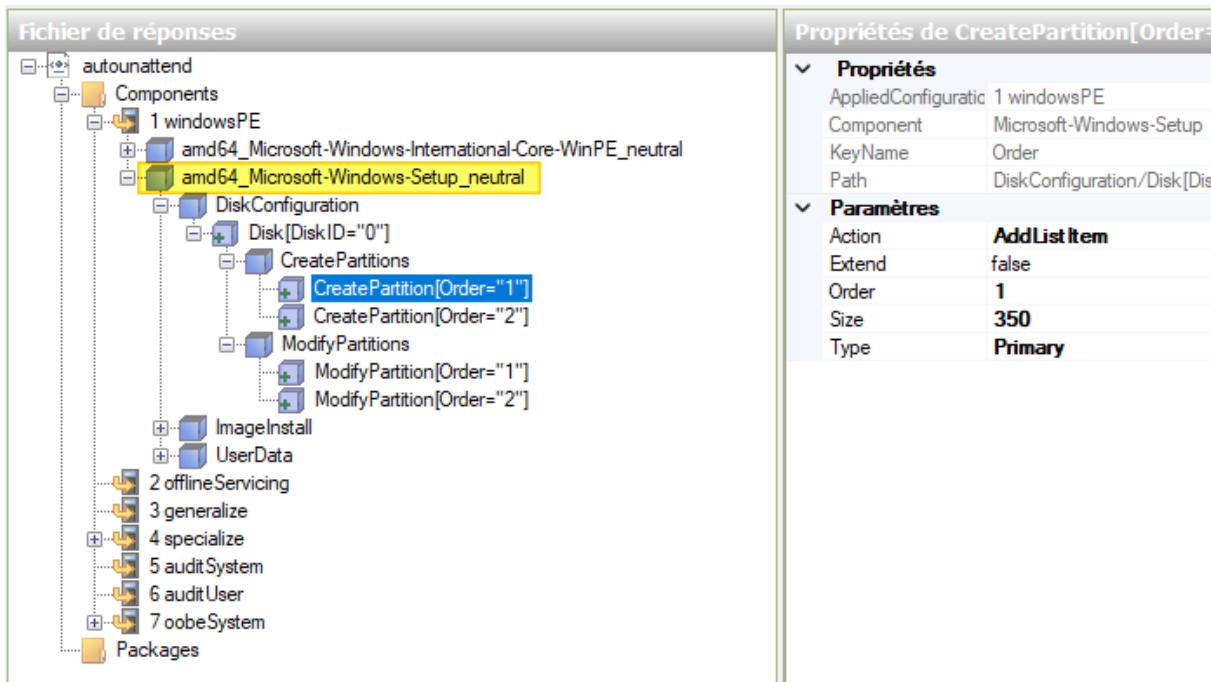


Il doit être complété pour intégrer les informations nécessaires pour l'installation automatique (afin de **répondre à l'assistant d'installation** à notre place) :

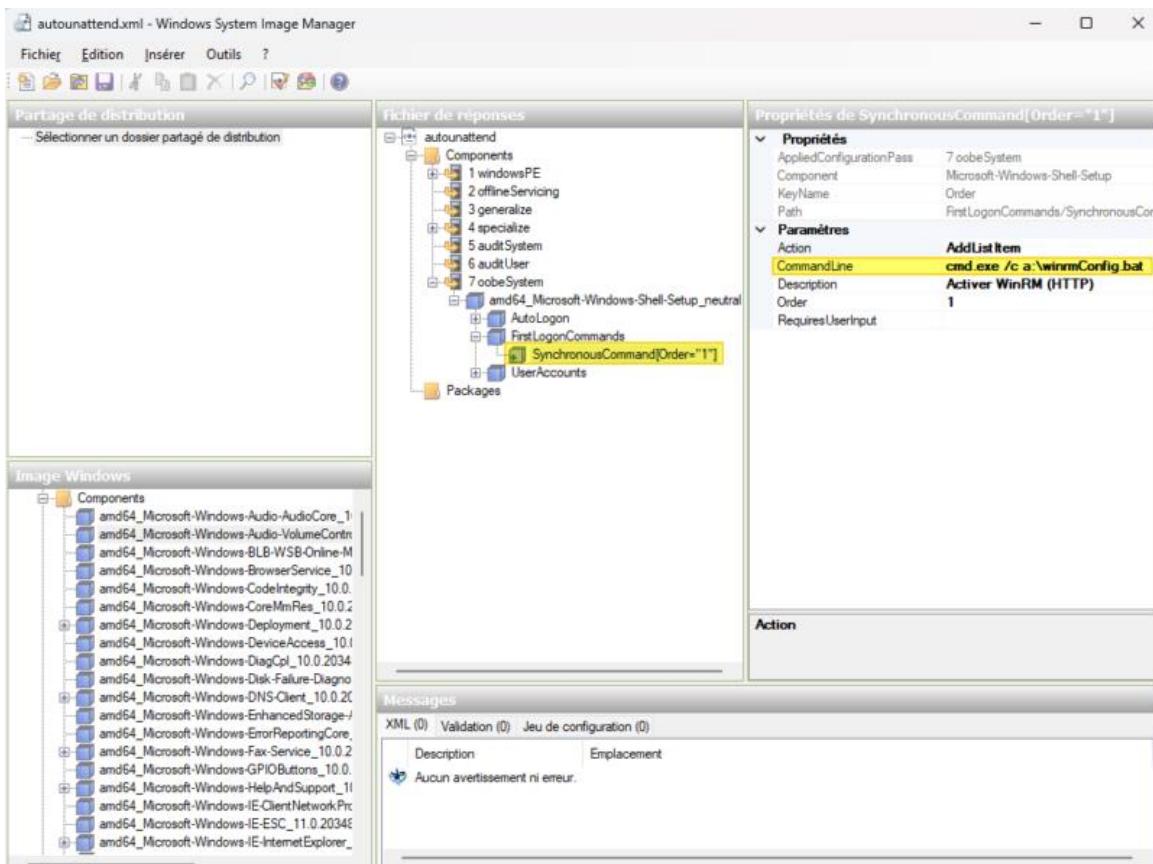
- Partitionnement du disque,
- Langage du système
- Disposition du clavier
- Nom du compte Administrateur
- Mot de passe du compte Administrateur
- Clé de licence
- Etc.

