

# \_\_ Cours Vagrant

---

Ecrit par **Youenn DUVAL**

Mail: [youenn@barbed.fr](mailto:youenn@barbed.fr)

Linkedin: [Youenn DUVAL](#)

Derniere mise à jour : **10/11/2024**

---

## Table des matières

1. [Concept](#)
2. [Objectif](#)
3. [Pourquoi utiliser des VMs plutôt que des containers ?](#)
4. [Installer Vagrant](#)
5. [Hub et Boxes](#)
6. [Les commandes de base](#)
7. [Le Vagrantfile](#)
  1. [Concept du Vagrantfile](#)
  2. [Fonctionnement d'un projet Vagrant](#)
  3. [Le plus simple possible](#)
8. [Vagrantfile minimaliste pour une VM Debian 12](#)
  1. [Démo](#)
9. [Plus complet](#)
10. [Vagrantfile pour une VM Debian 12 sous VirtualBox](#)
  1. [Plusieurs VM](#)
11. [Fichier mysql.sh](#)
12. [Créer un nouvel utilisateur et attribuer des privilèges](#)
13. [Créer une nouvelle base de données](#)
14. [Appliquer les privilèges](#)
15. [Modifier le fichier de configuration de MariaDB pour permettre les connexions externes](#)
16. [Redémarrer le service MariaDB pour appliquer les changements](#)
17. [Exemple pour une VM Docker](#)
18. [Vagrantfile pour une VM Debian 12 avec Docker depuis les dépôts Docker](#)
19. [Comment je travail avec Vagrant](#)
  1. [Préparer VScode pour qu'il se connecte dans ma VM](#)
  2. [Connecter VScode à ma VM](#)
  3. [Tips supplémentaires](#)
20. [Connaitre son provider](#)

## Concept

Vagrant est un outil open source qui permet de créer et de gérer des environnements de développement virtualisés de manière simple et reproductible. Il automatise la configuration des machines virtuelles en utilisant des scripts (comme des fichiers *Vagrantfile*), facilitant ainsi la collaboration entre développeurs et assurant la cohérence des environnements.

### Caractéristiques principales :

- Crée des environnements de développement isolés
- Utilise des systèmes comme VirtualBox, ou VMWare Desktop (et d'autres) pour manipuler de la virtualisation
- Sous Linux il est aussi possible piloter des containers Docker
- Automatisation des configurations avec des scripts
- Simplifie la gestion des dépendances et des configurations pour les projets
- Tourne autour d'un fichier yaml appelé Vagrantfile

Le cœur de Vagrant est donc de configurer rapidement des VM. Pour cela il pilote votre outil de gestion de VM et offre un store avec des VMs déjà prêtes. Vagrant appelle cela des "Boxes". Le concept est sensiblement le même que pour des images Docker. Vagrant s'occupe de télécharger ces Boxes qui peuvent être fait par le premier venu comme être des images officielles. Certaines sont des OS plus ou moins brutes, d'autres sont déjà configurées pour des tâches spécifiques.

Le produit est fait par [HashiCorp](#).

## Objectif

Vagrant peut répondre à plusieurs problématiques mais nous allons nous axer ici sur la possibilité de déployer des machines virtuelles afin de mettre en place des environnements de travail sans modifier notre machine hôte.

Si Vagrant peut monter une machine rapidement il peut également en monter plusieurs, déployer un script d'installation, paramétrer des réseaux etc.

Par exemple, si vous avez besoin d'une machine Ubuntu fonctionnelle, quelques lignes dans un fichier suffiront à en déployer une fonctionnelle en quelques minutes (dépendant aussi de votre connexion internet). La détruite et la reconstruire en 2 commandes. Changer pour un autre OS en modifiant une ligne etc.

Si vous souhaitez expérimenter du réseau ou découvrir des OS ou comprendre comment déployer des outils, Vagrant sera un outil utile.

## Pourquoi utiliser des VMs plutôt que des containers ?

- **Découverte de plusieurs systèmes d'exploitation** : Les VMs permettent de pratiquer sur différents OS (Linux, Windows, etc.) sans avoir à les installer sur la machine physique, ce qui est idéal pour apprendre à configurer et administrer des environnements variés.
- **Reproduction fidèle d'environnements réels** : Pour des cours sur l'administration système, la mise en réseau, ou la sécurité, les VMs reproduisent mieux les environnements de serveurs réels, car elles fournissent un système complet, avec un accès total aux fonctions du système d'exploitation.
- **Isolation des expérimentations** : Les étudiants peuvent expérimenter des configurations complexes ou des manipulations risquées sans affecter leur machine hôte, ce qui est essentiel pour éviter d'endommager leur propre système pendant les tests.
- **Approfondissement des notions de virtualisation** : L'utilisation des VMs permet de mieux comprendre les concepts fondamentaux de la virtualisation, des systèmes d'exploitation et de la gestion des ressources matérielles, qui ne sont pas aussi visibles avec les conteneurs.
- **Facilité de modification** : Pouvoir expérimenter en direct dans l'environnement.

Si les containers sont très pratiques pour déployer des applications, savoir comment le faire manuellement et de manière traditionnelle est tout aussi crucial ne serait-ce que pour apprendre ensuite à conteneuriser correctement des applications.

### Warning

Dans la suite de ce document tout sera orienté pour une installation sur un Windows dans le but de faire tourner une VM sous Linux. Vagrant fonctionne bien sur parfaitement bien sur Mac comme sur Linux. Je travaille également avec VirtualBox (que généralement je conseille) mais c'est également compatible avec les outils VMware comme VMware Desktop.


## Installer Vagrant

Se rendre sur la page de téléchargement de Vagrant:

[https://developer.hashicorp.com/vagrant/install?product\\_intent=vagrant](https://developer.hashicorp.com/vagrant/install?product_intent=vagrant)

On récupère le binaire

Developer / Vagrant



# Install Vagrant


2.4.1 (latest) ▾


## macOS

### Package manager

```
brew tap hashicorp/tap  
brew install hashicorp/tap/hashicorp-vagrant
```


### Binary download


AMD64  
Version: 2.4.1  
Download 

ARM64  
Version: 2.4.1  
Download 

## Windows

### Binary download

AMD64  
Version: 2.4.1  
Download 

I686  
Version: 2.4.1  
Download 

## Linux

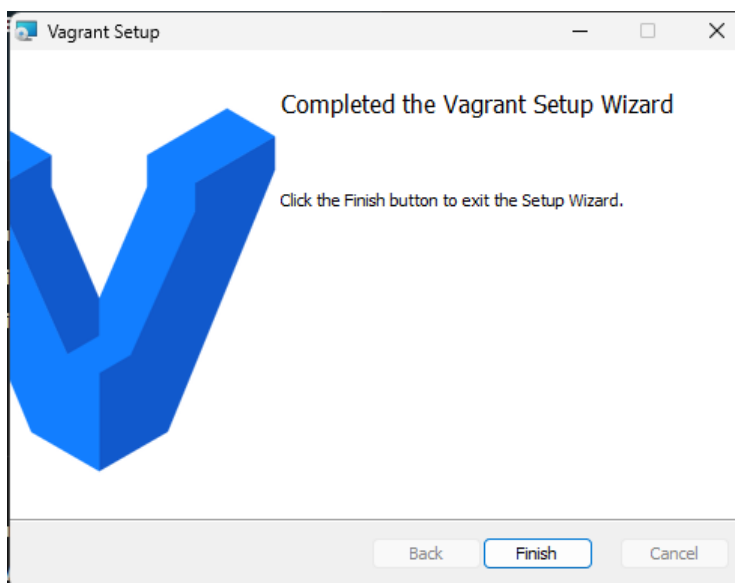
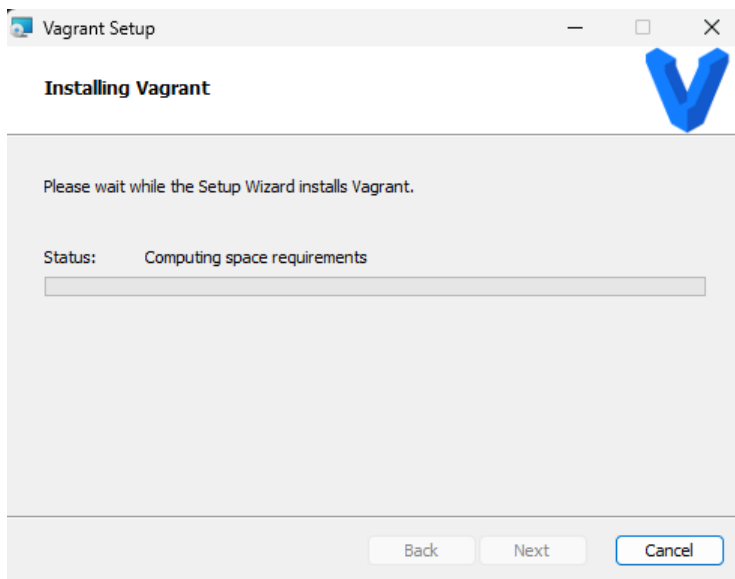
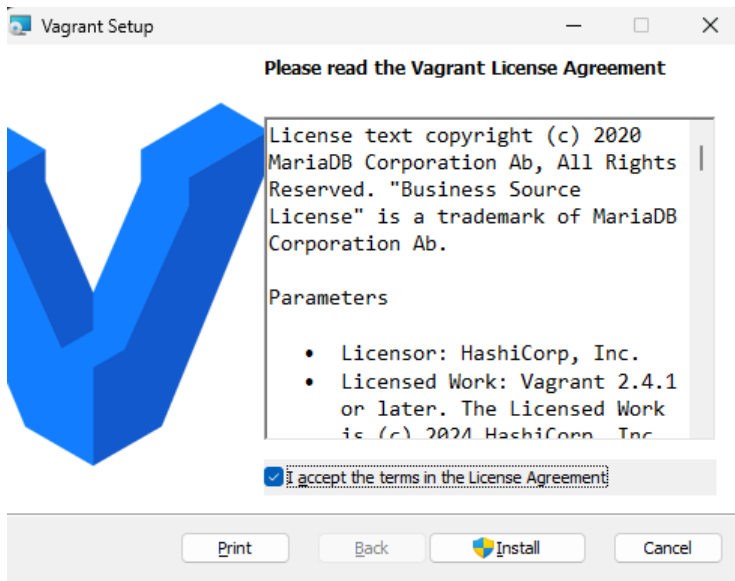
### Package manager

Ubuntu/Debian CentOS/RHEL Fedora Amazon Linux Homebrew

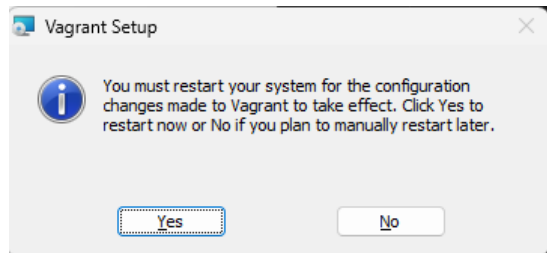
```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg  
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list  
sudo apt update && sudo apt install vagrant
```

### Binary download

On installe, attention, ça peut prendre du temps!



Il va falloir redémarrer la machine.



