COM & COMPANY

Création de serveurs Master-Slave

Avec Packer et Vagrant

Allan CHAPUIS
03/11/2020

Création de serveurs Master-Slave de réplication avec Packer et Vagrant

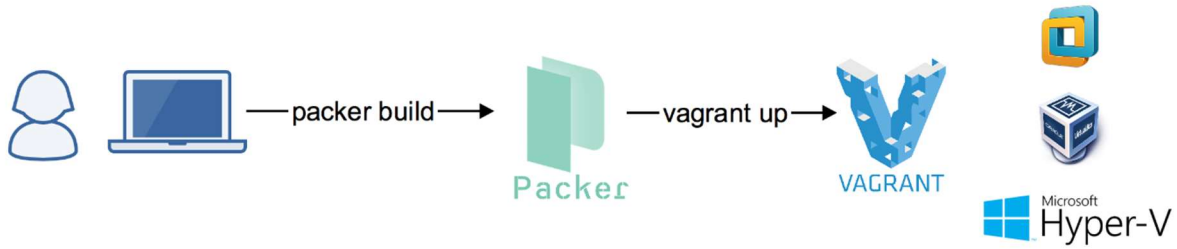


Table des matières

Introduction 2
C'est quoi Packer ? 2
Pourquoi utiliser Packer ? 2
Installation de Packer 3
Installation du binaire 3
Ajout à la variable d'environnement 4
Utilisation de Packer 7
Création de notre image 7
La section variable 9
La section builders 9
La section provisioners 10
La section post-processors 10
Validation et création du template 11
Vagrant 12
C'est quoi Vagrant ? 12
Installation de Vagrant 12
Utilisation de Vagrant 12
Initialisation 12
Modification du fichier Vagrantfile 13
Déploiement des serveurs 16
Test de la réplication des serveurs 17
Conclusion 18

Introduction

C’est quoi Packer ?

Packer est un outil open source développé par HashiCorp qui permet de créer des images machine pour plusieurs plates-formes à partir d'une configuration source unique. Il reste un outil léger qui fonctionne sur tous les principaux systèmes d'exploitation. Il crée donc essentiellement des images prêtes à l'emploi avec le système d'exploitation et les logiciels supplémentaires.

Pourquoi utiliser Packer ?

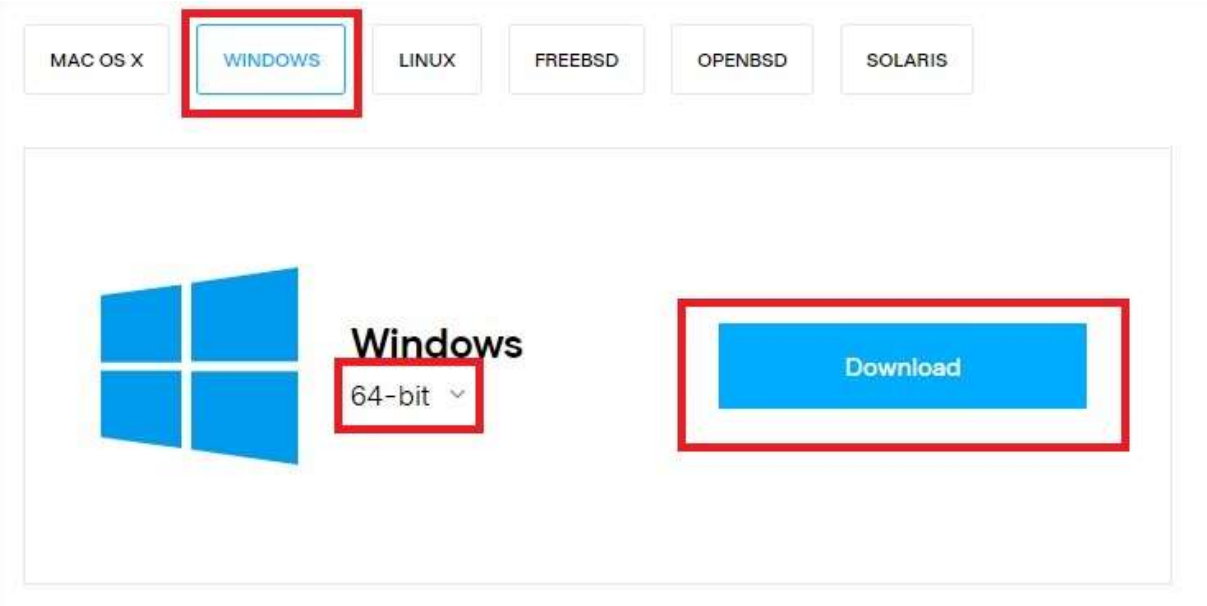
Voici une liste des avantages offerts par l'outil Packer :

- **Rapidité :** une fois vos images créées depuis Packer, vous pouvez les utiliser sur des machines qui seront provisionnées entièrement et configurées en quelques secondes, plutôt que plusieurs minutes ou heures sur les autres méthodes (ex: user-data sur AWS)
- **Portabilité** : packer vous permet de créer plusieurs images machine sur plusieurs environnements, ex : AWS, cloud privé (ex: OpenStack) et des solutions de virtualisation comme VMware ou VirtualBox, etc...
- **Stabilité :** packer installe et configure tous les logiciels d'une machine au moment de la création de l'image. Donc s'il existe des bugs dans vos scripts, ils seront alors aussi tôt détectés, avant même la création de votre infrastructure.
- **Tests :** une fois qu'une image machine est créée, elle peut être rapidement lancée et testée pour vérifier que vos tâches fonctionnent correctement. Si tel est le cas, vous pouvez être sûr que toutes les autres machines lancées à partir de cette image fonctionneront correctement et rapidement.
- **Facilité :** vous pouvez créer des images depuis Packer à partir d'un seul fichier de configuration JSON très simple à configurer.