Pour ajouter un composant au fichier de réponses, il faut effectuer un clic droit dessus pour l'ajouter à une section du fichier (*windowsPE*, *offline Servicing*, *generalize*, *etc.*).

Par exemple, avec "**amd64_Microsoft-Windows-Setup_neutral**", on va pouvoir configurer l'élément "**DiskConfiguration**" pour définir le schéma de partitionnement du disque.



Par ailleurs, et là on est sur un besoin spécifique à la **construction d'une image** avec un outil comme Packer, on doit **activer WinRM sur le système**. Pour cela, on configure l'élément "**SynchronousCommand**" disponible dans "**amd64_Microsoft-Windows-Shell-Setup_neutral**" de façon à exécuter le script "winrmConfig.bat" pour activer et préconfigurer WinRM sur la machine. En fait, **Packer a besoin de se connecter à la machine via WinRM pour effectuer la configuration**.



Pensez à enregistrer vos modifications au fur et à mesure, à l'aide de l'icône en forme de disquette. Nommez ce fichier **autounattend.xml**. Le fichier XML obtenu est le reflet de la configuration effectuée avec l'outil WSIM. Plutôt que de partir de zéro, vous pouvez utiliser mon exemple de fichier XML (et l'ouvrir dans WSIM pour le personnaliser) :

- [GitHub - Fichier autounattend.xml pour Windows Server 2022](#)

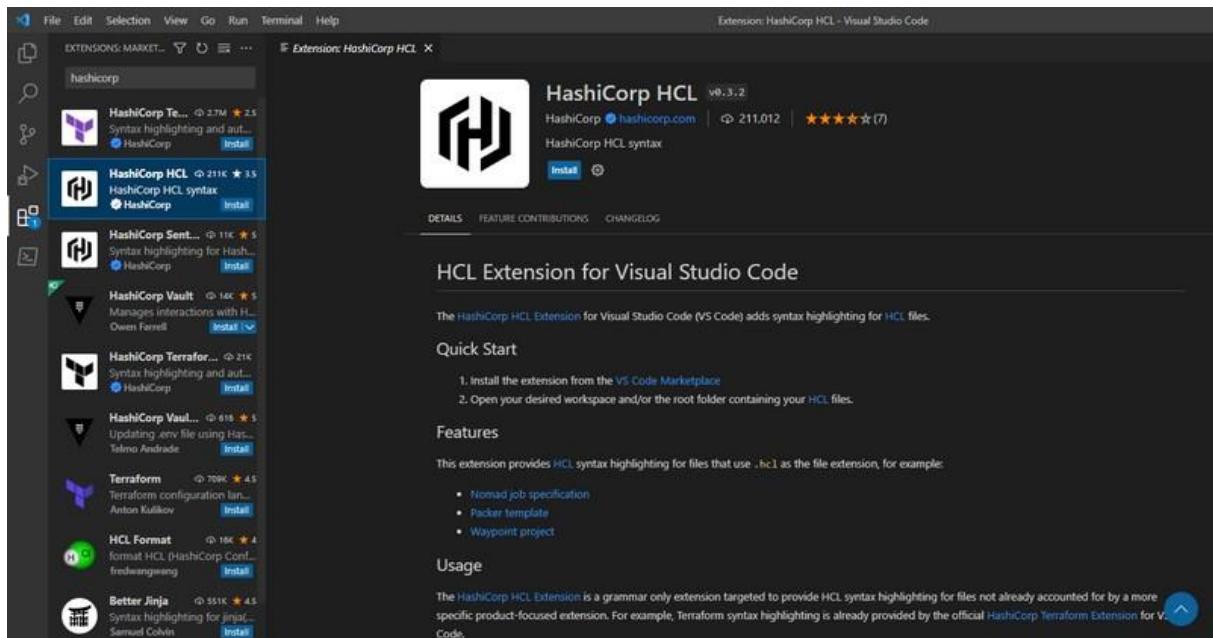
```
Packer > VM-WS2022 > setup > autounattend.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <unattend xmlns="urn:schemas-microsoft-com:unattend">
3   <settings pass="windowsPE">
4     <component name="Microsoft-Windows-International-Core-WinPE" processorArchitecture="amd64"
5       <SetupUILanguage>
6         <UILanguage>fr-FR</UILanguage>
7       </SetupUILanguage>
8       <UILanguage>fr-FR</UILanguage>
9       <InputLocale>fr-FR</InputLocale>
10      <SystemLocale>fr-FR</SystemLocale>
11      <UILanguageFallback>fr-FR</UILanguageFallback>
12      <UserLocale>fr-FR</UserLocale>
13    </component>
14    <component language="neutral" name="Microsoft-Windows-Setup" processorArchitecture="amd64"
15      <DiskConfiguration>
16        <Disk wcm:action="add">
17          <CreatePartitions>
18            <CreatePartition wcm:action="add">
19              <Type>Primary</Type>
20              <Order>1</Order>
21              <Size>350</Size>
22            </CreatePartition>
23            <CreatePartition wcm:action="add">
24              <Order>2</Order>
25              <Type>Primary</Type>
26              <Extend>true</Extend>
27            </CreatePartition>
28          </CreatePartitions>
29          <ModifyPartitions>
30            <ModifyPartition wcm:action="add">
31              <Active>true</Active>
32              <Format>NTFS</Format>
33              <Label>boot</Label>
34              <Order>1</Order>
35              <PartitionID>1</PartitionID>
36            </ModifyPartition>
37            <ModifyPartition wcm:action="add">
38              <Format>NTFS</Format>
39              <Label>Windows Server 2022</Label>
40              <Letter>C</Letter>
41              <Order>2</Order>
42              <PartitionID>2</PartitionID>
43            </ModifyPartition>
```

Le fichier de réponses étant prêt, nous pouvons passer à la suite. Gardez à l'esprit qu'il sera peut-être nécessaire de l'éditer par la suite pour faire des ajustements ou ajouter des étapes supplémentaires.