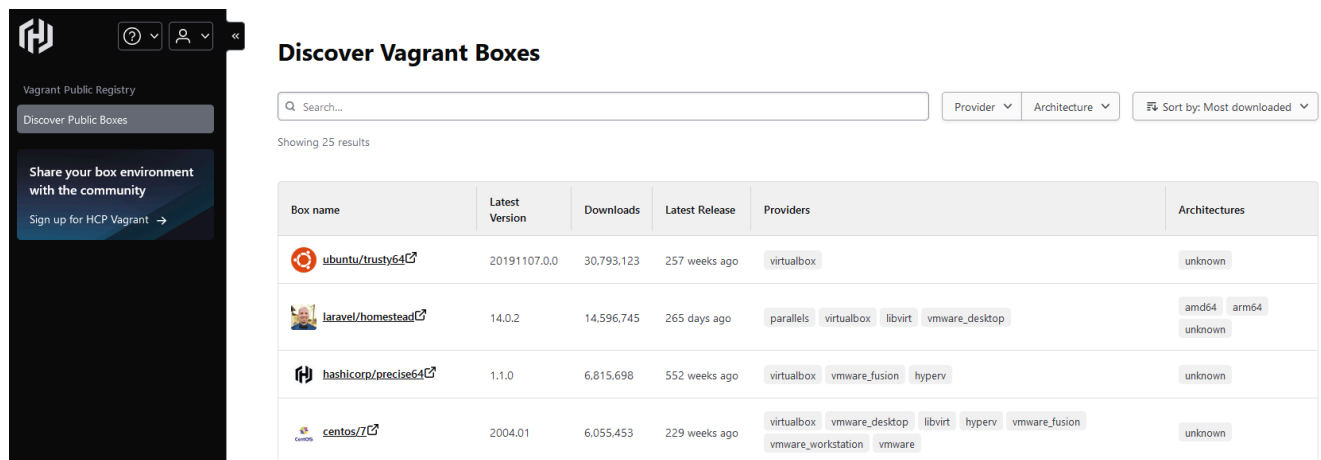
## Hub et Boxes





Comme expliqué plus tôt dans le document, Vagrant fournit un store qui héberge des Boxes. Comprendre des images de VM, ou plutôt des disques et fichier de configuration de VMs déjà configurées et prêtes à démarrer.

Démarrer un VM avec Vagrant pour la première fois va donc impliquer que ce dernier vérifie la disponibilité de la Box choisie, vérifier que nous avons la box désirée sur notre machine et dans sa dernière version. Si non, nous allons devoir la télécharger.

Il est bien sûr possible de visiter le store et d'explorer les différentes boxes pour choisir celle adaptée à notre besoin.

<https://portal.cloud.hashicorp.com/vagrant/discover>

A screenshot of the 'Discover Vagrant Boxes' web interface. On the left is a dark sidebar with the HashiCorp logo, a search icon, a user icon, and a 'Vagrant Public Registry' header. Below this is a 'Discover Public Boxes' button and a section titled 'Share your box environment with the community' with a 'Sign up for HCP Vagrant' link. The main area has the title 'Discover Vagrant Boxes' and a search bar. Below the search bar, it says 'Showing 25 results'. There are filters for 'Provider' and 'Architecture', and a 'Sort by: Most downloaded' dropdown. A table lists four boxes: 'ubuntu/trusty64', 'laravel/homestead', 'hashicorp/precise64', and 'centos7'. Each row shows the box name, latest version, download count, latest release date, providers, and architectures.

Box name	Latest Version	Downloads	Latest Release	Providers	Architectures
 ubuntu/trusty64	20191107.0.0	30,793,123	257 weeks ago	virtualbox	unknown
 laravel/homestead	14.0.2	14,596,745	265 days ago	parallels virtualbox libvirt vmware_desktop	amd64 arm64 unknown
 hashicorp/precise64	1.1.0	6,815,698	552 weeks ago	virtualbox vmware_fusion hyperv	unknown
 centos7	2004.01	6,055,453	229 weeks ago	virtualbox vmware_desktop libvirt hyperv vmware_fusion vmware_workstation vmware	unknown

Dans l'écran ci dessus vous allez pouvoir faire des recherches mais également récupérer des informations:

Le nom de la box qui est au format Auteur/Nomdelabox. Par exemple une Ubuntu/Trusty64 est une image officielle du compte Ubuntu et trusty64 la version de l'OS.

La dernière version (souvent une date)

Le nombre de téléchargement (souvent élevé sur les distributions officielles)

La date de la dernière release

Le Providers avec laquelle l'image est faite pour fonctionner. Le providers est l'outil de virtualisation. L'image Ubuntu n'est disponible ici que pour du VirtualBox alors que celles en dessous peuvent fonctionner sur beaucoup d'autres outils.

Ci-dessous je peux voir les informations pour une box officiel Debian 12 (Nom de code Bookworm)

## debian / bookworm64 Vagrant box

How to use this box with [Vagrant](#):



Vagrantfile [New](#)

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/bookworm64"
end
```

**v12.20240905.1** currently released version

This version was created about 1 month ago.

There isn't a description.

2 providers for this version.		
<b>libvirt</b>		
unknown *	Hosted by Vagrant Cloud (331 MB)	
<b>virtualbox</b>		
unknown *	Hosted by Vagrant Cloud (332 MB)	

En haut nous pouvons voir un exemple extrêmement simple de Vagrant File pour démarrer cette box.

On peut constater également qu'elle est disponible pour deux providers, Libvirt et Virtualbox.

## Les commandes de base

Voici une liste des commandes de base.

- `vagrant init` : Initialise un nouveau Vagrantfile dans le répertoire courant.
- `vagrant up` : Démarre et provisionne la machine virtuelle.
- `vagrant halt` : Arrête la machine virtuelle sans la détruire.
- `vagrant reload` : Redémarre la machine virtuelle en appliquant les modifications du Vagrantfile.
- `vagrant provision` : Exécute le script de provisionnement défini dans le Vagrantfile sans redémarrer la VM.
- `vagrant ssh` : Accède à la machine virtuelle via SSH.
- `vagrant suspend` : Met la machine virtuelle en veille.
- `vagrant resume` : Relance une machine virtuelle suspendue.
- `vagrant destroy` : Supprime la machine virtuelle et ses données.
- `vagrant status` : Affiche l'état actuel de la machine virtuelle.
- `vagrant box add` : Ajoute une box Vagrant au système local.
- `vagrant box list` : Liste toutes les boxes disponibles localement.
- `vagrant box remove` : Supprime une box du système local.

## Le Vagrantfile

### Concept du Vagrantfile

Le **Vagrantfile** est un fichier de configuration utilisé par Vagrant pour définir et gérer des environnements de machines virtuelles de manière reproductible. Il décrit tous les paramètres nécessaires pour configurer, provisionner et démarrer une machine virtuelle (VM) ou un ensemble de VMs. Ce fichier permet aux développeurs de partager facilement des environnements de développement identiques, réduisant les problèmes de compatibilité.

### Fonctionnement d'un projet Vagrant

Tout commence avec un dossier pour votre projet. Son nom n'a pas d'importance. Dans ce dossier vous allez y déposer votre Vagrantfile qui aura le nom suivant `Vagrantfile`. On doit respecter la casse et le fichier ne doit pas avoir d'extension, il se nomme juste `Vagrantfile`. Si vous avez besoin de pousser des fichiers à l'intérieur de votre VM vous pouvez les mettre dans ce même dossier, nous verrons plus loin comment le faire.

## Le plus simple possible

```
# Vagrantfile minimaliste pour une VM Debian 12
Vagrant.configure("2") do |config|
  # Configuration de la box Debian 12
  config.vm.box = "debian/bullseye64"
end
```

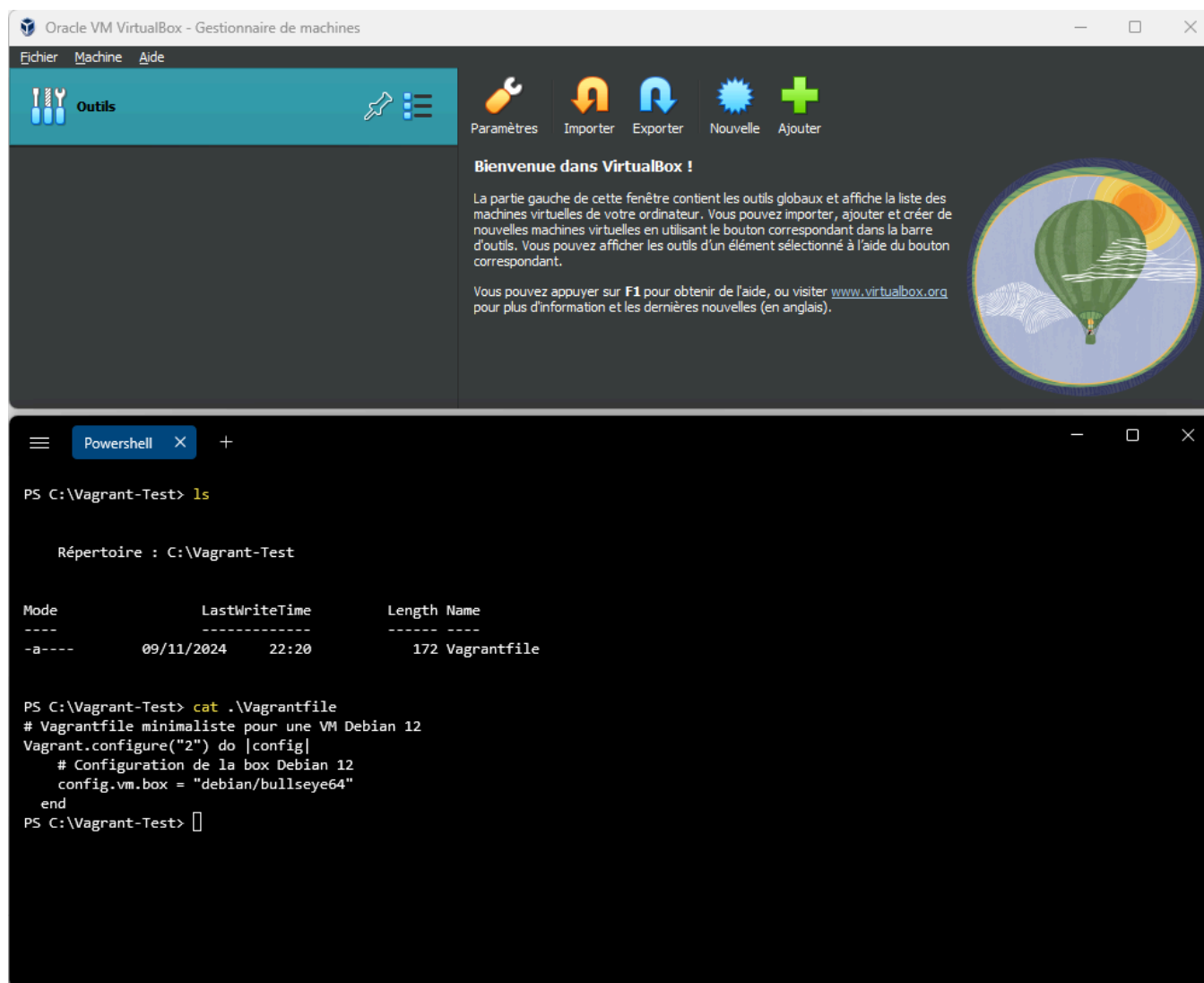
Dans cet exemple de `Vagrantfile` nous demandons de construire une VM de type Debian 12. On peut voir ici le nom de la box que nous souhaitons récupérer.

Vous pouvez connaître plus détail sur cette box à cette adresse [debian/bullseye64](https://cloud-images.debian.org/README-bullseye64.html)

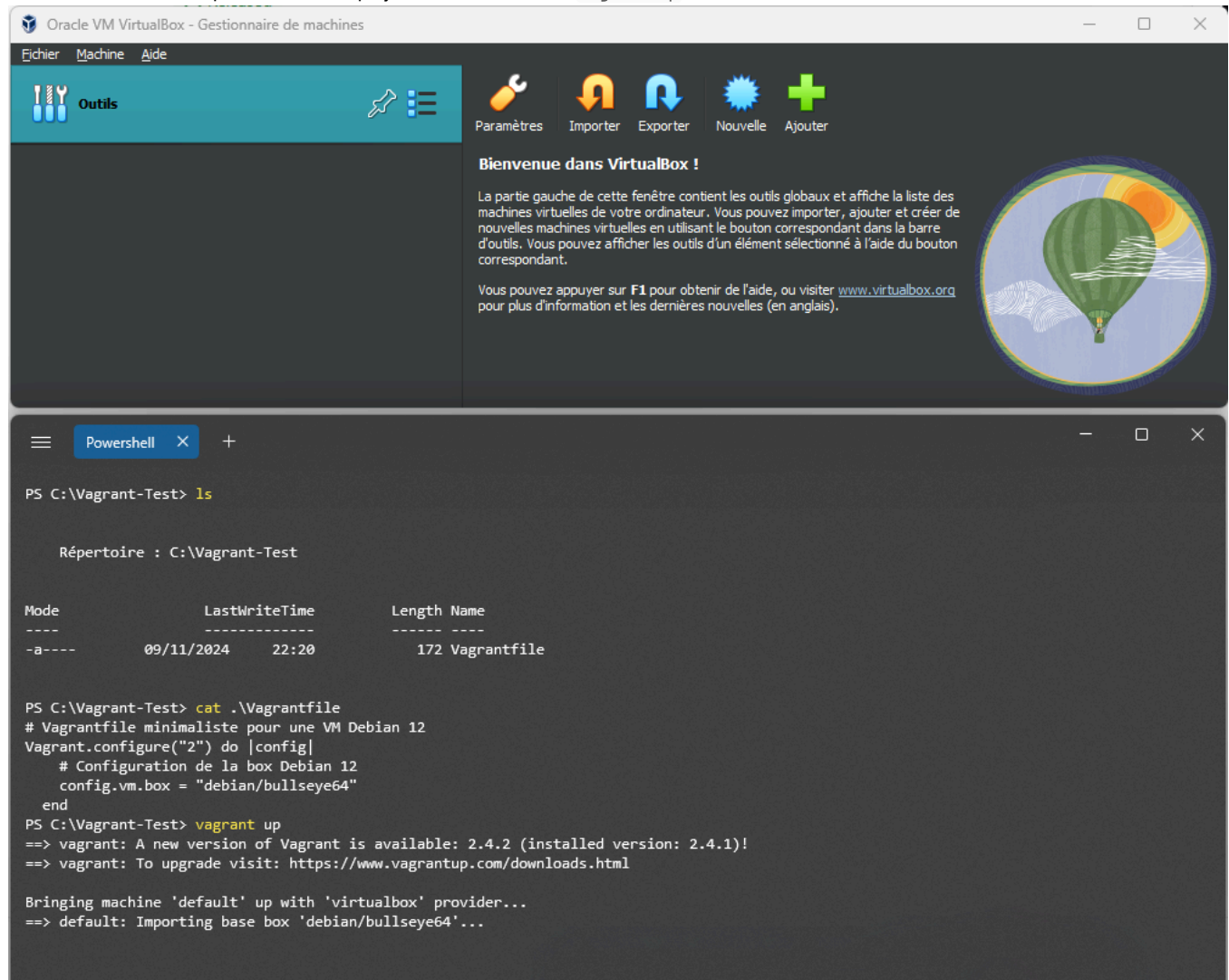
## Démo

### Etape 1: Créer une VM

- VirtualBox
- Un terminal ouvert dans le dossier du projet
- Le Vagrantfile



Nous allons maintenant pouvoir lancer le projet avec la commande `vagrant up`.



cette commande va effectuer plusieurs étapes:

- **Initialisation** : Vagrant lit le Vagrantfile pour obtenir les instructions.
- **Création** : Vagrant télécharge la box si elle n'est pas disponible localement, puis crée la VM.
- **Configuration et provisionnement** : Vagrant applique les configurations (réseau, CPU, etc.) et exécute les scripts de provisionnement.
- **Démarrage** : La VM est prête à l'emploi, configurée exactement comme spécifié dans le Vagrantfile.



Plus de détail sur l'ensemble du retour de la commande:

```
PS C:\Vagrant-Test> vagrant up
==> vagrant: A new version of Vagrant is available: 2.4.2 (installed version: 2.4.1)!
==> vagrant: To upgrade visit: https://www.vagrantup.com/downloads.html

Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'debian/bullseye64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'debian/bullseye64' version '11.20240905.1' is up to date...
==> default: Setting the name of the VM: Vagrant-Test_default_1731187557822_20786
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 6.0.0 r127566
    default: VirtualBox Version: 7.0
==> default: Mounting shared folders...
    default: /vagrant => C:/Vagrant-Test

==> default: Machine 'default' has a post `vagrant up` message. This is a message
==> default: from the creator of the Vagrantfile, and not from Vagrant itself:
==> default:
==> default: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports
PS C:\Vagrant-Test> 
```

En rouge, l'initialisation et l'import de la box.

En bleu, Vérification que la box est dans la dernière version en local. Ici c'est le cas, si ce n'était pas le cas alors Vagrant la téléchargerai.

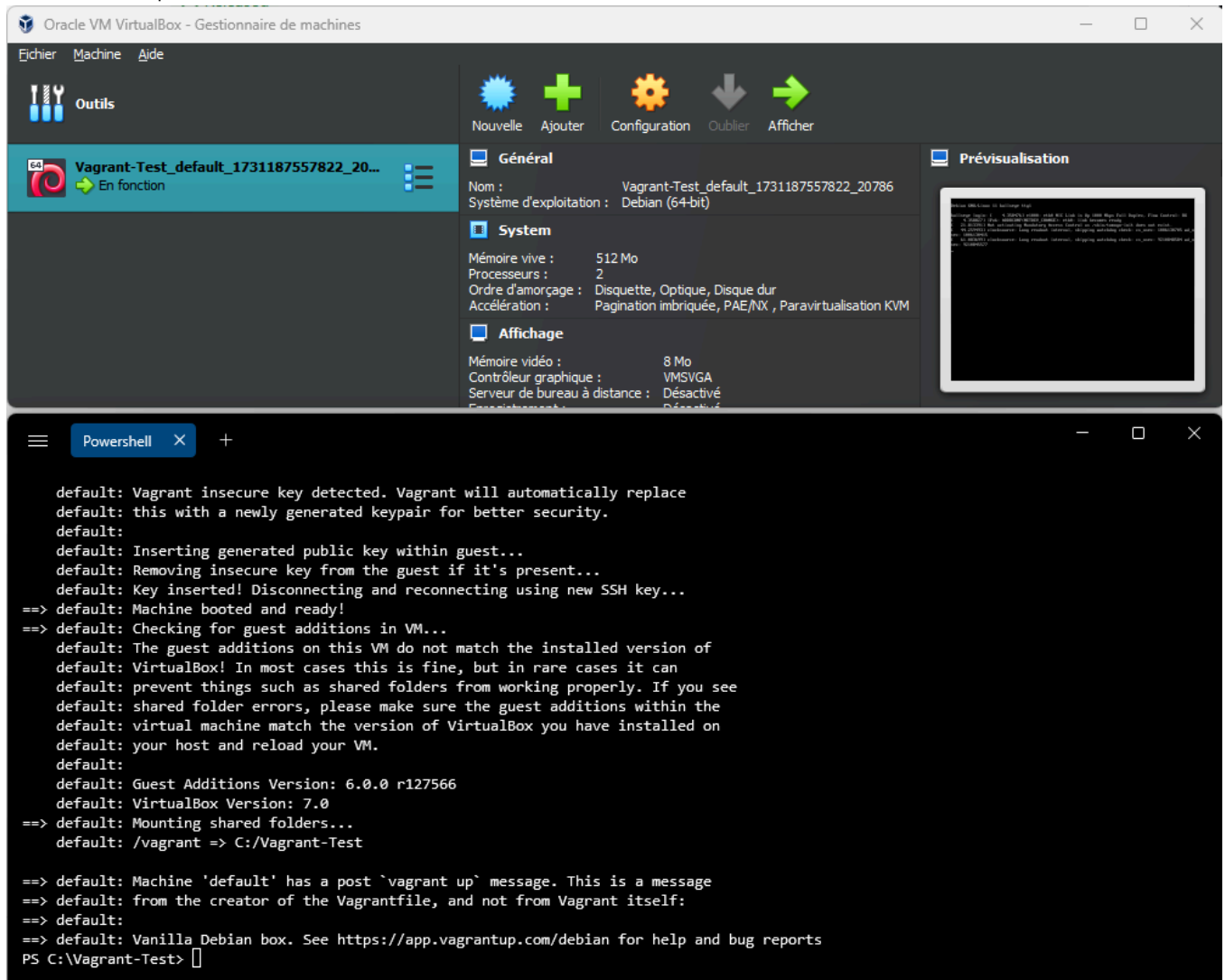
En vert, initialisation des paramètres de la VM dans VirtualBox (nom, réseau, partage de port).

En orange, démarrage de la VM.

En rose, Vagrant se connecte à la VM pour y manipuler des éléments tel que la mise en place de clé SSH, vérification de la présence de Guest Additions et d'éventuelles autres étapes qui ici ne sont pas demandées dans le Vagrantfile.

En violet, affiche un message de la personne qui a créé la box. Ici pour dire que c'est une Debian vanilla.

Une fois fini nous pouvons voir notre VM dans Virtualbox.



## Etape 2: Se connecter à notre VM

De manière classique nous pourrions ouvrir la VM dans VirtualBox et travailler dans la fenêtre. Cela ne serait pas pratique. Il est possible d'installer un OS avec une interface graphique mais dans notre cas c'est une VM avec uniquement une interface en ligne de commande.

Les box fournis par Vagrant fournissent toujours (ou presque) quelques configurations de base. Déjà la présence des Guest additions (drivers et autres outils pour la VM interagisse plus facilement avec VirtualBox) mais également un compte administrateur du nom de "vagrant". Si c'est possible ce compte sera également accessible via SSH et lors de l'installation des clés RSA seront liées à ce compte.

Il est possible d'avoir des informations grâce à la commande `vagrant ssh-config` mais cela nous ressortira plus tard.

```
PS C:\Vagrant-Test> vagrant ssh-config
Host default
  HostName 127.0.0.1
  User vagrant
  Port 2222
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile C:\Vagrant-Test\.vagrant\machines\default\virtualbox\private_key
  IdentitiesOnly yes
  LogLevel FATAL
  PubkeyAcceptedKeyTypes +ssh-rsa
  HostKeyAlgorithms +ssh-rsa

PS C:\Vagrant-Test>
```

Il est possible de se connecter directement en SSH dans la machine Vagrant depuis le terminal avec la commande `vagrant ssh`.

```
PS C:\Vagrant-Test> vagrant ssh
Linux bullseye 5.10.0-32-amd64 #1 SMP Debian 5.10.223-1 (2024-08-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
vagrant@bullseye:~$
```

Vous pouvez voir que la mire à changer. Je suis maintenant dans la VM. A partir de la, je peux travailler dans ma VM via ce terminal. Je montre plus loin comment travailler avec un éditeur de texte/IDE tel que VScode.

On peut sortir de la VM avec la commande `exit` comme si on quittait n'importe quelle connexion SSH.

### Etape 3: Stopper la VM

Pour cela, une commande simple `vagrant halt`.

```
vagrant@bullseye:~$ exit
logout
PS C:\Vagrant-Test> vagrant halt
==> default: Attempting graceful shutdown of VM...
PS C:\Vagrant-Test>
```

### Etape 4: Supprimer la VM

Pour cela une autre commande, `vagrant destroy`, qui sera suivi d'une validation utilisateur.

```
PS C:\Vagrant-Test> vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Destroying VM and associated drives...
PS C:\Vagrant-Test>
```

L'ensemble des fichiers de la VM sera supprimé.

#### Note

Depuis la première création, un dossier `.vagrant` a été créé dans le dossier du projet. Vous pouvez éventuellement le supprimer. Notamment si vous souhaitez réinitialiser le projet de zéro si quelque chose semble bloquer.

## Plus complet

Dans l'exemple suivant nous avons rajouté quelques paramètres.

Plusieurs paramètres, plusieurs sont liés à VirtualBox.

La partie provisionnement sert à passer un script qui sera exécuter une seule fois, lors du premier démarrage de la VM.