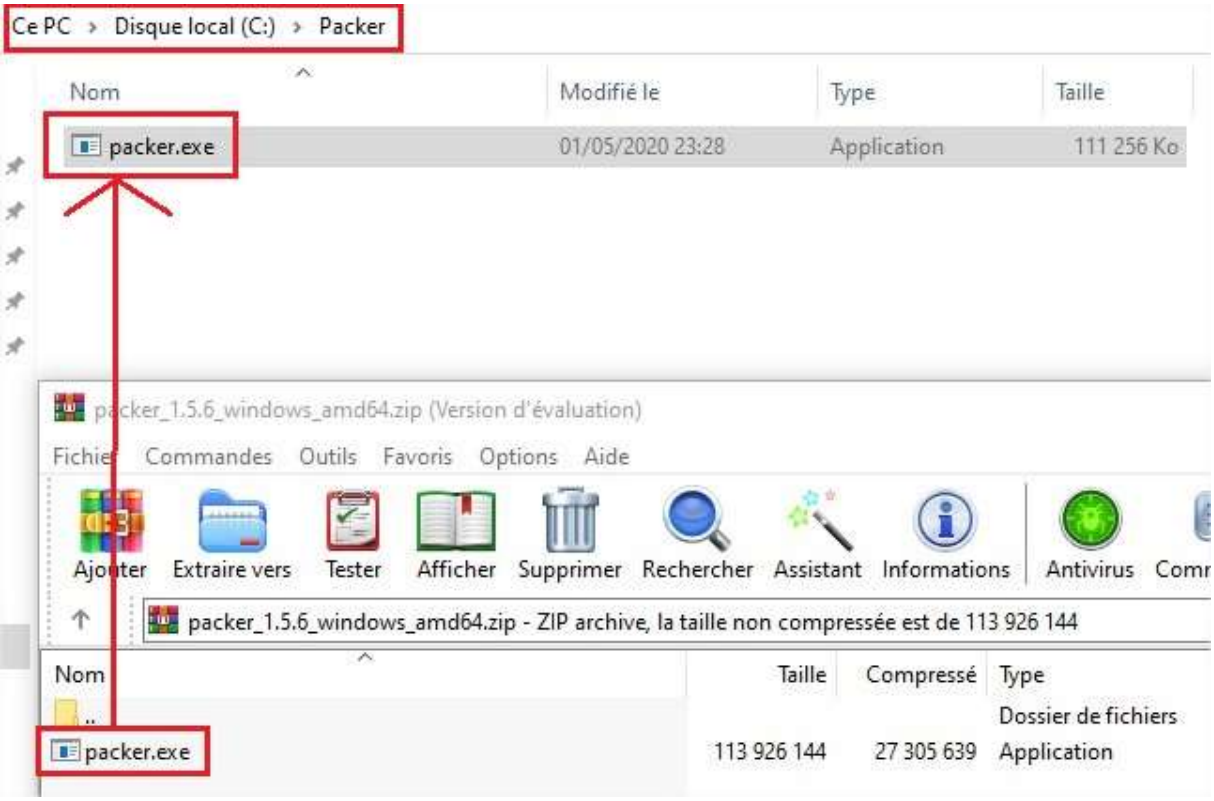
Installation de Packer

Installation du binaire

Tout d'abord rendez-vous sur le site https://www.packer.io/downloads pour télécharger packer.



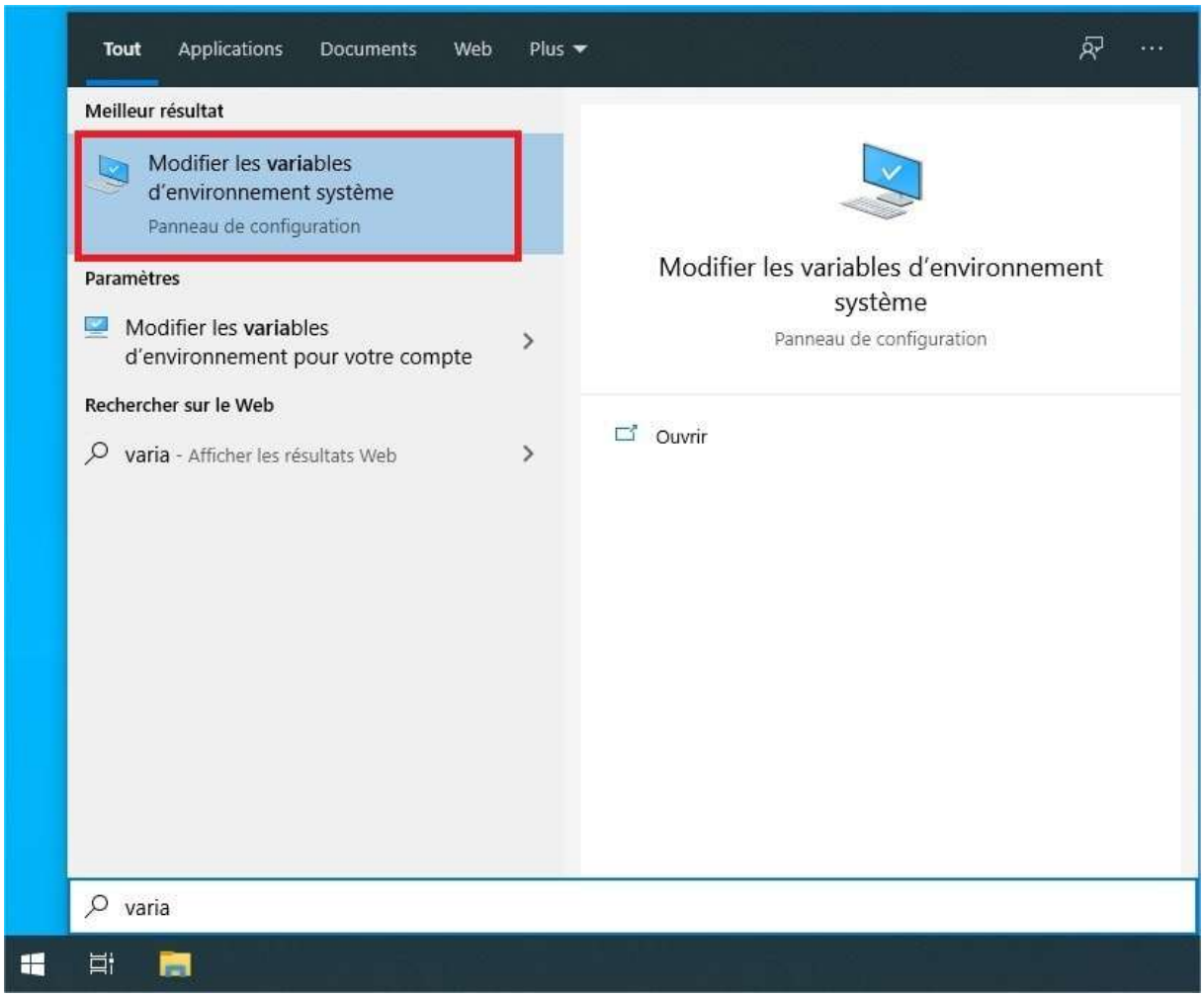
Une fois le téléchargement terminé, créez un dossier sur votre lecteur C:\ où vous placerez votre exécutable Packer. Maintenant extrayez le fichier zip téléchargé, en plaçant le binaire dans le dossier créé.