VI. Créer le modèle de VM avec Packer sous VMware Workstation

Pour la syntaxe des fichiers déclaratifs Packer, nous avons le choix entre deux langages : **JSON** et **HCL2**, le second correspondant à la syntaxe propre aux outils de chez HashiCorp. Pour cet exemple, ce sont des fichiers au format HCL qui sont utilisés.

Commencez par **créer un dossier "WS2022"** qui regroupera tous les fichiers de notre projet, et à l'intérieur, créez le fichier "**ws2022.pkr.hcl**" qui sert à déclarer la configuration de notre machine virtuelle. Pour l'édition, je vous recommande l'utilisation de Visual Studio Code avec l'extension "**HashiCorp HCL**".



A. Le bloc packer

Le fichier HCL se découpe en plusieurs blocs. Le premier bloc se nomme "**packer**" et va permettre de déclarer la version de Packer requise, ainsi que le(s) plugin(s) nécessaires pour le bon fonctionnement de notre projet. Ici, le seul plugin qui fait office de dépendance se nomme "**vmware**".

- [Documentation Packer - Les blocs](#)

Ce qui donne :

```
packer {
  required_version = ">= 1.7.0"
  required_plugins {
    vmware = {
      version = ">= 1.0.0"
      source = "github.com/hashicorp/vmware"
    }
  }
}
```

B. Le bloc source

Ensuite, on doit construire un second bloc de type "**source**" où l'on va préciser que l'on veut utiliser le builder "**vmware-iso**" pour VMware et nommer ce builder "**WS2022**". Ainsi, Packer sait que les instructions de ce bloc sont adaptées à VMware Workstation. Dans ce bloc, on va **déclarer tous les paramètres qui vont permettre à Packer de créer la machine virtuelle et d'effectuer l'installation du système** : nom de la VM, type de système invité, chemin vers l'image ISO, nombre de CPU, quantité de RAM, type de connexion au réseau virtuel, etc....

L'ensemble des paramètres pris en charge par le builder "vmware-iso" sont définis dans la documentation :

- [Documentation Packer - Builder vmware-iso - Paramètres](#)

Voici la définition utilisée pour cet exemple :

```
source "vmware-iso" "WS2022" {
    // Nom de la VM et système invité (Windows Server 2022)
    vm_name = "WS2022"
    guest_os_type = "windows2019srvnext-64"
    version = "20"

    // ISO source pour l'installation et checksum (Get-FileHash)
    iso_url = "W:/ISO/fr-
fr_windows_server_2022_updated_dec_2022_x64_dvd_34eae3b9.iso"
    iso_checksum = "md5:290B43B5A6FE9B52B561D34EB92E8003"

    // Config de la VM : CPU, RAM, disque, réseau
    cpus = "2"
    cores = "1"
    memory = "4096"
    disk_adapter_type = "nvme"
    disk_size = "40960"
    network_adapter_type = "e1000e"
    network = "nat"

    // Connexion WINRM
    communicator = "winrm"
    winrm_port = "5985"
    insecure_connection = "true"
    winrm_username = "Administrateur"
    winrm_password = "P@ssword!"

    // Lecteur disquette (pour charger autounattend.xml et les scripts)
    floppy_files = ["${path.root}/setup/"]
    floppy_label = "floppy"

    // Commande pour arrêter le système (ici, on effectue un SYSPREP avant
    l'arrêt)
    // arrêt seul : shutdown_command = "shutdown /s /t 30 /f"
    shutdown_command = "C:\\Windows\\System32\\Sysprep\\sysprep.exe
/generalize /oobe /shutdown /unattend:a:\\sysprep-autounattend.xml"
    shutdown_timeout = "60m"
}
```

Note : dans l'exemple ci-dessus, une valeur statique est affectée à chaque paramètre. Pour que notre modèle soit plus flexible, **il est recommandé d'utiliser un système de variables**. Ainsi, le fichier déclaratif ci-dessus contiendrait principalement des noms de variables, et un second fichier permettrait d'associer des valeurs à ces variables.

La machine virtuelle créée s'appellera WS2022 (le nom du système est déterminé par le fichier autounattend.xml), elle aura 2 vCPU avec 1 cœur, 4 Go de RAM et 40 Go d'espace disque. La carte réseau de la VM sera connectée en mode "NAT".

On peut voir aussi qu'une image ISO disponible en local est utilisée (paramètre **iso_url**) et que l'on vérifie l'intégrité de l'image système avant de l'utiliser (paramètre **iso_checksum**). La valeur spécifiée pour le paramètre **iso_checksum** est obtenue avec la commande PowerShell Get-FileHash afin de calculer la somme MD5 de l'image ISO source.

Pour se connecter à la machine virtuelle, Packer va utiliser le **protocole WinRM** (paramètre **communicator**) en mode HTTP et s'authentifier avec l'utilisateur "Administrateur" et le mot de passe "P@ssword!" (paramètres **winrm_username** et **winrm_password**). Ces identifiants doivent correspondre au compte créé à l'installation avec le fichier autounattend.xml.

D'ailleurs, **Packer doit pouvoir accéder au fichier autounattend.xml et à notre script "winrmConfig.bat"** qui sert à activer et configurer WinRM. Pour cela, on s'appuie sur le **lecteur disquette de la machine virtuelle** : la disquette virtuelle va contenir le contenu du répertoire "setup" que l'on va créer à la racine de notre projet "VM-WS2022". Comme le montre l'exemple ci-dessous, ce répertoire contient le fichier XML et le script ".bat" ainsi que d'autres fichiers qui seront dévoilés plus tard ! 😊

	Nom	Modifié le	Type	Taille
	autounattend.xml	03/05/2023 11:26	xmlfile	6 Ko
	boot.wim	28/04/2023 10:50	Script Windows P...	1 Ko
	boot.vhd	28/04/2023 17:12	Script Windows P...	2 Ko
	boot.vpx	28/04/2023 17:57	Script Windows P...	1 Ko
	boot.vsd	03/05/2023 14:59	xmlfile	4 Ko
	winrmConfig.bat	26/04/2023 11:54	Fichier de commande Windows	1 Ko

Enfin, Packer doit avoir connaissance d'une commande lui permettant d'éteindre la machine virtuelle lorsqu'il aura terminé d'exécuter les différentes actions (paramètre **shutdown_command**).