```
# Vagrantfile pour une VM Debian 12 sous VirtualBox
Vagrant.configure("2") do |config|
  # Configuration de la box Debian 12
  config.vm.box = "debian/bullseye64"

  # Configuration du réseau en mode bridge au lieu du nat par défaut
  config.vm.network "public_network", bridge: true

  # Configuration des ressources (RAM et CPU) avec spécification du provider
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 2
  end

  # Provisionnement avec Apache et copie du fichier index.html
  config.vm.provision "shell", inline: <<-SHELL
  # Met à jour les paquets et installe Apache
  apt-get update
  apt-get install -y apache2

  # Copie du fichier index.html dans le répertoire web d'Apache
  echo "<html><body><h1>Bienvenue sur mon serveur Apache</h1></body></html>" > /var/www/html/index.html
SHELL
end
```

Si je démarre cette VM je pourrais voir les commandes du provisionnement passer ainsi que leur réponse ce qui est très utile pour déboguer un problème avec le script. Attention cela dit il n'est pas possible d'interagir avec il faut donc passer un script qui n'aura strictement aucune interaction.

## Plusieurs VM

Dans cet exemple je vais construire 2 VMs.

Une VM Web nommé `vagrant-srvweb` dans VirtualBox mais nommé `web` dans Vagrant. Son hostname sur le réseau sera `Vagrant-SRVWEB`.  
Une VM DB nommé `vagrant-db` dans VirtualBox mais nommé `db` dans Vagrant. Son hostname sur le réseau sera `Vagrant-SRVDB`.

Les deux VM sont dans un réseau privé pour pouvoir communiquer entre elles.

Dans la VM web j'installe apache 2 et PHP avec une paquet supplémentaire.

Je provisionne un fichier `index.php` que vous retrouvez juste en dessous et DOIT se trouver à la racine du dossier du projet. Il sera copié dans la VM puis placé dans le dossier d'apache. La VM sera donc fonctionnelle. Cette page PHP appel au chargement une base de donnée situé dans la deuxième VM.

La deuxième VM va plutôt lancer un script. Ce script se situe juste en dessous. Ce script installera MariaDB, va configurer ce dernier, va créer un utilisateur et permettre l'accès au serveur web.

Des ports sont également partagés entre les VM et l'hôtes, le site web sera donc accessible depuis votre machine avec l'adresse `http://localhost:8080`.

Pour démarer les deux VM en meme temps, `vagrant up`. Pour n'en démarrer qu'une il suffit de rajouter son nom dans le fichier, pour la web par exemple `vagrant up web`. Idem pour stopper ou détruire une VM.

```
Vagrant.configure("2") do |config|

  config.vm.define "web" do |web|
    web.vm.box = "debian/bookworm64"
    web.vm.hostname = "Vagrant-SRVWEB"
    web.vm.provider :virtualbox do |v|
      v.customize ["modifyvm", :id, "--memory", 512]
      v.customize ["modifyvm", :id, "--cpus", 1]
      v.customize ["modifyvm", :id, "--name", "vagrant-srvweb"]
    end
    web.vm.network :forwarded_port, host: 8080, guest: 80
    web.vm.network "private_network", ip: "192.168.56.2"
    web.vm.provision "file", source: "../index.php", destination: "~/index.php"
    web.vm.provision "shell", inline: <<-SHELL
    sudo apt update
    sudo apt install -y apache2 php php-mysql
    sudo systemctl restart apache2
    sudo rm -rf /var/www/html/index.html
    sudo mv /home/vagrant/index.php /var/www/html/index.php
  SHELL
end

  config.vm.define "db" do |db|
    db.vm.box = "debian/bookworm64"
    db.vm.hostname = "Vagrant-SRVDB"
    db.vm.provider :virtualbox do |w|
      w.customize ["modifyvm", :id, "--memory", 512]
      w.customize ["modifyvm", :id, "--cpus", 1]
      w.customize ["modifyvm", :id, "--name", "vagrant-db"]
    end
    db.vm.network :forwarded_port, host: 3306, guest: 3306
    db.vm.network "private_network", ip: "192.168.56.3"
    db.vm.provision "shell", path: "mysql.sh"
  end

end
```

```
// Fichier index.php
<?php
// Paramètres de connexion
$servername = "192.168.56.3"; // Remplacez par l'adresse de votre serveur
$username = "nouvel_utilisateur"; // Remplacez par votre nom d'utilisateur de la base de données
$password = "mot_de_passe"; // Remplacez par votre mot de passe de la base de données
$dbname = "ma_base_de_donnees"; // Remplacez par le nom de votre base de données

// Création de la connexion
$conn = new mysqli($servername, $username, $password, $dbname);
// Vérification de la connexion
if ($conn->connect_error) {
    echo "Connexion échouée!!!!";
} else {
```

```

    echo "Connexion réussie";
}
// Fermeture de la connexion
$conn->close();
?>

```

```

# Fichier mysql.sh
sudo apt update
sudo apt install -y mariadb-server
sudo mysql -u root <<EOF

# Créer un nouvel utilisateur et attribuer des privilèges
CREATE USER 'nouvel_utilisateur'@'%' IDENTIFIED BY 'mot_de_passe'; -- Remplacez par vos propres nom d'utilisateur et mot
de passe
GRANT ALL PRIVILEGES ON *.* TO 'nouvel_utilisateur'@'%' WITH GRANT OPTION;

# Créer une nouvelle base de données
CREATE DATABASE ma_base_de_donnees; -- Remplacez par le nom de votre base de données

# Appliquer les privilèges
FLUSH PRIVILEGES;

EOF

# Modifier le fichier de configuration de MariaDB pour permettre les connexions externes
sudo sed -i 's/^bind-address\s*=\s*/bind-address = 0.0.0.0/' /etc/mysql/mariadb.conf.d/50-server.cnf

# Redémarrer le service MariaDB pour appliquer les changements
sudo systemctl restart mariadb

```

## Exemple pour une VM Docker

Quand je travail sous Windows, fonctionner avec des Dockers est assez chiant. J'ai pris l'habitude de travailler dans une VM. Pour me simplifier la vie j'ai donc un Vagrantfile pour monter une VM toute prête pour faire du docker. En voici le fichier:

```

# Vagrantfile pour une VM Debian 12 avec Docker depuis les dépôts Docker
Vagrant.configure("2") do |config|
  # Définition de la box Debian 12
  config.vm.box = "debian/bullseye64"

  # Configuration des ressources de la VM
  config.vm.provider "virtualbox" do |vb|
    vb.name = "Debian-Docker"          # Nom de la VM dans VirtualBox
    vb.memory = "2048"                 # Mémoire RAM allouée
    vb.cpus = 1                        # Nombre de CPU alloués
  end

  # Provisionnement : Installation de Docker depuis les dépôts officiels
  config.vm.provision "shell", inline: <<-SHELL
  # Mise à jour des paquets
  apt-get update

  # Installation des dépendances pour l'ajout des dépôts Docker
  apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release

  # Ajout de la clé GPG officielle de Docker
  curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-
keyring.gpg

  # Ajout du dépôt Docker officiel
  echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/debian $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list >
/dev/null

  # Mise à jour des paquets pour inclure les nouveaux paquets Docker
  apt-get update

  # Installation de Docker
  apt-get install -y docker-ce docker-ce-cli containerd.io
SHELL
end

```

# Comment je travail avec Vagrant

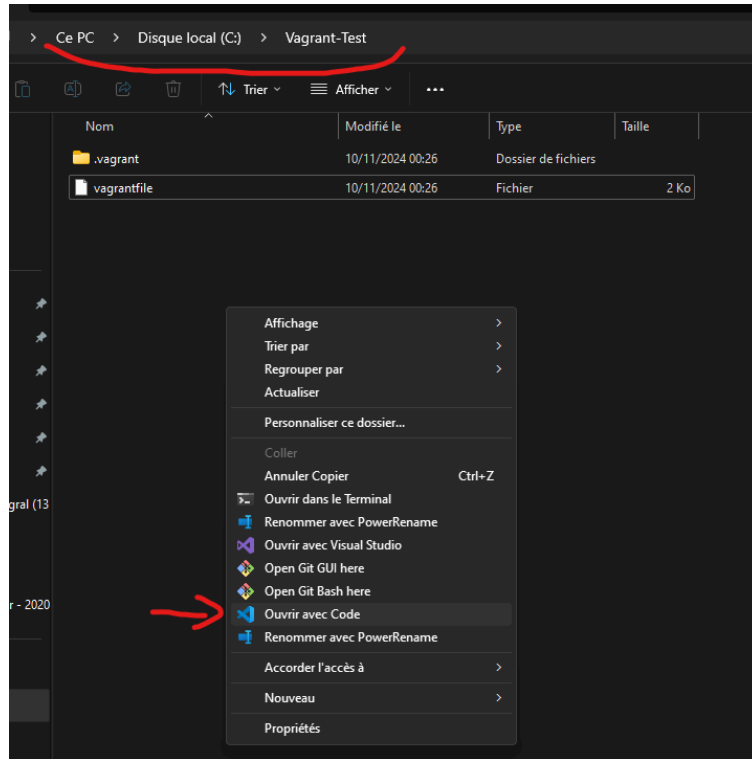
Pour expliquer tout ça je vais partir sur la VM pour docker présenté au dessus.

## Note

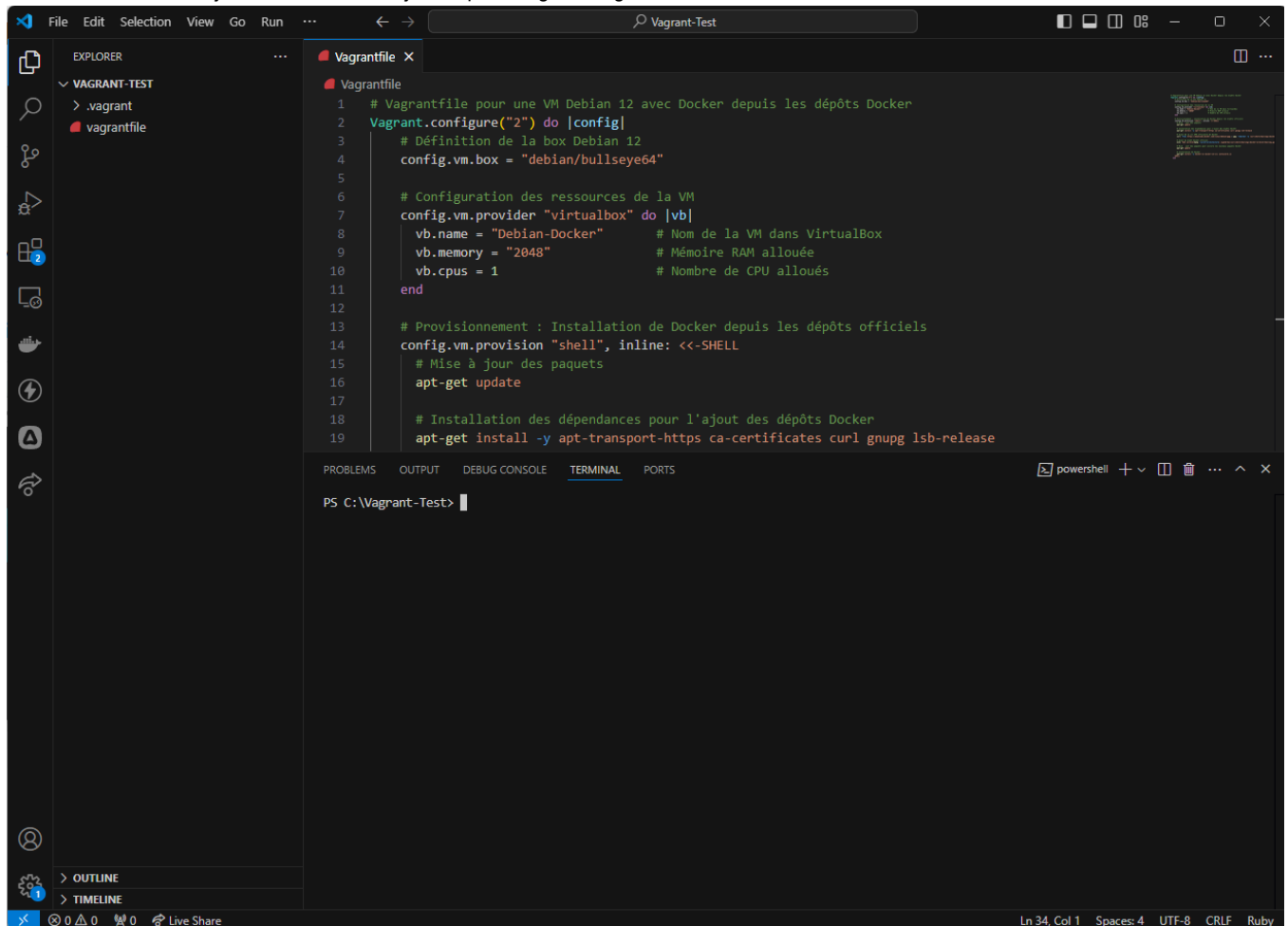
Il est possible d'utiliser plusieurs type d'IDE. Il est possible de faire la meme chose avec les outils de JetBrains ou Sublim Text par exemple. Pour la suite, moi j'utilise VScode.

## Préparer VScode pour qu'il se connecte dans ma VM

Généralement je commence par ouvrir mon dossier de travail avec VScode



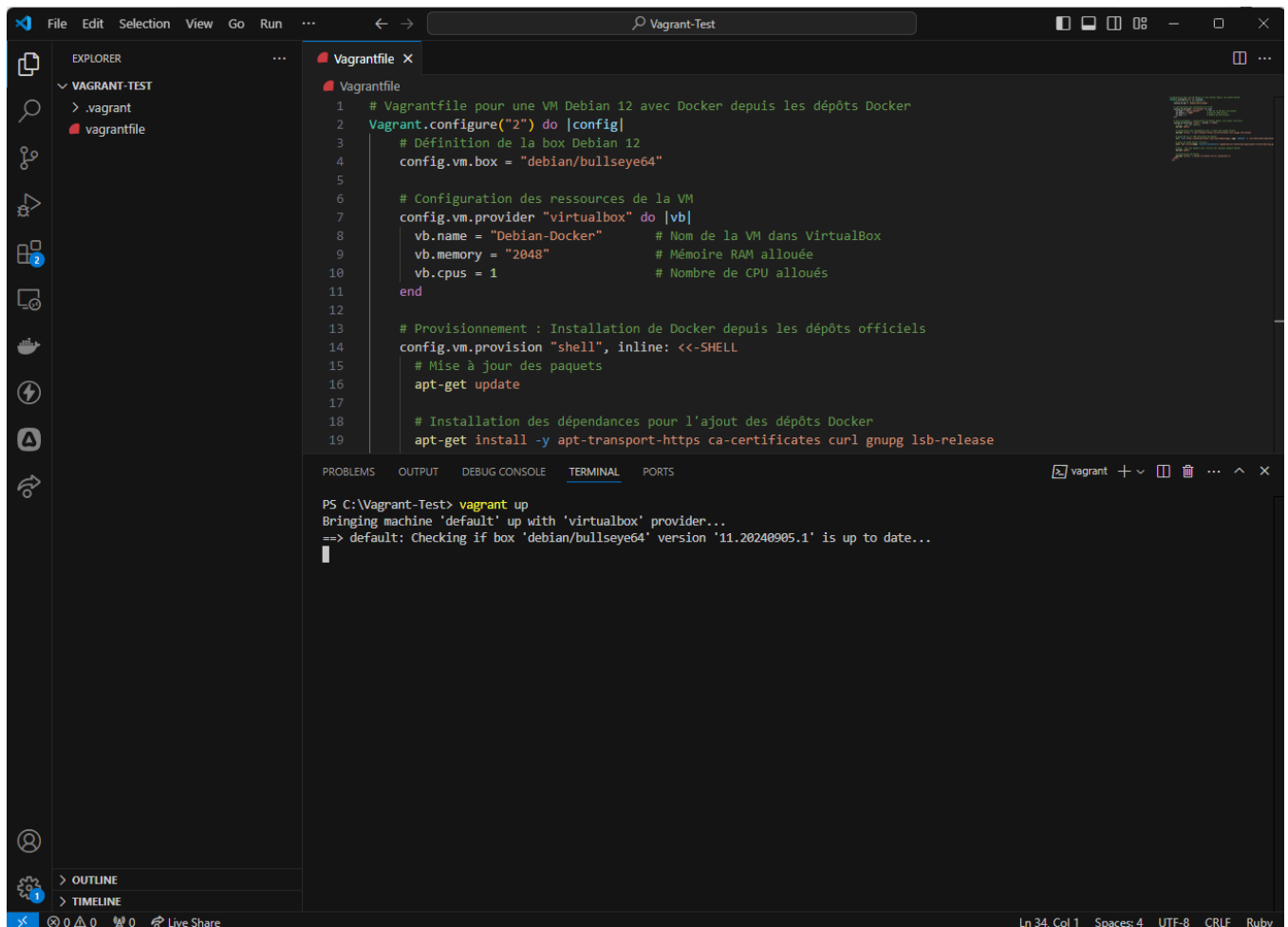
Une fois dans VScode, j'ouvre un terminal et je vais pouvoir gérer Vagrant dedans et donc modifier le fichier au besoin.



The screenshot shows the VS Code interface with the Vagrantfile open in the editor. The file explorer on the left shows the project structure with 'VAGRANT-TEST' containing '.vagrant' and 'vagrantfile'. The terminal at the bottom shows the PowerShell prompt 'PS C:\Vagrant-Test>'.