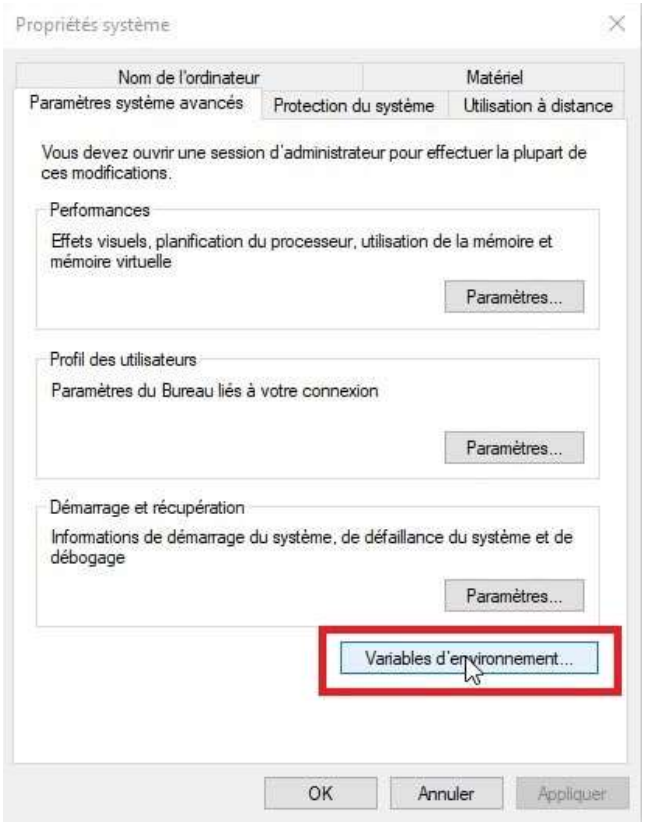
Ajout à la variable d’environnement

Maintenant il suffit de rajouter votre exécutable Packer à votre variable d'environnement nommée PATH, c'est la variable système utilisée par Windows pour localiser les fichiers exécutables, ainsi vous pourrez exécuter le programme Packer depuis n'importe où en ligne de commande. Pour cela, suivez les étapes suivantes :

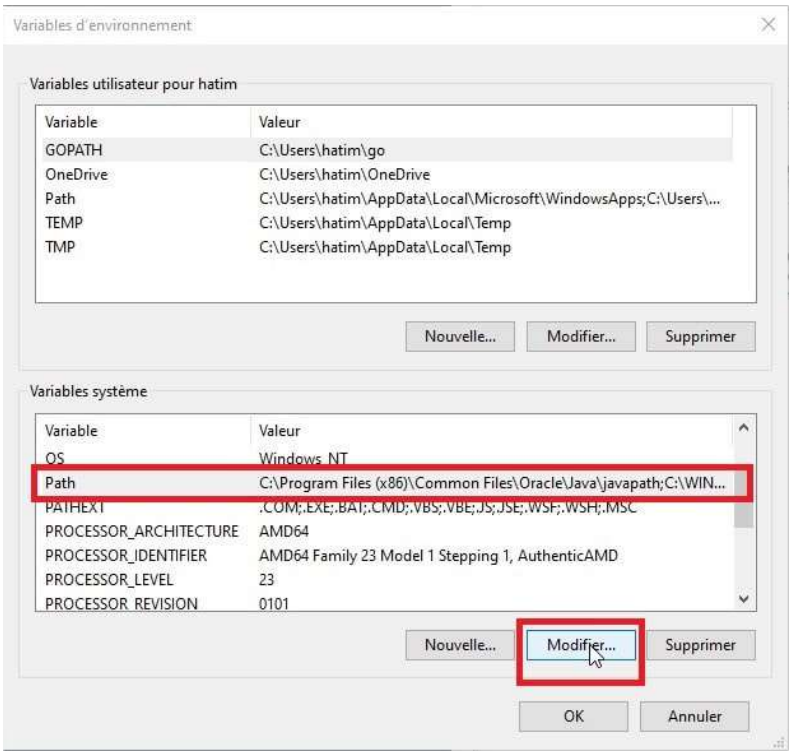
Commencez par ouvrir votre menu Démarrer et tapez "environnement" et la première chose qui apparaît devrait être "Modifier les variables d'environnement système".



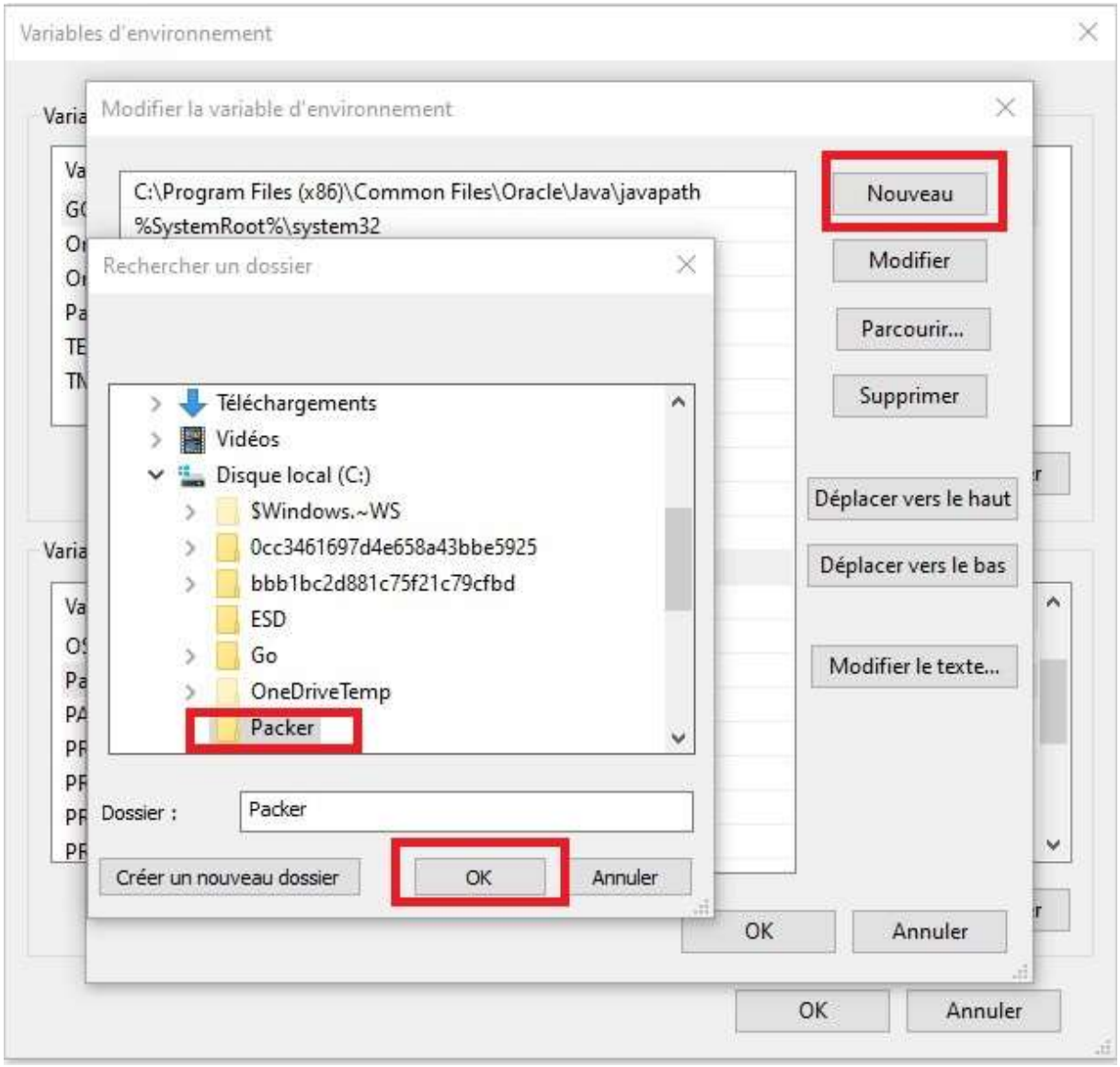
Cliquez dessus et vous devriez voir cette fenêtre, cliquez ensuite sur le bouton "Variables d'environnement" :