On pourrait tout simplement arrêter la machine, mais comme il s'agit d'**une image Windows destinée à être dupliquée** (modèle de box Vagrant), il est essentiel de **réaliser un SYSPREP** sinon les machines générées auront le même identifiant unique... Pour que le SYSPREP soit effectué et que la nouvelle machine virtuelle soit initialisée de façon automatique, un second fichier de réponses nommé "**sysprep-autounattend.xml**" est utilisé (*il contient beaucoup moins de paramètres !*). Ceci explique la ligne suivante :

```
shutdown_command = "C:\\Windows\\system32\\Sysprep\\sysprep.exe /generalize /oobe /shutdown /unattend:a:\\sysprep-autounattend.xml"
```

Ce fichier est disponible aussi sur mon GitHub et peut être lu et personnalisé avec l'outil **WSIM** ("Assistant gestion d'installation").

- [**GitHub - Fichier sysprep-autounattend.xml**](#)

Pour le fichier HCL complet, c'est par ici :

- [**GitHub - ws2022.pkr.hcl**](#)

C. Le bloc build

Il est important de préciser qu'à partir du moment où WinRM sera accessible, **Packer va se connecter à la machine virtuelle pour exécuter les instructions du bloc "build"**. En se basant sur la structure du fichier, on voit que l'on sépare bien la partie "construction de la VM sur la plateforme" et la partie "configuration de la VM (système)".

Dans le bloc "build", on va utiliser ce que l'on appelle un "**provisioner**" pour exécuter différentes actions. Par exemple, il y a un provisioner nommé "**powershell**" qui sert à exécuter des scripts ou commandes PowerShell. Dans l'exemple ci-dessous, **deux scripts PowerShell stockés dans "setup"** (*donc accessibles via le lecteur disquette*) sont exécutés :

- **install-powershell.ps1** pour installer la dernière version de PowerShell (PowerShell 7.X à l'heure actuelle) via Internet
- **install-vmtools.ps1** pour installer les VMware Tools de Windows Server dans la machine virtuelle

Ces scripts sont disponibles sur mon GitHub. En complément, on va effectuer un redémarrage de la machine virtuelle via le provisioner "**windows-restart**".

Enfin, une instruction de type "**post-processor**" est déclarée. Elle est optionnelle et sera traitée après les différents provisioners. Dans le cas présent, on utilise le post-processor "**vagrant**" pour **générer une box Vagrant** à partir de l'image virtuelle construite par Packer (en l'appelant *packer_WS2022_vmware.box*).

Ce qui donne :

```
build {
    sources = ["sources.vmware-iso.WS2022"]

    // Installer PowerShell (dernière version disponible sur GitHub)
    provisioner "powershell" {
        script = "${path.root}/setup/install-powershell.ps1"
    }

    // Installer VMware Tools
    provisioner "powershell" {
        script = "${path.root}/setup/install-vmtools.ps1"
    }

    // Initier un redémarrage de la machine
    provisioner "windows-restart" {
        restart_timeout = "10m"
    }

    // Export sous la forme d'une box Vagrant
    post-processor "vagrant" {
        keep_input_artifact = false
        output = "packer_{{.BuildName}}_{{.Provider}}.box"
        provider_override = "vmware"
    }
}
```

Tout le contenu évoqué dans cette partie de l'article est regroupé dans le même fichier : "**ws2022.pkr.hcl**". Il contient la déclaration de toutes les actions que Packer doit effectuer. Sauvegardez le fichier.

VII. Générer la VM avec Packer (VMware Workstation)

Le fichier déclaratif de Packer étant prêt, nous allons pouvoir tester la construction de notre image. Ouvrez une console PowerShell et positionnez-vous dans le répertoire où se situe le fichier HCL. En ce qui me concerne :

```
cd W:\Packer\VM-WS2022
```

Ensuite, on appelle le fichier "**HCL**" avec l'option "**init**" pour que Packer lise notre fichier afin de voir de **quels plugins** il aura besoin pour construire l'image. Il va télécharger les plugins nécessaires, en l'occurrence ici le plugin "**vmware**". La première fois que l'on utilise un nouveau template, c'est **important de commencer par un init**. La documentation officielle le précise clairement : "*Il s'agit de la première commande à exécuter lorsque l'on travaille avec un nouveau modèle ou un modèle existant.*"

```
packer.exe init ws2022.pkr.hcl
```