```
1 # Vagrantfile pour une VM Debian 12 avec Docker depuis les dépôts Docker
2 Vagrant.configure("2") do |config|
3   # Définition de la box Debian 12
4   config.vm.box = "debian/bullseye64"
5
6   # Configuration des ressources de la VM
7   config.vm.provider "virtualbox" do |vb|
8     vb.name = "Debian-Docker"      # Nom de la VM dans VirtualBox
9     vb.memory = "2848"            # Mémoire RAM allouée
10    vb.cpus = 1                    # Nombre de CPU alloués
11  end
12
13  # Provisionnement : Installation de Docker depuis les dépôts officiels
14  config.vm.provision "shell", inline: <<-SHELL
15    # Mise à jour des paquets
16    apt-get update
17
18    # Installation des dépendances pour l'ajout des dépôts Docker
19    apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release
```

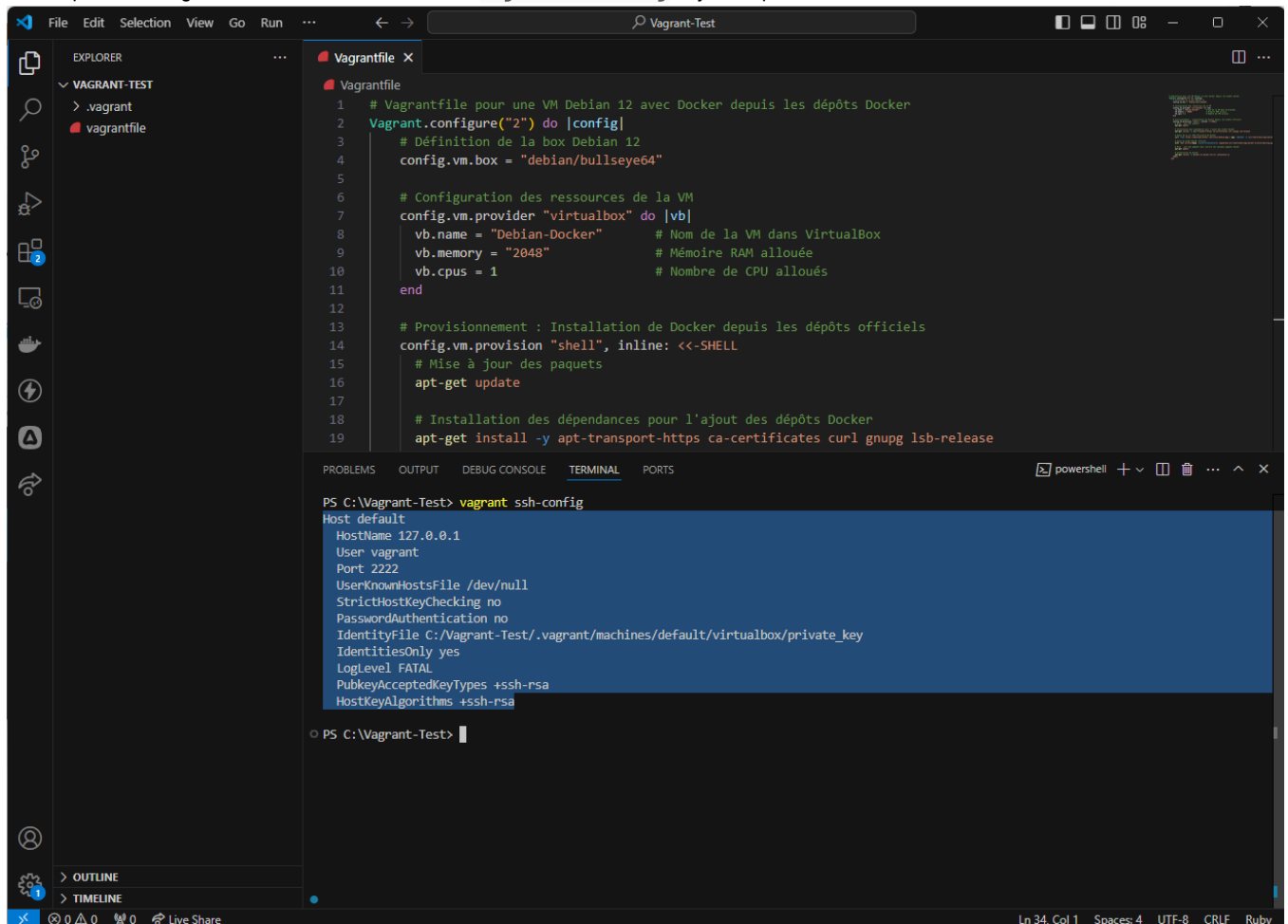
Je démarre ma VM



The screenshot shows the VS Code interface with the Vagrantfile open in the editor. The terminal at the bottom shows the output of the 'vagrant up' command, indicating that the machine 'default' is being brought up with the 'virtualbox' provider and checking for updates.

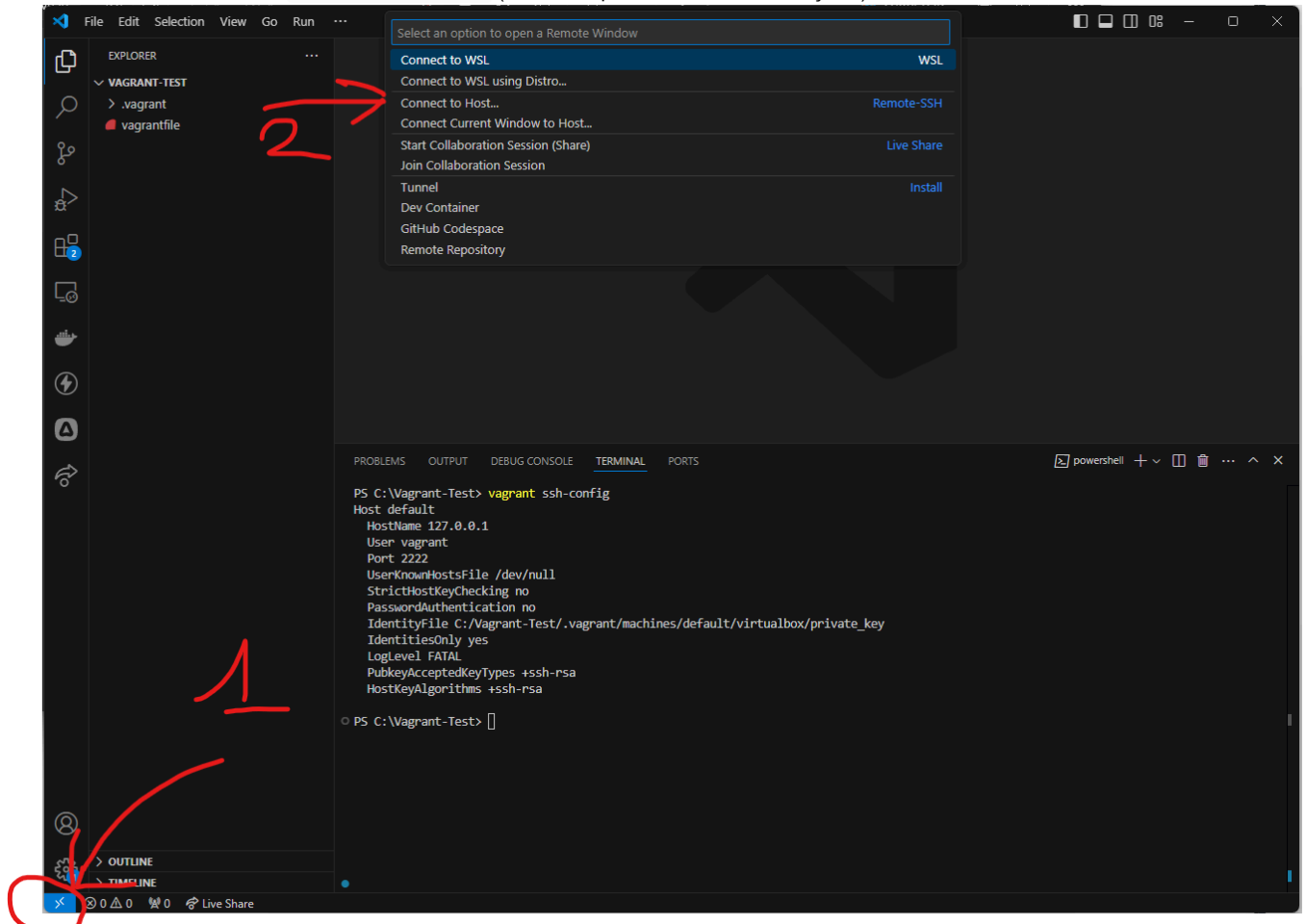
```
PS C:\Vagrant-Test> vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
=> default: Checking if box 'debian/bullseye64' version '11.20240905.1' is up to date...
```