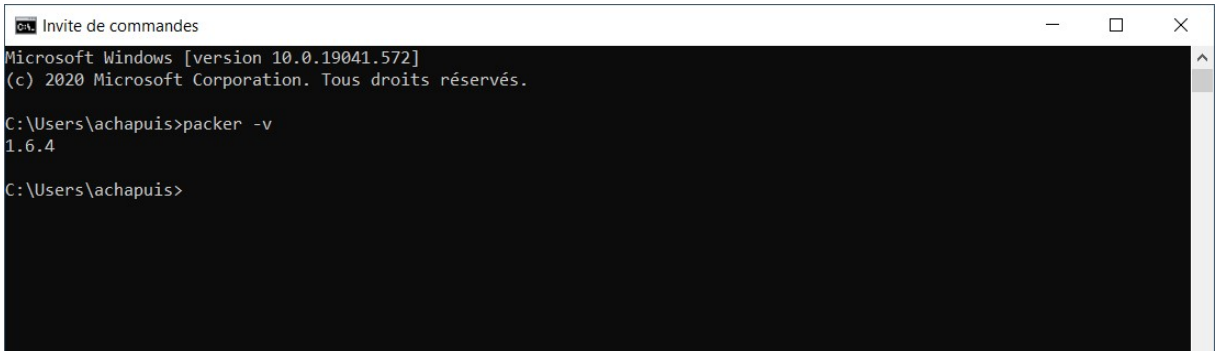
Dans la section inférieure où il est indiqué "variables système", recherchez la variable nommée PATH et cliquez sur modifier :



Cliquez ensuite sur le bouton "Parcourir" et ajoutez le chemin du dossier où se trouve le binaire *packer.exe* :



Voilà maintenant nous pouvons utiliser Packer dans le cmd et le PowerShell. Pour vérifier si Packer s'est installé avec succès vous pouvez ouvrir un terminal et lancez la commande :



Utilisation de Packer

Création de notre image

Dans la terminologie Packer, le fichier de configuration utilisé pour définir une image dans Packer est appelé "template", ce template devra être sous le format JSON. Ici mon template est pour le provider VirtualBox.

Voici d'abord le code de mon template, puis nous passerons en revus chaque section.

```
{
    "_comment": "Build with `packer build ubuntu.json`",
    "variables": {
        "cpus": "2",
        "disable_ipv6": "true",
        "disk_size": "32768",
        "headless": "true",
        "http_proxy": "{{env `http_proxy`}}",
        "https_proxy": "{{env `https_proxy`}}",
        "iso_checksum": "443511f6bf12402c12503733059269a2e10dec602916c0a75263e
5d990f6bb93",
        "iso_name": "ubuntu-20.04.1-live-server-amd64.iso",
        "iso_url": "https://releases.ubuntu.com/focal/ubuntu-20.04.1-live-
server-amd64.iso",
        "memory": "2048",
        "no_proxy": "{{env `no_proxy`}}",
        "ssh_password": "vagrant",
        "ssh_username": "vagrant",
        "update": "true",
        "vagrantfile_template": "",
        "version": "0.1",
        "virtualbox_guest_os_type": "Ubuntu_64",
        "vm_name": "ubuntu-20.04"
    },
    "builders": [{
        "boot_command": [
            "<enter><enter><f6><esc><wait>",
            "autoinstall ds=nocloud-net;s=http://{{.HTTPIP}}:{{.HTTPPort}}/",
            "<enter>"
        ],
        "boot_wait": "5s",
        "disk_size": "{{user `disk_size`}}",
        "guest_additions_path": "VBoxGuestAdditions_{{.Version}}.iso",
        "guest_os_type": "{{user `virtualbox_guest_os_type`}}",
        "hard_drive_interface": "sata",
        "headless": "{{user `headless`}}",
        "http_directory": "http",
```

```
        "iso_checksum": "{{user `iso_checksum`}}",
        "iso_urls": [
            "iso/{{user `iso_name`}}",
            "{{user `iso_url`}}"
        ],
        "output_directory": "output-{{user `vm_name`}}-virtualbox-iso",
        "shutdown_command": "echo '{{user `ssh_password`}}'|sudo -S shutdown -
P now",
        "ssh_handshake_attempts": "20",
        "ssh_password": "{{user `ssh_password`}}",
        "ssh_username": "{{user `ssh_username`}}",
        "ssh_pty": true,
        "ssh_timeout": "1h",
        "type": "virtualbox-iso",
        "vboxmanage": [
            ["modifyvm", "{{.Name}}", "--audio", "none"],
            ["modifyvm", "{{.Name}}", "--usb", "off"],
            ["modifyvm", "{{.Name}}", "--vram", "16"],
            ["modifyvm", "{{.Name}}", "--vrde", "off"],
            ["modifyvm", "{{.Name}}", "--nictype1", "virtio"],
            ["modifyvm", "{{.Name}}", "--memory", "{{user `memory`}}"],
            ["modifyvm", "{{.Name}}", "--cpus", "{{user `cpus`}}"]
        ],
        "virtualbox_version_file": ".vbox_version",
        "vm_name": "{{user `vm_name`}}"
    }],
    "provisioners": [{
        "environment_vars": [
            "DEBIAN_FRONTEND=noninteractive",
            "UPDATE={{user `update`}}",
            "DISABLE_IPV6={{user `disable_ipv6`}}",
            "SSH_USERNAME={{user `ssh_username`}}",
            "SSH_PASSWORD={{user `ssh_password`}}",
            "http_proxy={{user `http_proxy`}}",
            "https_proxy={{user `https_proxy`}}",
            "no_proxy={{user `no_proxy`}}"
        ],
        "execute_command": "echo '{{user `ssh_password`}}'|{{.Vars}} sudo -E -
S bash '{{.Path}}'",
        "expect_disconnect": true,
        "scripts": [
            "script/update.sh",
            "script/vagrant.sh",
            "script/virtualbox.sh",
            "script/motd.sh",
            "script/minimize.sh",
            "script/cleanup.sh"
        ],
        "type": "shell"
```