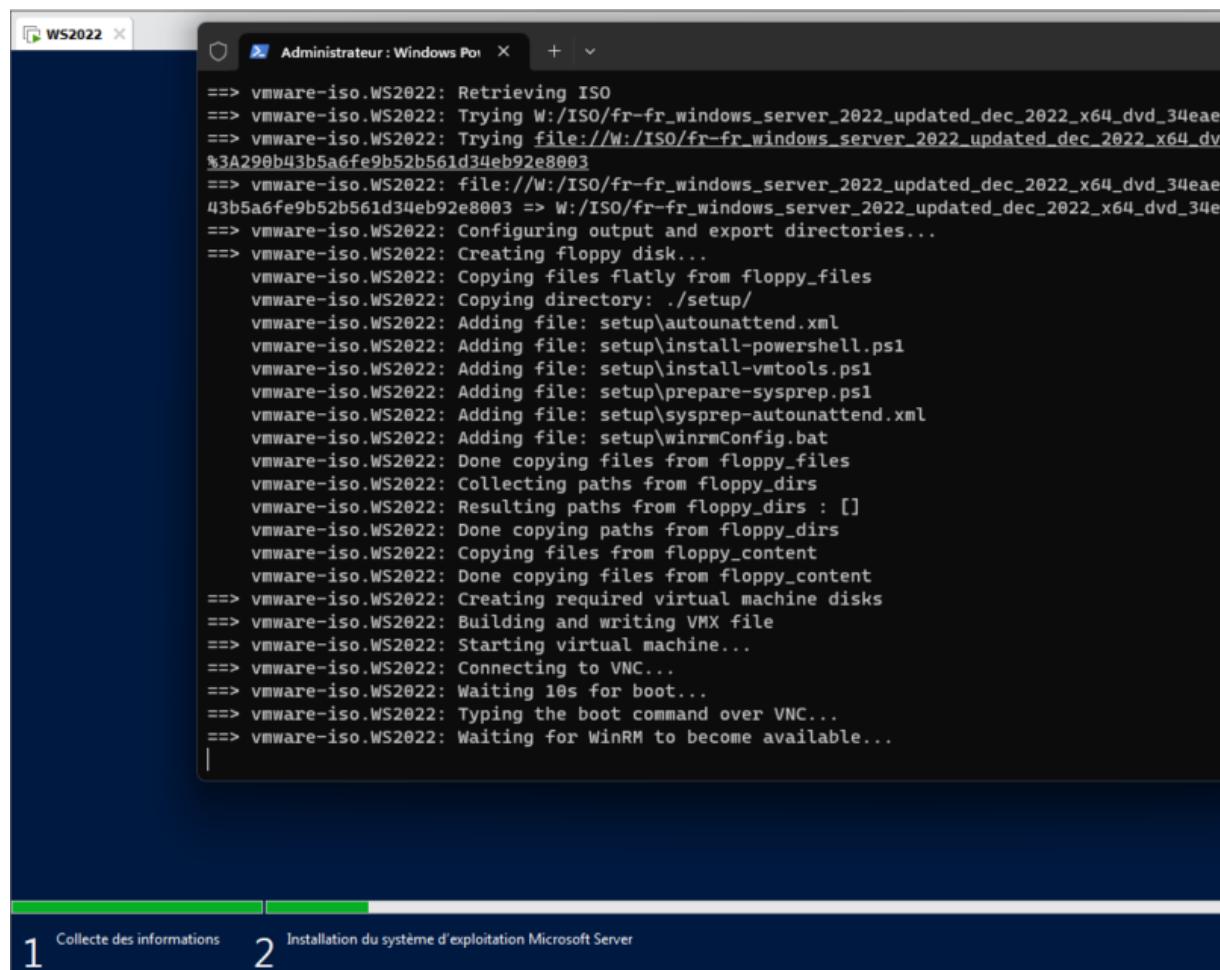
Une fois que c'est fait, vous pouvez lancer la construction de l'image en remplaçant "init" par "build". Ce qui donne :

```
packer.exe build ws2022.pkr.hcl
```

À partir de ce moment-là, Packer va commencer par charger l'image ISO, calculer son checksum MD5 et comparer la valeur obtenue avec celle déclarée dans le fichier HCL. Puis, il va enchaîner les autres étapes, notamment créer la machine virtuelle dans VMware Workstation, charger les fichiers du répertoire "setup" dans le lecteur disquette, etc... **L'avancement est visible dans la console.**

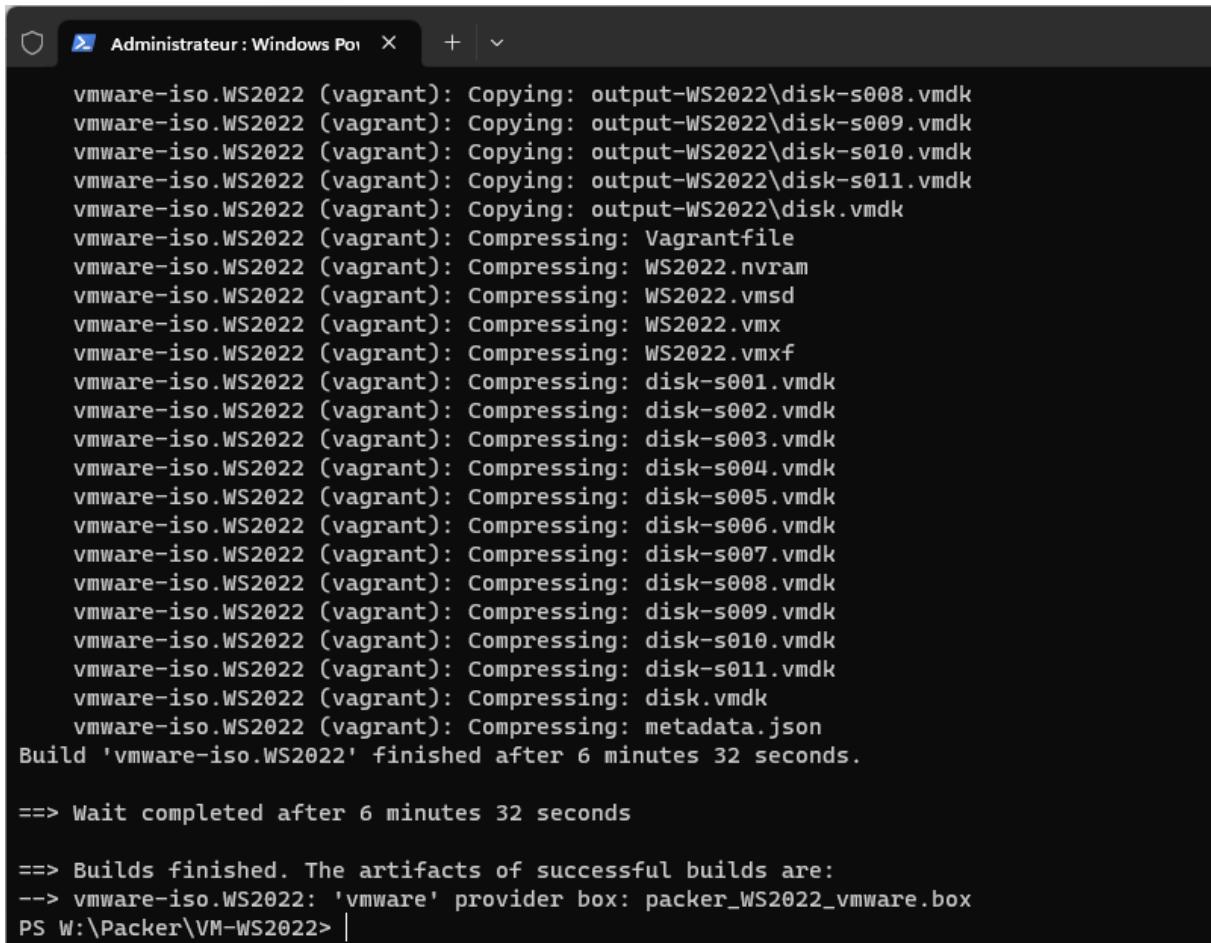
Grâce au fichier "autounattend.xml", Packer va pouvoir réaliser l'installation du système d'exploitation sans aucune action de notre part. En complément, pour suivre l'avancement, il est possible de regarder ce qu'il se passe dans **VMware Workstation : en temps réel, on voit les actions s'enchaîner.**