Je récupère la configuration SSH avec la commande `vagrant ssh-config` et je la copie.



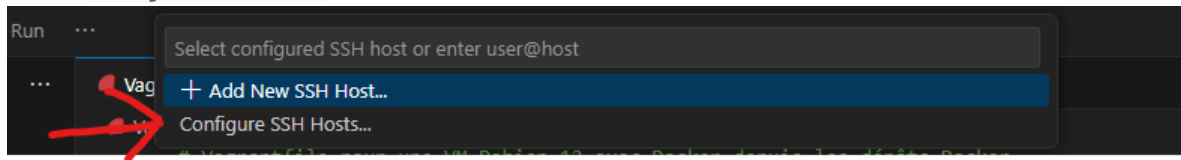
Ensuite je click en bas a droite pour ouvrir le menu de Remote Connexion.

Ensuite dans le menu du haut, Connecto to Host... (vous avez peut être la version française).

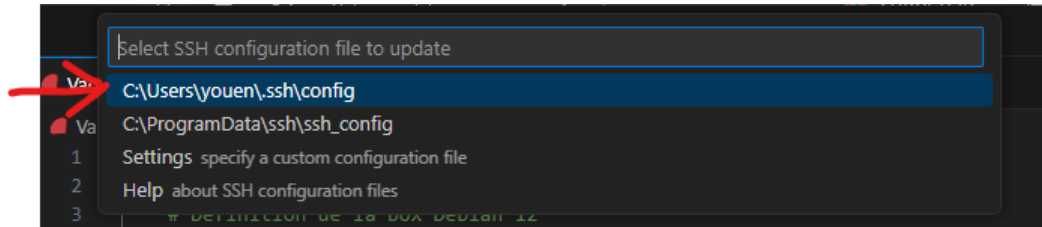




Ensuite, Configure SSH Host...

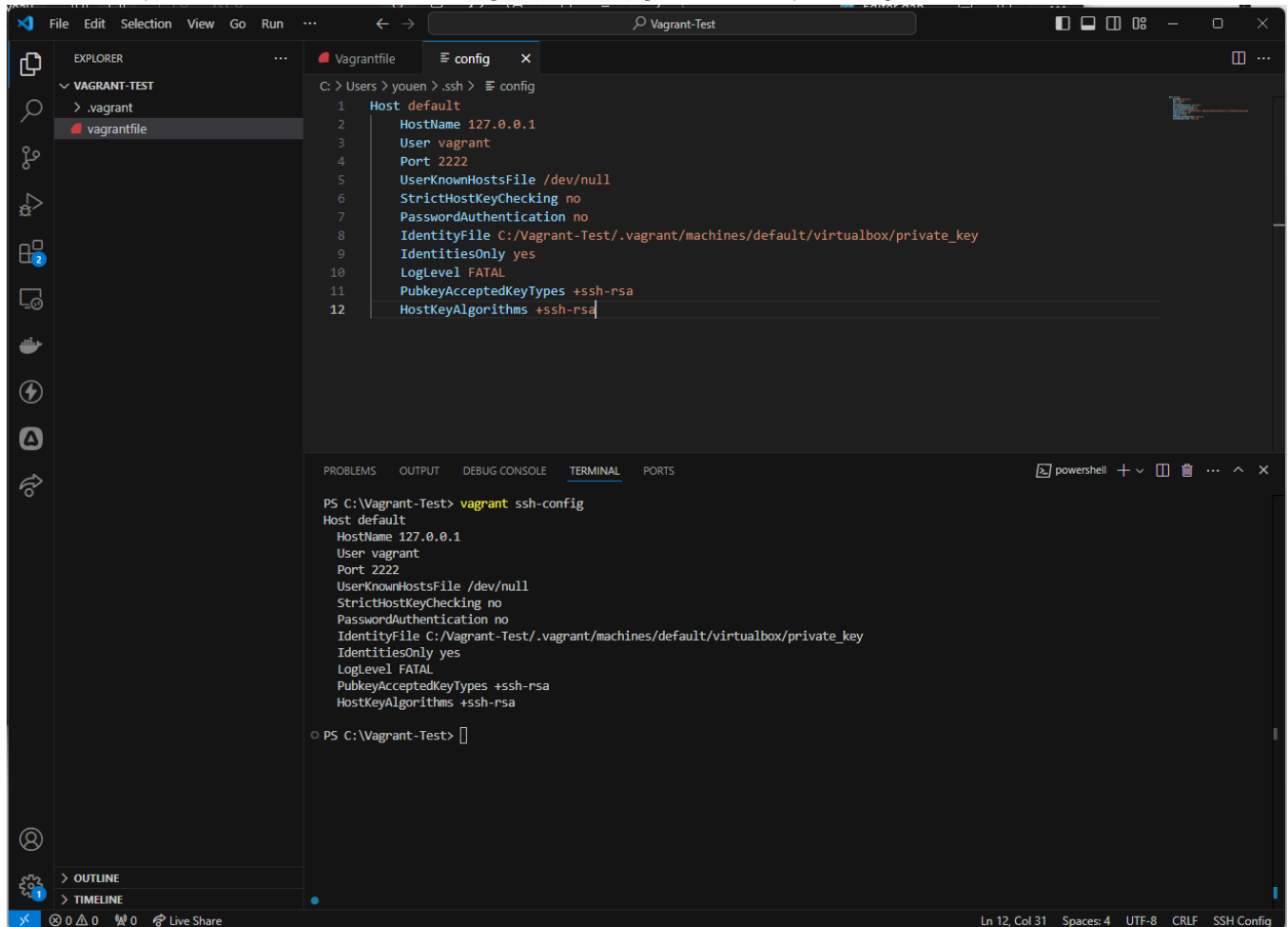


Ensuite le fichier de config SSH



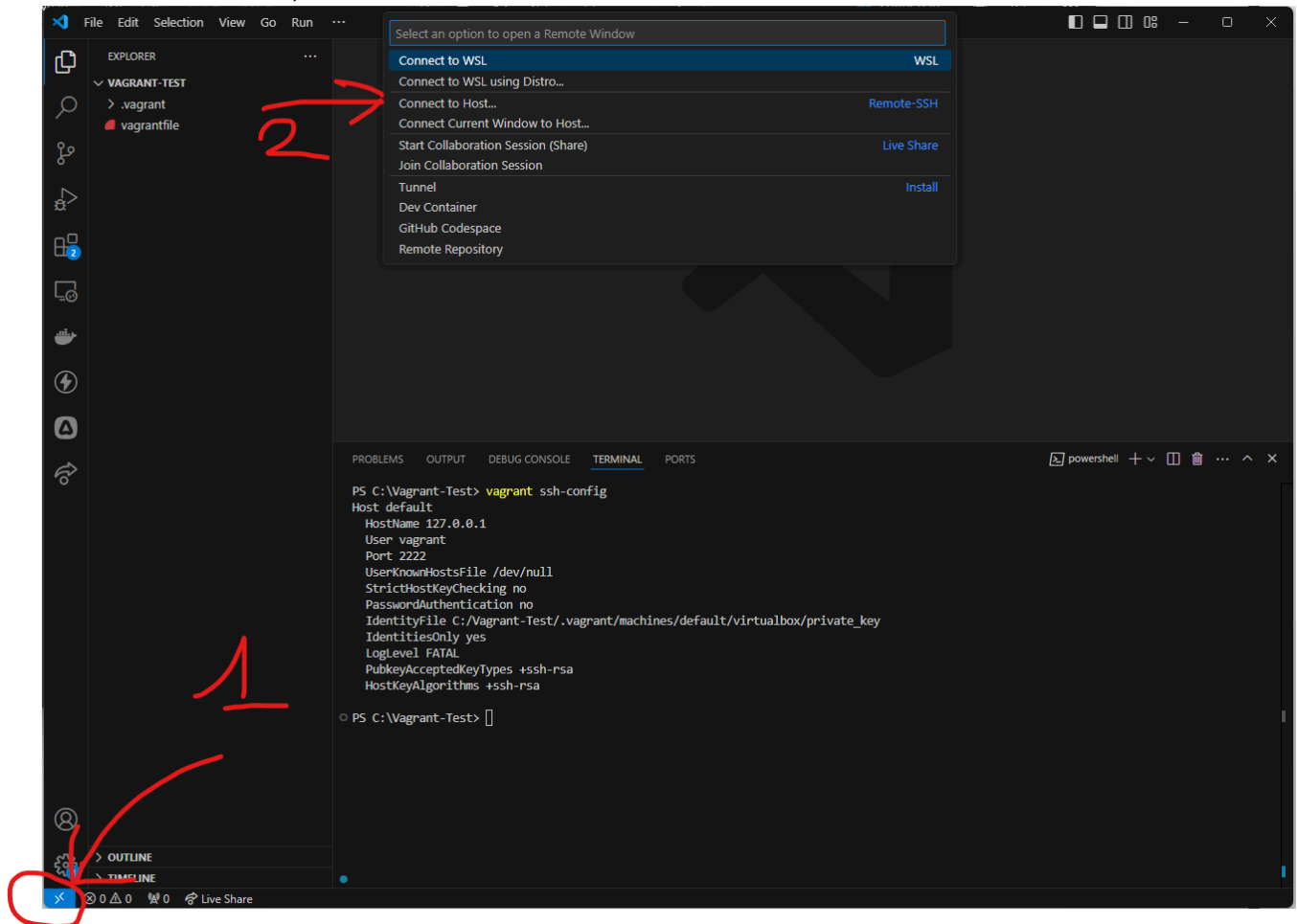
Cela va ouvrir le fichier en question.

Vous devrez copier coller la sortie de la commande `vagrant ssh-config` dans le fichier puis l'enregistrer.

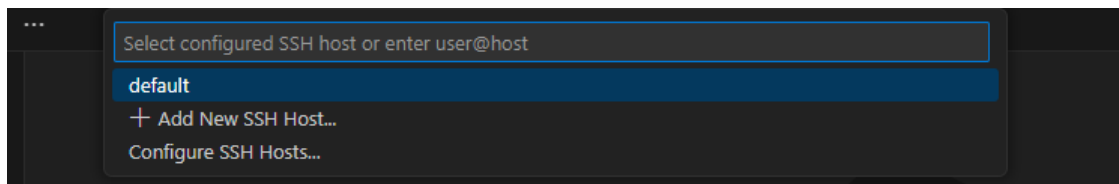


**Connecter VScode à ma VM**

Je vais retourner dans les menu pour me connecter à un hôte.



Cette fois-ci je vois une nouvelle ligne `default`. C'est ma VM! Je clique dessus

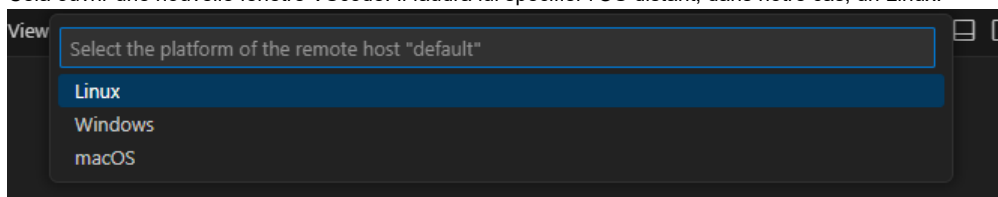


#### Note

Il est possible de changer le nom. C'est dans la config qui a été copié un peu avant La première ligne `Host default`, il suffit de changer `default` par le nom que vous voulez.

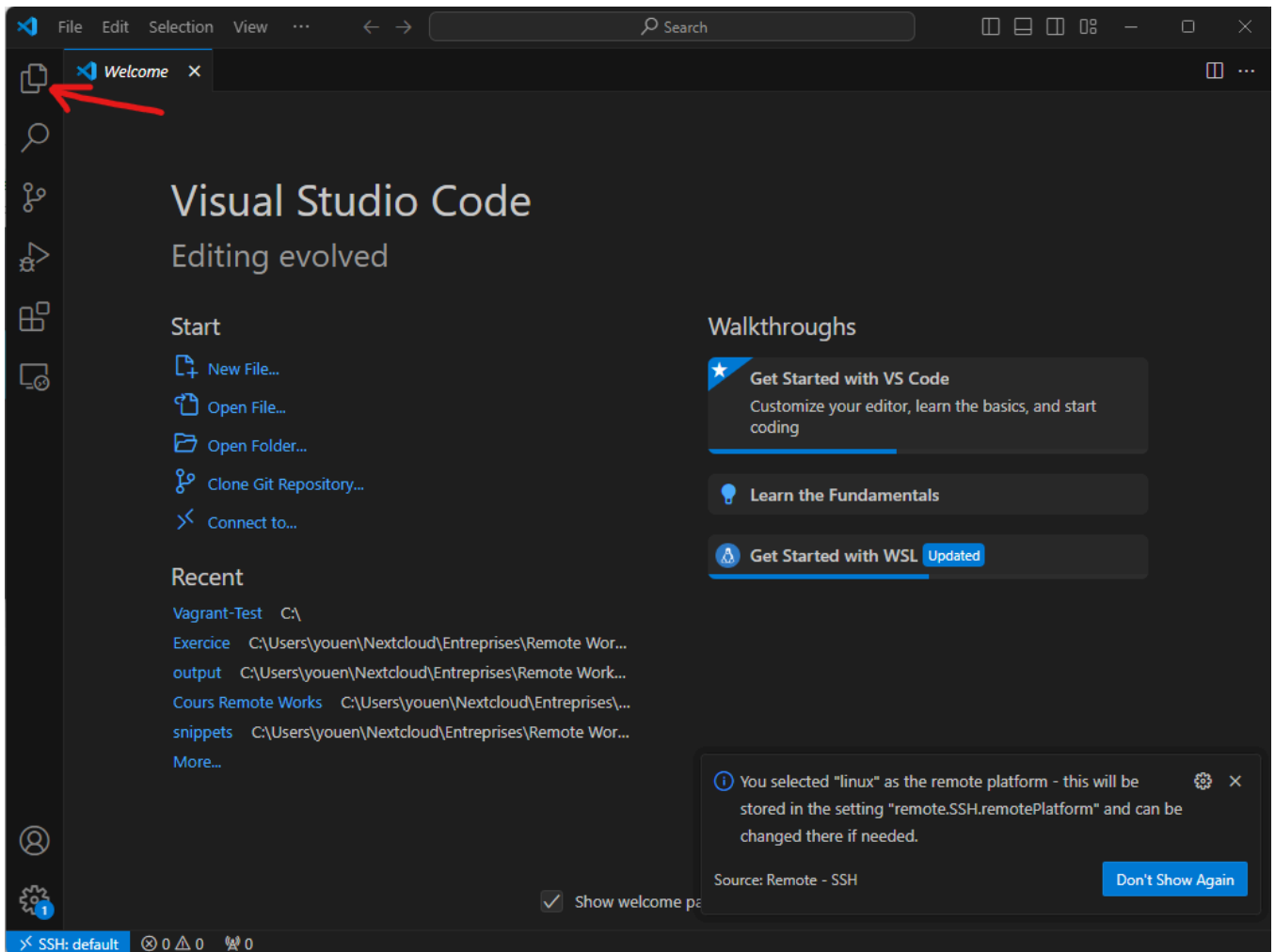
Il est également possible de mettre plusieurs configuration les unes à la suite des autres!

Cela ouvre une nouvelle fenêtre VSCode. Il faudra lui spécifier l'OS distant, dans notre cas, un Linux.

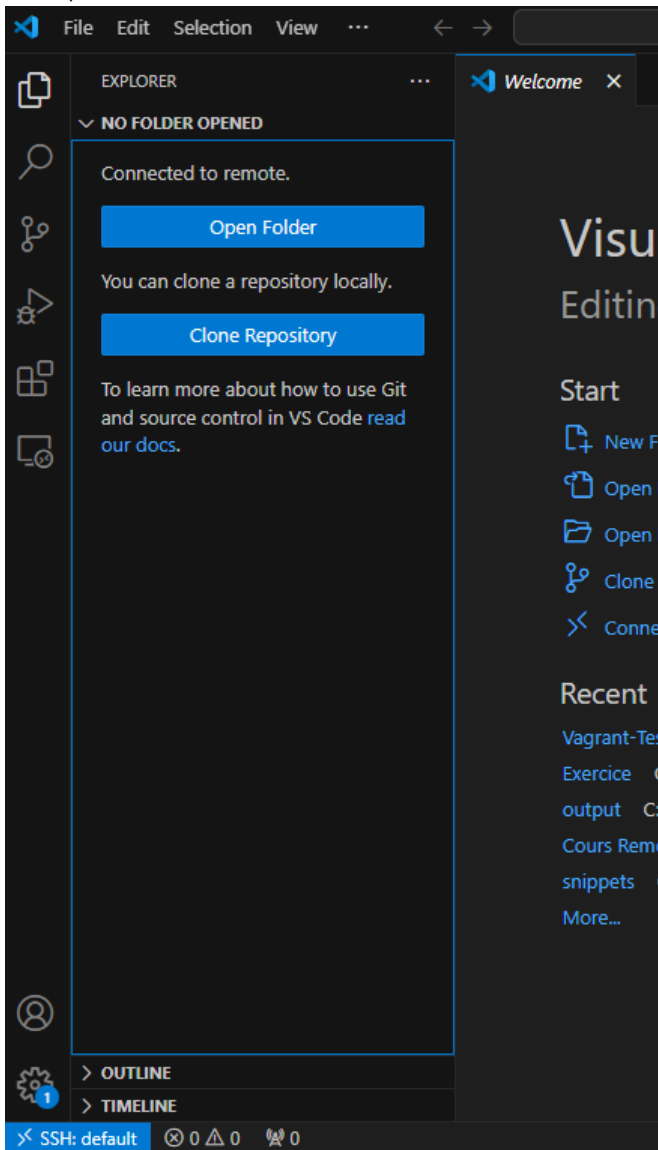