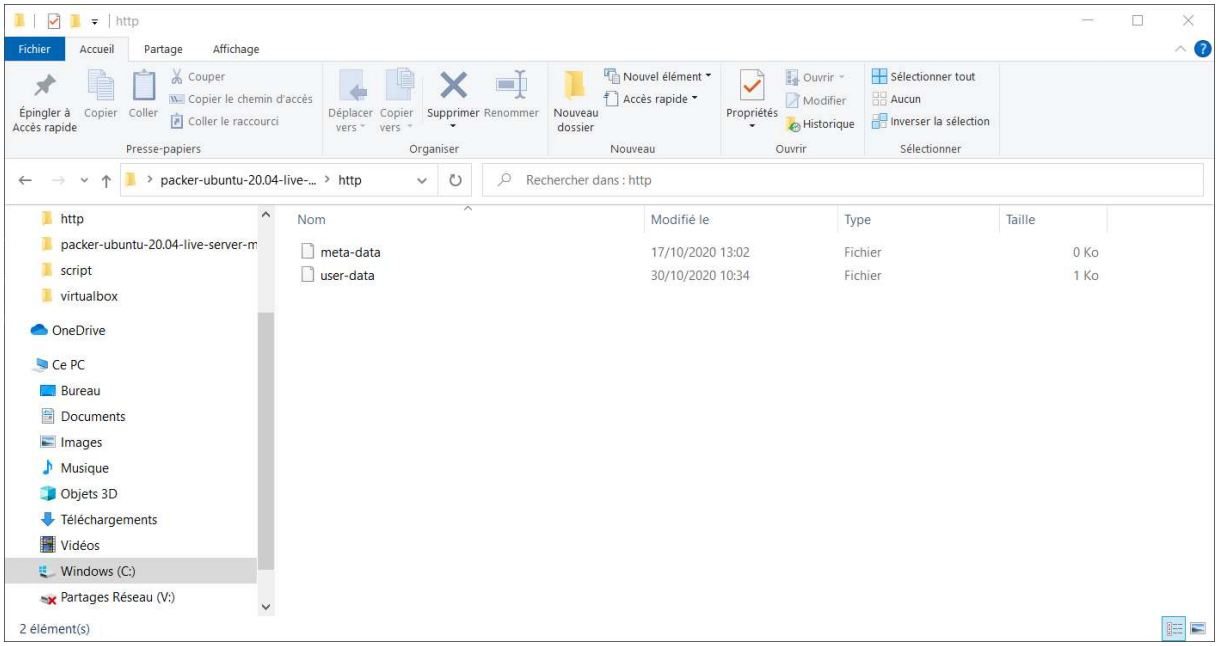
```
    }],
    "post-processors": [{
        "keep_input_artifact": false,
        "output": "box/{{.Provider}}/{{user `vm_name`}}-
{{user `version`}}.box",
        "type": "vagrant",
        "vagrantfile_template": "{{user `vagrantfile_template`}}"
    }]
}
```

La section variable

C'est dans cette section que nous déclarerons toutes les variables utilisés dans le code de notre template. Ici certaines variables sont spécifiques à VirtualBox. Pour utiliser une variable il suffit d'écrire : {{user `*nom_variable*`}} avec le nom de la variable déclarer à la place de *nom_variable.*

La section builders

Dans la section builders nous trouvons la configuration principale pour la construction de notre image. C'est ici que nous mettons les commandes qui sont exécuter lorsque l'image boot. Pour ma part, je fais l'installation d'Ubuntu grâce à subiquity, c'est le nouvel installeur pour les serveurs. Je créé donc un fichier appelé user-data dans un dossier http et avec un fichier met-data qui doit être vide.



Et dans le **user-data** je mets ceci :

```
#cloud-config
autoinstall:
  version: 1
  early-commands:
    - 'sudo systemctl stop ssh'
  network:
    network:
      version: 2
      ethernets:
        enp0s3:
          dhcp4: true
  identity:
    hostname: ubuntu
    username: vagrant
    password: $6$r0.SCOWKMTrHIpz$JCnAEUhC6P5RuZAihFyITQztGyaydcNMNWwp0DhmpHZFo
WAIgd7Z8Yup.fVmivsbqYbYg0Do47uzY7s1m0SOR.
  ssh:
    install-server: true
  user-data:
    timezone: Europe/Paris
```

Ce type de fichier et plus facile d'utilisation que le fichier de configuration « *preseed.cfg* » couramment utilisé.

Pour continuer dans la section builders, d'autres paramètres permettent de renseigner l'endroit où se situe l'ISO (ici d'Ubuntu20.04 live server), de spécifié les différents paramètres pour VirtualBox comme le cpu, la ram, la taille du disque, etc…

La section provisioners

Dans cette section nous allons pouvoir exécuter des commandes directement sur la machine pour par exemple configurer des paramètres ou mettre à jour la machine. Ici j'utilise plusieurs scripts que j'ai placer dans un dossier nommé scripts. Ces scripts permettent de mettre à jours et de nettoyer les applications qui ne servent pas pour la création du serveur. Vous pouvez en fonctions de vos besoins lancer des scripts.

La section post-processors

Elle permet de spécifier le format du template générer, ici j'ai choisi de générer un template sous la forme d'un *.box*(utilisé pour VirtualBox), dans un dossier appelé box.

Validation et création du template

Avant d'exécuter notre template et de créer notre image à partir de celui-ci, validons d'abord la configuration de notre template packer en exécutant la commande ***packer validate***. Cette commande vérifiera la syntaxe ainsi que les valeurs de configuration fournies afin de s'assurer de leurs fiabilités. La sortie doit ressembler à ceci :

```
packer validate ubuntu.json
```

Résultat :

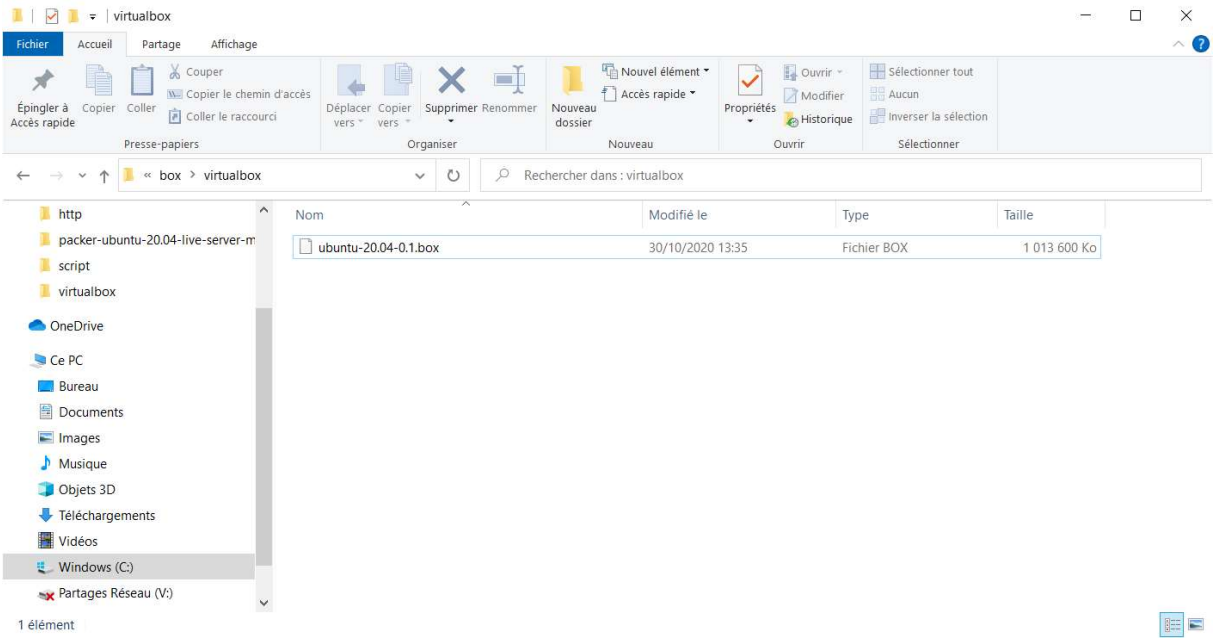
```
Template validated successfully.
```