À partir du moment où le script "**winrmConfig.bat**" sera exécuté, Packer pourra se connecter en WinRM à la machine virtuelle. Ceci va lui permettre d'exécuter les actions du bloc "build". On voit bien que Packer est en attente grâce à la présence de la ligne "**Waiting for WinRM to become available...**" dans la console.



```
==> vmware-iso.WS2022: Retrieving ISO
==> vmware-iso.WS2022: Trying W:/ISO/fr-fr_windows_server_2022_updated_dec_2022_x64_dvd_34eae
==> vmware-iso.WS2022: Trying file://W:/ISO/fr-fr_windows_server_2022_updated_dec_2022_x64_dv
%3A290b43b5a6fe9b52b561d34eb92e8003
==> vmware-iso.WS2022: file://W:/ISO/fr-fr_windows_server_2022_updated_dec_2022_x64_dvd_34eae
43b5a6fe9b52b561d34eb92e8003 => W:/ISO/fr-fr_windows_server_2022_updated_dec_2022_x64_dvd_34e
==> vmware-iso.WS2022: Configuring output and export directories...
==> vmware-iso.WS2022: Creating floppy disk...
    vmware-iso.WS2022: Copying files flatly from floppy_files
    vmware-iso.WS2022: Copying directory: ./setup/
    vmware-iso.WS2022: Adding file: setup\autounattend.xml
    vmware-iso.WS2022: Adding file: setup\install-powershell.ps1
    vmware-iso.WS2022: Adding file: setup\install-vmtools.ps1
    vmware-iso.WS2022: Adding file: setup\prepare-sysprep.ps1
    vmware-iso.WS2022: Adding file: setup\sysprep-autounattend.xml
    vmware-iso.WS2022: Adding file: setup\winrmConfig.bat
    vmware-iso.WS2022: Done copying files from floppy_files
    vmware-iso.WS2022: Collecting paths from floppy_dirs
    vmware-iso.WS2022: Resulting paths from floppy_dirs : []
    vmware-iso.WS2022: Done copying paths from floppy_dirs
    vmware-iso.WS2022: Copying files from floppy_content
    vmware-iso.WS2022: Done copying files from floppy_content
==> vmware-iso.WS2022: Creating required virtual machine disks
==> vmware-iso.WS2022: Building and writing VMX file
==> vmware-iso.WS2022: Starting virtual machine...
==> vmware-iso.WS2022: Connecting to VNC...
==> vmware-iso.WS2022: Waiting 10s for boot...
==> vmware-iso.WS2022: Typing the boot command over VNC...
==> vmware-iso.WS2022: Waiting for WinRM to become available...
```

Quelques minutes plus tard, ou plutôt 6 minutes et 32 secondes plus tard pour être précis, Packer a terminé de construire notre machine virtuelle et il a généré une box Vagrant correspondante.



```

vmware-iso.WS2022 (vagrant): Copying: output-WS2022\disk-s008.vmdk
vmware-iso.WS2022 (vagrant): Copying: output-WS2022\disk-s009.vmdk
vmware-iso.WS2022 (vagrant): Copying: output-WS2022\disk-s010.vmdk
vmware-iso.WS2022 (vagrant): Copying: output-WS2022\disk-s011.vmdk
vmware-iso.WS2022 (vagrant): Copying: output-WS2022\disk.vmdk
vmware-iso.WS2022 (vagrant): Compressing: Vagrantfile
vmware-iso.WS2022 (vagrant): Compressing: WS2022.nvram
vmware-iso.WS2022 (vagrant): Compressing: WS2022.vmsd
vmware-iso.WS2022 (vagrant): Compressing: WS2022.vmx
vmware-iso.WS2022 (vagrant): Compressing: WS2022.vmxsf
vmware-iso.WS2022 (vagrant): Compressing: disk-s001.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s002.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s003.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s004.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s005.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s006.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s007.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s008.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s009.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s010.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk-s011.vmdk
vmware-iso.WS2022 (vagrant): Compressing: disk.vmdk
vmware-iso.WS2022 (vagrant): Compressing: metadata.json
Build 'vmware-iso.WS2022' finished after 6 minutes 32 seconds.

==> Wait completed after 6 minutes 32 seconds

==> Builds finished. The artifacts of successful builds are:
--> vmware-iso.WS2022: 'vmware' provider box: packer_WS2022_vmware.box
PS W:\Packer\VM-WS2022> |

```

VIII. Créer une box Vagrant avec Packer

Comme je l'évoquais précédemment, c'est bien l'instruction "**post-processor "vagrant"**" qui permet d'indiquer à Packer qu'il doit générer une box Vagrant en guise de sortie, après avoir construit l'image dans VMware.

Cela est possible grâce à la présence de ce bloc dans le fichier HCL :

```
// Export sous la forme d'une box Vagrant
post-processor "vagrant" {
    keep_input_artifact = false
    output = "packer_{{.BuildName}}_{{.Provider}}.box"
    provider_override = "vmware"
}
```

Sans ce bout de code, Packer crée une machine virtuelle et s'arrête à la fin de l'exécution. Dans ce cas, **la machine virtuelle créée pourrait être utilisée, mais elle serait unique**. Grâce à la création d'un modèle de box Vagrant, on peut créer une multitude de machines virtuelles basées sur notre modèle généré avec Packer.