A ce moment là, VScode va installer ce qui est nécessaire pour interagir avec votre VM.

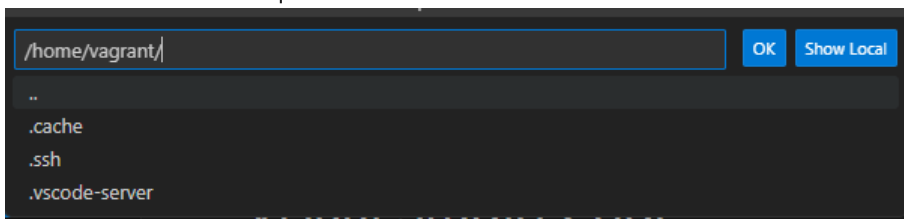
Il nous faut maintenant ouvrir un dossier.



Puis Open Folder

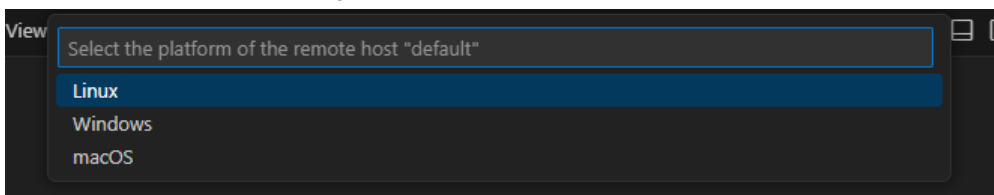


Ensuite on choisit le dossier que l'on veut ouvrir.

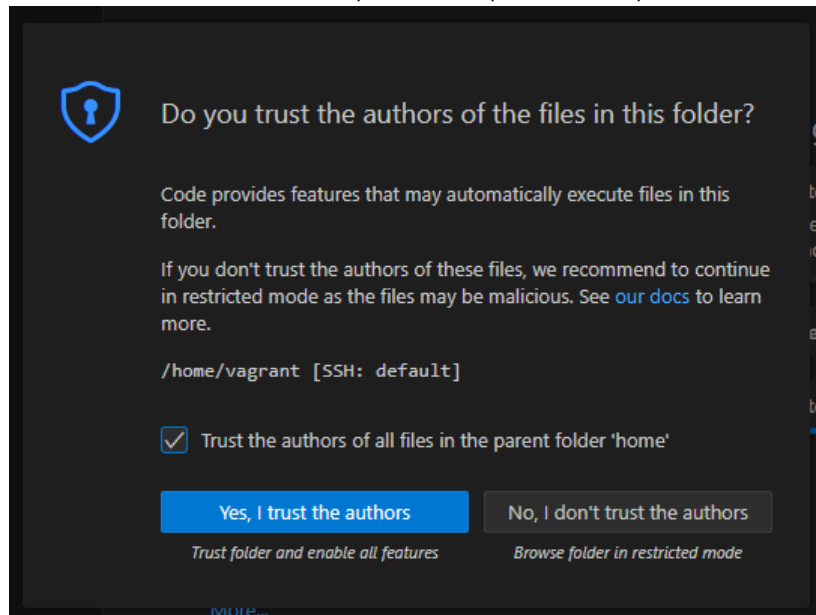


Ici je choisis celui par défaut à savoir le dossier de mon utilisateur

Possiblement on doit lui redonner l'OS

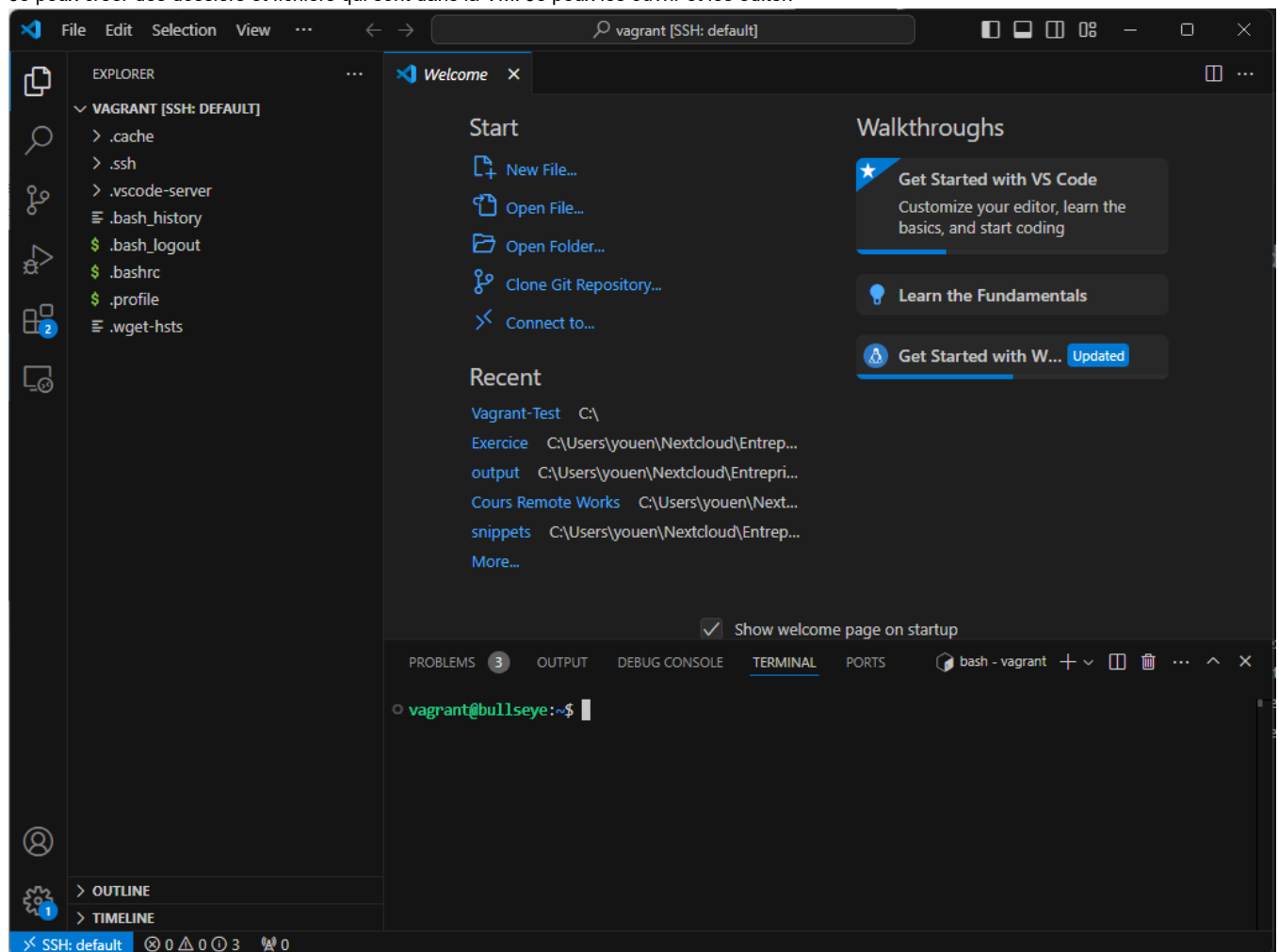


Je vais avoir confiance au dossier en question ainsi que ses dossier parents



Me voici dans ma VM.

Je peux créer des dossiers et fichiers qui sont dans la VM. Je peux les ouvrir et les éditer.

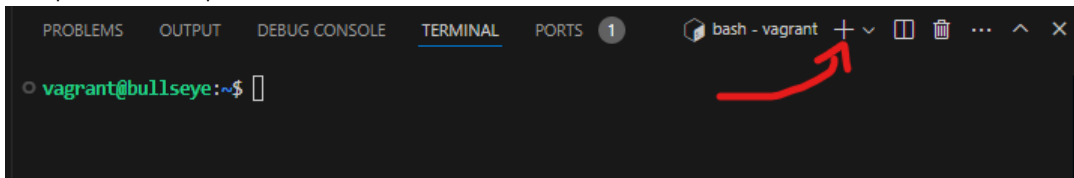


## Tips supplémentaires

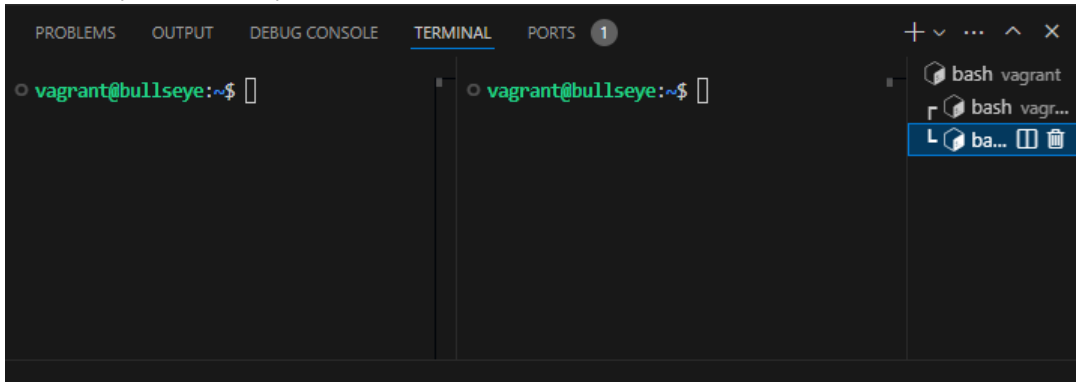
### Le terminal :

Sur le screen j'ai aussi ouvert un terminal, le terminal se trouve dans ma VM.

Il est possible d'avoir plusieurs terminal d'ouvert.



Il est même possible de les splitters.

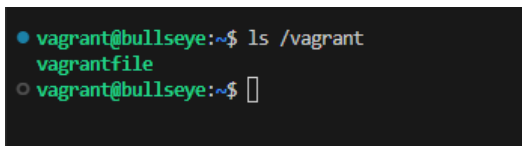


### Sudo :

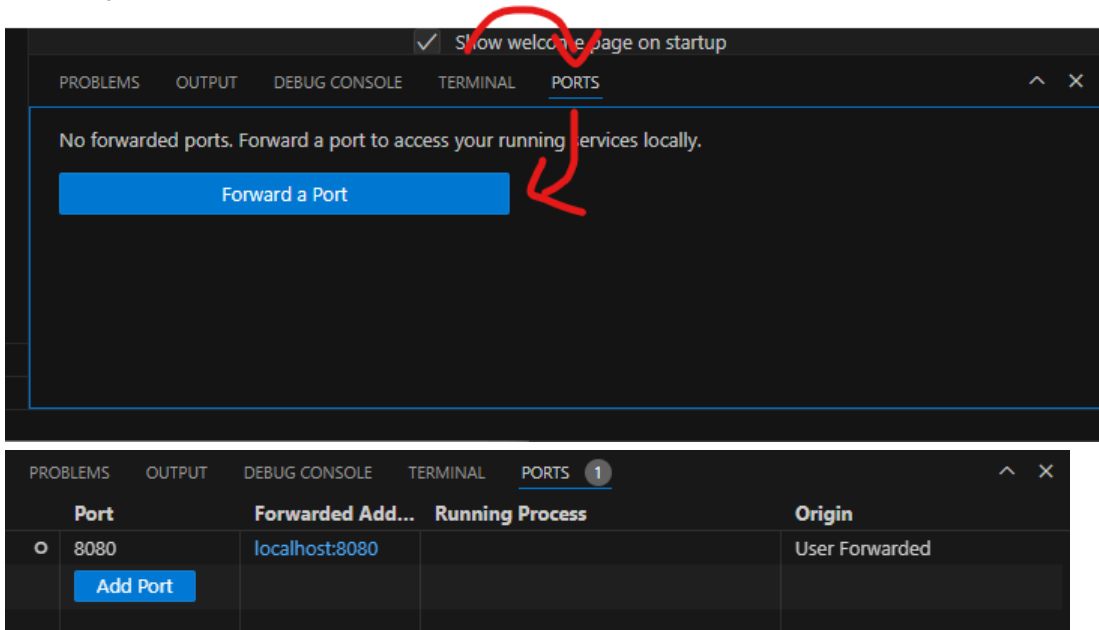
Pour rappel mon utilisateur Vagrant par défaut est Sudoer.

### Dossier Partagé avec l'hôte :

Il est possible d'accéder au dossier projet qui se trouve sur mon pc depuis la VM, ce qui est pratique pour passer facilement un fichier de mon hôte à ma VM et inversement. Ce dossier se trouve dans la VM au chemin `/vagrant`.