Maintenant pour construire le template il suffit de taper la commande ***packer build*** suivi du fichier template. La sortie devrait ressembler à celle ci-dessous (notez que ce processus prend généralement quelques minutes) :

```
packer build ubuntu.json
```

Maintenant si vous allez dans le dossier box vous devez retrouver votre template d'Ubuntu serveur avec l'extension *.box.*

Maintenant il nous faut utiliser ce template ! Nous allons le faire grâce à Vagrant.



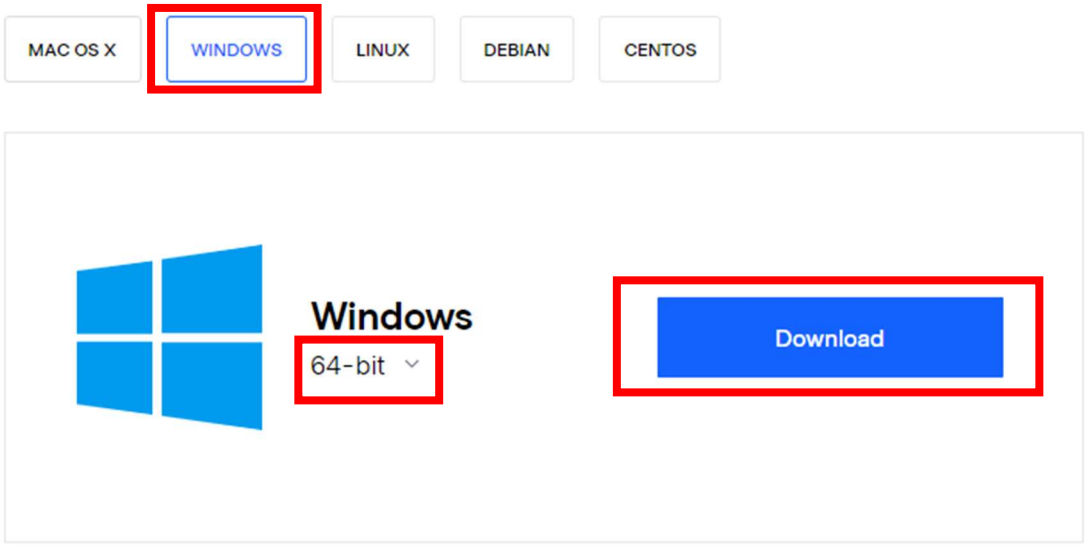
Vagrant

C’est quoi Vagrant ?

Vagrant est un outil permettant de créer des machines virtuelles pendant le développement de votre application afin d’obtenir l’environnement souhaité sans pour autant changer la configuration de votre machine.

Installation de Vagrant

Rendez-vous sur le site https://www.vagrantup.com/downloads.html, puis vous n’avez plus qu’à lancer le fichier téléchargé.



Utilisation de Vagrant

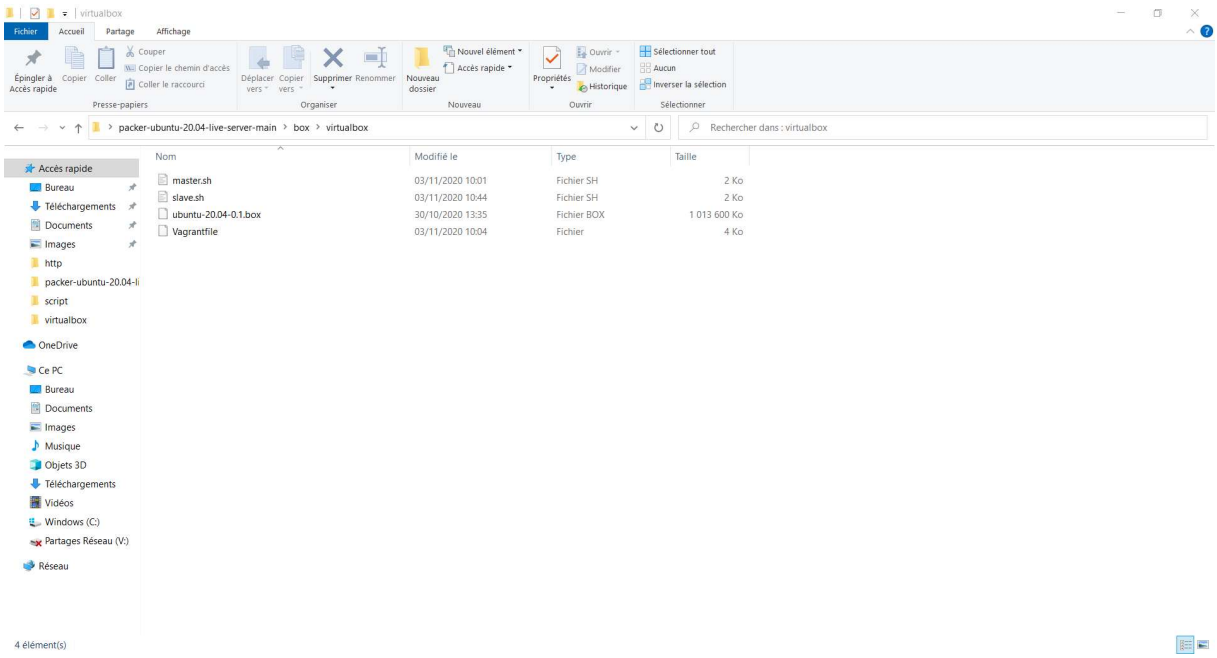
Initialisation

Maintenant que nous avons télécharger Vagrant, nous allons ouvrir le cmd et allons dans le dossier ou se situe le template avec l’extension *.box* précédemment créé. Et nous taperons la commande :

```
vagrant init ubuntu-20.04-0.1.box
```

Après avoir fait cette commande nous trouverons un nouveau fichier qui a été générer nommé « *Vagrantfile* ».