Sur ma machine, la box Vagrant est bien disponible :

Nom	Modifié le	Type	Taille
.vagrant	28/04/2023 11:20	Dossier de fichiers	
packer_cache	03/05/2023 15:12	Dossier de fichiers	
setup	02/05/2023 17:58	Dossier de fichiers	
packer_WS2022_vmware.box	24/05/2023 14:03	Fichier BOX	5 551 207 Ko

IX. Créer une machine virtuelle avec Vagrant

À partir de notre box Vagrant "packer_WS2022_vmware.box", on va pouvoir générer une nouvelle machine virtuelle VMware. Pour commencer, toujours dans le répertoire "W:\Packer\VM-WS2022", on va créer le fichier ci-dessous pour déclarer la VM à générer :

```
vagrantFile
```

Remarque : ce fichier peut aussi être généré avec la commande "**vagrant init**" qui permet d'obtenir un nouveau fichier `vagrantFile` type. Mais là, je vais vous fournir le code à intégrer.

Bien sûr, ce fichier ne va pas rester vide ! On va ajouter du contenu dans ce fichier. Voici le bout de code que je vous propose d'utiliser :

```
Vagrant.configure("2") do |config|
  config.vm.box = "packer_WS2022_vmware.box"
  # Vagrant n'a plus besoin du plugin WinRM (c'est pris en charge nativement)
  config.vm.communicator = "winrm"
  config.winrm.username = "Administrateur"
  config.winrm.password = "P@ssword!"
  config.vm.provider "vmware_workstation" do |vmware|
    vmware.gui = true
  end
end
```

Quelques explications s'imposent :

- **Vagrant.configure("2") do |config|** : déclaration d'une configuration Vagrant en version 2
 - [Documentation Vagrant - Configuration Version](#)
- **config.vm.box** : nom de la box à utiliser ; ici on utilise une box locale, mais il existe aussi des box disponibles sur Internet
- **config.vm.communicator** : protocole à utiliser pour se connecter à la VM, ici WinRM
- **config.winrm.username** : nom d'utilisateur pour se connecter en WinRM (on réutilise le compte créé via Packer)
- **config.winrm.password** : le mot de passe de ce compte
- **config.vm.provider "vmware_workstation"** : déclaration du nom du fournisseur

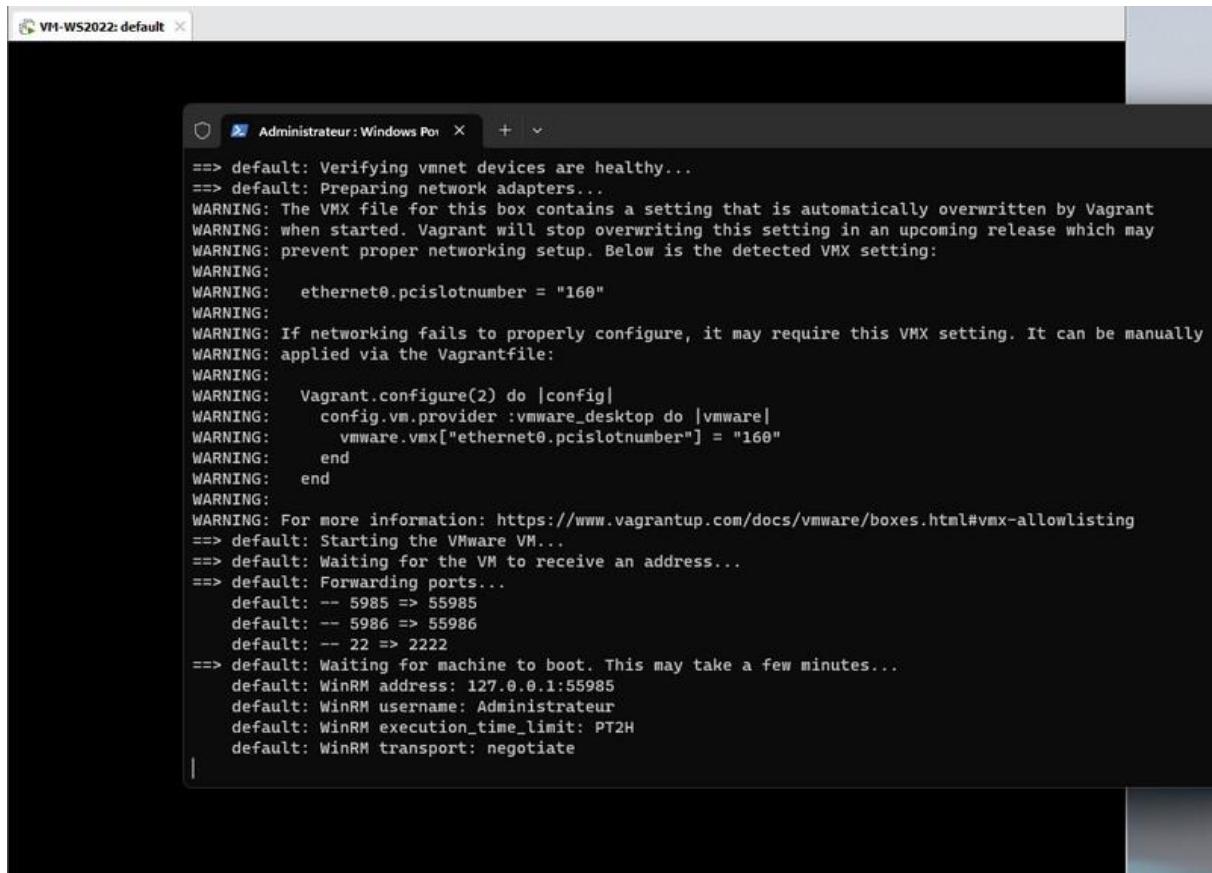
Pour récupérer directement le fichier `vagrantFile` :

- [GitHub - Fichier vagrantFile](#)

Actuellement, Vagrant n'a aucune box dans son inventaire. Celle spécifiée dans notre fichier `vagrantFile` sera ajoutée au moment de la création de la VM. D'ailleurs, on va créer cette machine virtuelle sans plus attendre avec cette commande :

```
PS W:\Packer\VM-WS2022> vagrant up --provider vmware_workstation
```