### Ouvrir un port de l'invité sur l'hôte :

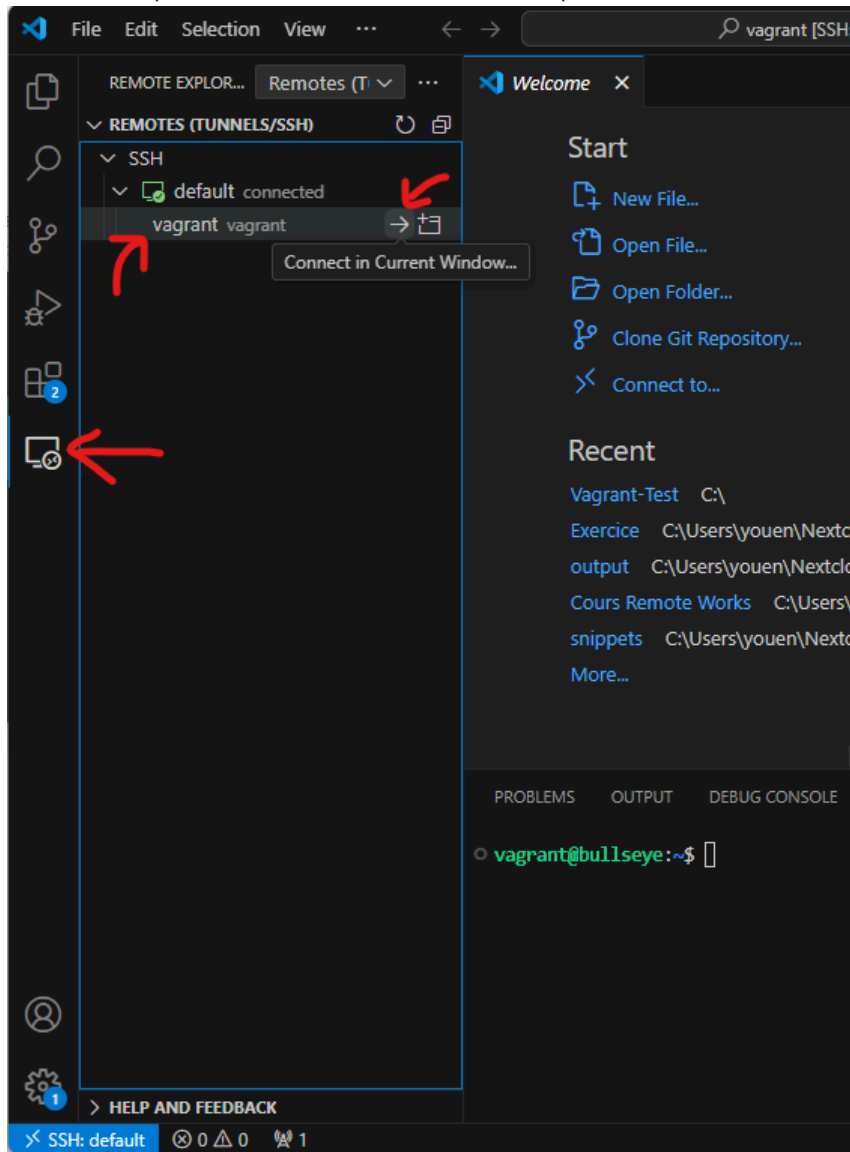


A ce moment la un port est partagé entre la VM et l'hôte et il est possible de communiquer avec, par exemple, votre navigateur. VScode est même parfois capable de comprendre que vous avez lancé un programme qui ouvre un port sur la VM et le forward automatiquement.

### Se reconnecter rapidement à sa VM :

Il y a un menu "Remote Explorer" qui liste toutes les connexions SSH.

Par machine et par dossier ouvert. Il suffit d'ouvrir ce menu pour se connecter en 3 clicks.



## Connaitre son provider

Il y a beaucoup de choses possible à faire, notamment au niveau du réseau (NAT, bridge, privé etc.). Si vous souhaitez approfondir ça, il sera important de mieux comprendre le fonctionnement de votre provider. Dans les exemples le provider est toujours VirtualBox mais il y'en a d'autres. Toutes les box de Vagrant ne sont pas disponibles pour tous les providers. Une liste des providers se trouve sur le site de Vagrant ainsi que sur les pages des box.

Par exemple l'image que j'ai utilisé dans ces exemples est disponible sur VirtualBox et libvirt.

### Latest Version

**11.20240905.1** ✓ Released

Created Sep 5, 2024, 6:55:35 AM

#### libvirt

unknown Hosted on HCP Vagrant Registry (304,93 MB)

#### virtualbox

unknown Hosted on HCP Vagrant Registry (305,36 MB)