Vous pouvez maintenant ouvrir ce fichier pour pouvoir le modifier.



Modification du fichier Vagrantfile

Ce fichier est codé en Ruby, voici le contenu de mon fichier :

```
Vagrant.configure("2") do |config|
  config.vm.define "server1" do |srv1|
    srv1.vm.box = ".\\ubuntu-20.04-0.1.box"
    srv1.vm.hostname = 'ubuntu1'
    srv1.vm.network :public_network, ip: "192.168.1.98"
    srv1.vm.provision :shell, :path => "master.sh"
  end
  config.vm.define "server2" do |srv2|
    srv2.vm.box = ".\\ubuntu-20.04-0.1.box"
    srv2.vm.hostname = 'ubuntu2'
    srv2.vm.network :public_network, ip: "192.168.1.99"
    srv2.vm.provision :shell, :path => "slave.sh"
  end

end
```

Grâce à vagrant nous pouvons ainsi créer deux machines virtuelles, ici ubuntu1 et ubuntu2. Puis je leur défini une IP fixe. Et enfin grâce au paramètre « provision » je peux exécuter un script bash pour chaque serveur.

Pour configurer le Master voici le script que je créé dans le même dossier que le Vagrantfile :

```
#!/usr/bin/bash

# update
sudo apt update -y

# install mariadb
sudo apt-get install -y software-properties-common
sudo apt-key adv --fetch-
keys 'https://mariadb.org/mariadb_release_signing_key.asc'
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el] https://mirrors.n-
ix.net/mariadb/repo/10.5/ubuntu focal main'
sudo apt update -y
sudo apt install -y mariadb-server

# ensure it is running
sudo /etc/init.d/mysql restart

# set to auto start
sudo systemctl status mariadb

### post-install setup

# set root password
sudo /usr/bin/mysqladmin -u root password 'password'

# allow remote access (required to access from our private network host. Note
that this is completely insecure if used in any other way)
mysql -u root -ppassword -
e "GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'password' WITH GRA
NT OPTION; FLUSH PRIVILEGES;"

# restart
sudo systemctl restart mariadb

# creation du master
sed -i -e "s/127.0.0.1/0.0.0.0/g" /etc/mysql/mariadb.conf.d/50-server.cnf

echo 'server-id = 1' >> /etc/mysql/mariadb.conf.d/50-server.cnf
echo 'log_bin = /var/log/mysql/mysql-bin.log' >> /etc/mysql/mariadb.conf.d/50-
server.cnf
echo 'log_bin_index =/var/log/mysql/mysql-
bin.log.index' >> /etc/mysql/mariadb.conf.d/50-server.cnf
echo 'relay_log = /var/log/mysql/mysql-relay-
bin' >> /etc/mysql/mariadb.conf.d/50-server.cnf
echo 'relay_log_index = /var/log/mysql/mysql-relay-
bin.index' >> /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
# restart
sudo systemctl restart mariadb

mysql -u root -ppassword -
e "CREATE USER 'replication'@'%' identified by 'password'; GRANT REPLICATION S
LAVE ON *.* TO 'replication'@'%'; FLUSH PRIVILEGES;"
```

Et pour configurer le Slave:

```
#!/usr/bin/bash

# update
sudo apt update -y

# install mariadb
sudo apt-get install -y software-properties-common
sudo apt-key adv --fetch-
keys 'https://mariadb.org/mariadb_release_signing_key.asc'
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el] https://mirrors.n-
ix.net/mariadb/repo/10.5/ubuntu focal main'
sudo apt update -y
sudo apt install -y mariadb-server

# ensure it is running
sudo /etc/init.d/mysql restart

# set to auto start
sudo systemctl status mariadb

### post-install setup

# set root password
sudo /usr/bin/mysqladmin -u root password 'password'

# allow remote access (required to access from our private network host. Note
that this is completely insecure if used in any other way)
mysql -u root -ppassword -
e "GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'password' WITH GRA
NT OPTION; FLUSH PRIVILEGES;"

# restart
sudo systemctl restart mariadb

# creation du slave
sed -i -e "s/127.0.0.1/0.0.0.0/g" /etc/mysql/mariadb.conf.d/50-server.cnf

echo 'server-id = 2' >> /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
echo 'log_bin = /var/log/mysql/mysql-bin.log' >> /etc/mysql/mariadb.conf.d/50-
server.cnf
echo 'log_bin_index =/var/log/mysql/mysql-
bin.log.index' >> /etc/mysql/mariadb.conf.d/50-server.cnf
echo 'relay_log = /var/log/mysql/mysql-relay-
bin' >> /etc/mysql/mariadb.conf.d/50-server.cnf
echo 'relay_log_index = /var/log/mysql/mysql-relay-
bin.index' >> /etc/mysql/mariadb.conf.d/50-server.cnf

# restart
sudo systemctl restart mariadb

# show master status file et position
MASTER_STATUS=$(mariadb -h 192.168.1.98 "-uroot" "-ppassword" -
ANe "SHOW MASTER STATUS;" | awk '{print $1 " " $2}')
LOG_FILE=$(echo $MASTER_STATUS | cut -f1 -d ' ')
LOG_POS=$(echo $MASTER_STATUS | cut -f2 -d ' ')
mysql -u root -ppassword -
e "stop slave; CHANGE MASTER TO MASTER_HOST = '192.168.1.98', MASTER_USER = 'r
eplication', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = '$LOG_FILE', MAST
ER_LOG_POS = $LOG_POS; start slave;"
```

Déploiement des serveurs

Maintenant que nous avons tous ceux qu'il faut pour déployer les serveurs nous allons lancer sur le PowerShell la commande :

```
vagrant up
```

Et voilà vous n'avez plus qu'à attendre. Quand tout sera fini vous pourrez maintenant vous connecter en ssh sur n'importe quel serveur. Vous pouvez utiliser une option de vagrant pour vous connecter très facilement en ssh via la commande :