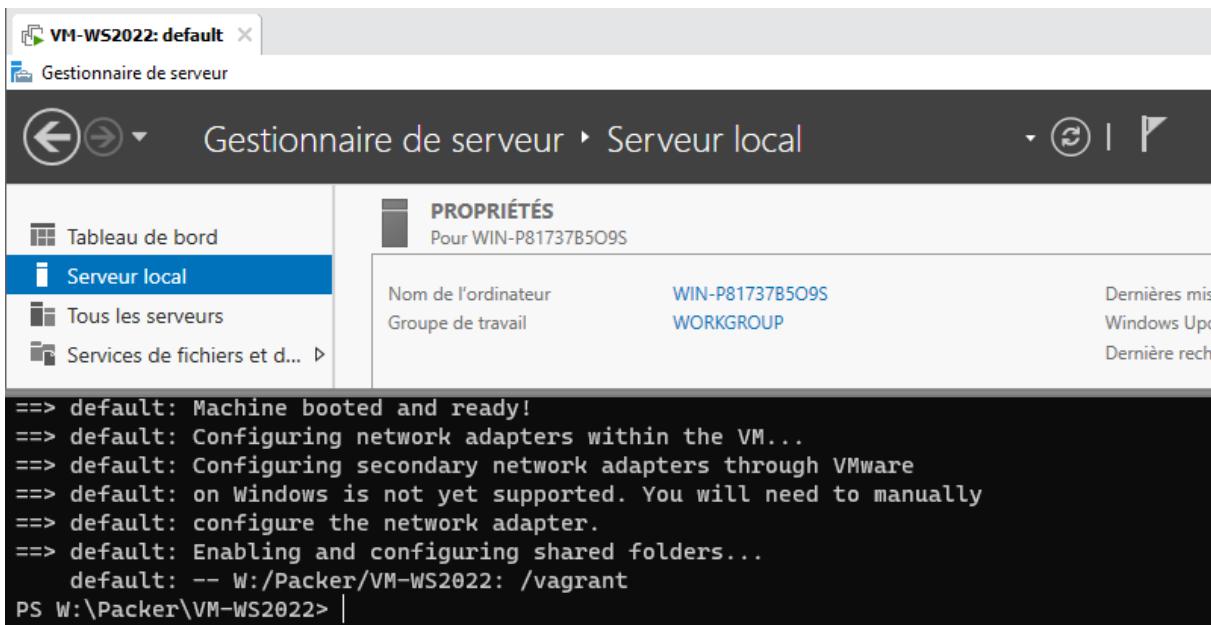
Suite à l'exécution de cette commande, une nouvelle machine virtuelle va apparaître dans VMware Worskstation ! Cette VM sera basée sur notre box créée avec Packer ! Il suffit de patienter quelques secondes...



```
>>> default: Verifying vmnet devices are healthy...
>>> default: Preparing network adapters...
WARNING: The VMX file for this box contains a setting that is automatically overwritten by Vagrant
WARNING: when started. Vagrant will stop overwriting this setting in an upcoming release which may
WARNING: prevent proper networking setup. Below is the detected VMX setting:
WARNING:
WARNING:   ethernet0.pcislotnumber = "160"
WARNING:
WARNING: If networking fails to properly configure, it may require this VMX setting. It can be manually
WARNING: applied via the Vagrantfile:
WARNING:
WARNING:   Vagrant.configure(2) do |config|
WARNING:     config.vm.provider :vmware_desktop do |vmware|
WARNING:       vmware.vmx["ethernet0.pcislotnumber"] = "160"
WARNING:     end
WARNING:   end
WARNING:
WARNING: For more information: https://www.vagrantup.com/docs/vmware/boxes.html#vmx-allowlisting
=>>> default: Starting the VMware VM...
=>>> default: Waiting for the VM to receive an address...
=>>> default: Forwarding ports...
    default: -- 5985 => 55985
    default: -- 5986 => 55986
    default: -- 22 => 2222
=>>> default: Waiting for machine to boot. This may take a few minutes...
    default: WinRM address: 127.0.0.1:55985
    default: WinRM username: Administrateur
    default: WinRM execution_time_limit: PT2H
    default: WinRM transport: negotiate
```

Voilà, la machine virtuelle Windows Server 2022 est déployée ! Elle est prête à l'emploi, avec PowerShell 7 installé et les VMware Tools installées également : conformément à ce qui a été fait en amont avec Packer. La VM créée est stockée à cet endroit :

```
W:\Packer\VM-WS2022\.vagrant\machines\default\vmware_workstation
```



La VM peut être arrêtée avec VMware Workstation, mais aussi avec Vagrant :

```
vagrant halt
==> default: Attempting graceful shutdown of VM...
```

Pour supprimer la VM depuis Vagrant, on utilisera la commande ci-dessous (qui requiert une validation) :

```
vagrant destroy
```

Même si l'on supprime la VM, **la box Vagrant est toujours enregistrée dans l'inventaire**. On peut le vérifier avec la commande ci-dessous :

```
vagrant box list
```

```
PS W:\Packer\VM-WS2022> vagrant box list
packer_WS2022_vmware.box (vmware_desktop, 0)
```

Si l'on génère une nouvelle version VM, la box enregistrée sera réutilisée.

De ce fait, si on crée **une nouvelle version de la box (*je dis bien de la box*) avec Packer** (pour ajouter un autre logiciel, par exemple), il faudra **supprimer la box enregistrée dans Vagrant et la réinscrire** (ce qui sera fait au moment du *vagrant up* même si l'on peut faire aussi un *vagrant box add*). Sinon, on générera une nouvelle version de notre box, mais Vagrant ne l'utilisera pas.

Pour supprimer une box Vagrant, il suffit de spécifier son nom :

```
vagrant box remove packer_WS2022_vmware.box
```

Ensuite, si l'on relance la création de la VM, la nouvelle version de la box Packer sera importée et utilisée. En attendant, profitez d'utiliser la VM nouvellement créée pour faire vos tests !