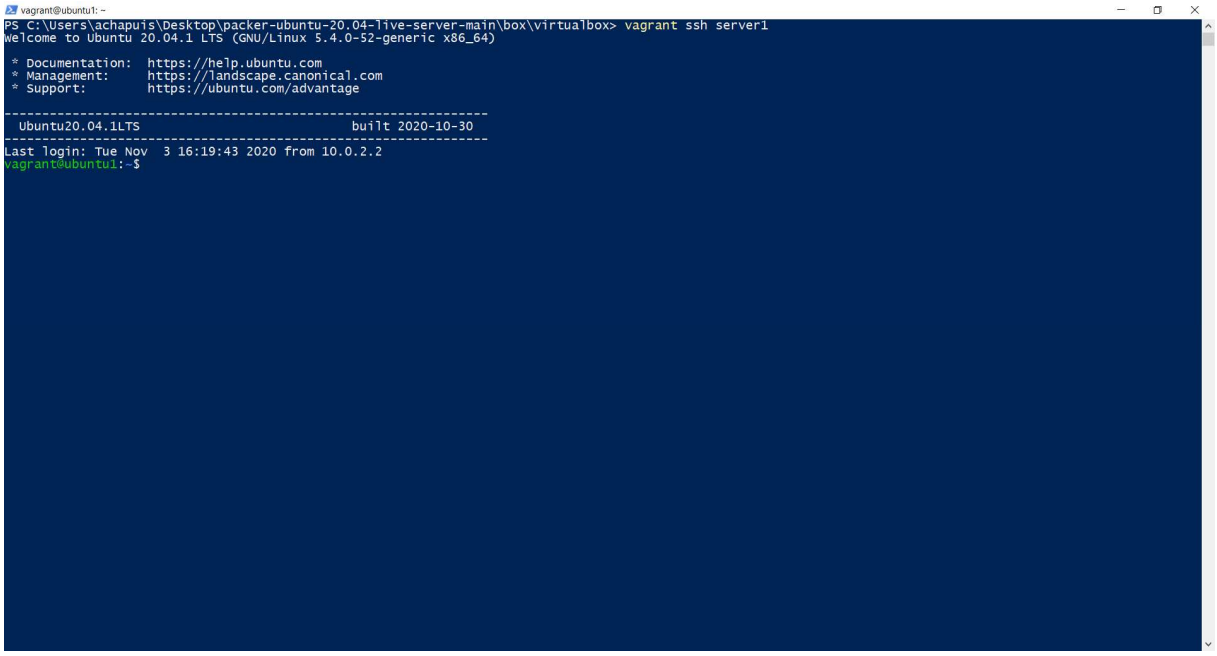
```
vagrant ssh server1
```

Ou pour le deuxième serveur avec la commande :

```
vagrant ssh server2
```

Test de la réplication des serveurs

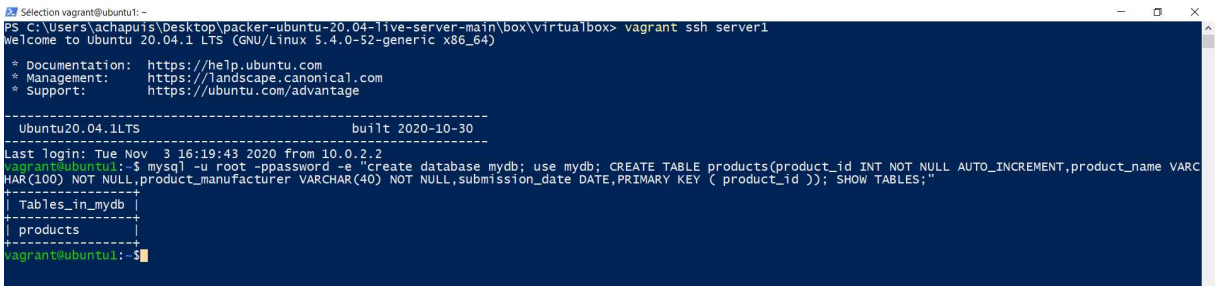
Pour tester si la réplication fonctionne nous allons nous connecter en ssh sur le serveur master grâce à la commande ***vagrant ssh server1.***



Maintenant pour vérifier le bon fonctionnement nous allons créer une base de donnée et une table fictive. Tapez la commande :

```
mysql -u root -ppassword -e "create database mydb; use mydb; CREATE TABLE products(product_id INT NOT NULL AUTO_INCREMENT,product_name VARCHAR(100) NOT NULL,product_manufacturer VARCHAR(40) NOT NULL,submission_date DATE,PRIMARY KEY ( product_id )); SHOW TABLES;"
```

Vous devriez avoir ce résultat :

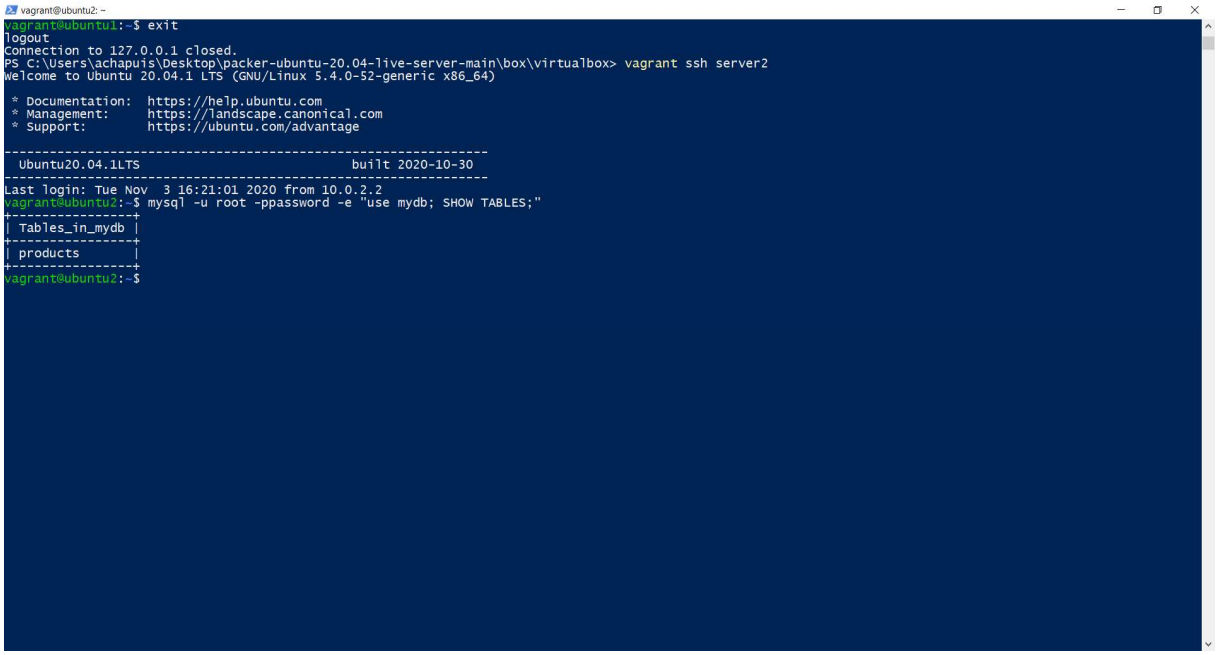


Maintenant connectons nous sur le serveur slave en quittant la connexion ssh du master en tapant ***exit***. Puis tapé la commande ***vagrant ssh server2.***

Une fois connecter sur le deuxième serveur, taper cette commande :

```
mysql -u root -ppassword -e "use mydb; SHOW TABLES;"
```

Et si cela fonctionne vous devriez voir le même tableau que précédemment.



Conclusion

Et voilà maintenant grâce à Packer et Vagrant vous savez déployer deux serveurs de réplication Master-Slave, une fois la box générée le déploiement et très facile et très rapide à reproduire.