Mastering Vagrant

Notes sur la formation vagrant :

1. **La commande vagrant global-status** 👍
   Elle affiche le statut de toutes les machines Vagrant sur votre système, quel que soit le répertoire où vous vous trouvez.
   Elle fournit plusieurs informations sur ces machines là, telles que :

- ID : Identifiant unique de la machine
- Name : Nom de la machine
- Provider : Hyperviseur utilisé (VirtualBox, VMware, etc.)
- State : État actuel (running, poweroff, aborted, etc.)
- Directory : Chemin où se trouve le Vagrantfile

>>> exemples :

id name provider state directory

---

a1b2c3d default virtualbox running /path/to/project1

>>> vagrant global-status -h pour afficher les sous-options de cette commande.

Une sous-commande très intéressante :

La commande vagrant global-status --prune sert à **afficher la liste de toutes les machines virtuelles gérées par Vagrant**, mais **en supprimant les anciennes informations (cachées)** qui ne sont plus valides.

- La commande vagrant global-status toute seule affiche l'état (en cours d'exécution, arrêtée, etc.) de toutes les machines Vagrant connues sur ton ordinateur.

- Cependant, Vagrant garde ces informations en **cache**. Du coup, il peut arriver qu'il affiche des machines qui n'existent plus réellement — des « VM obsolètes ».

- L'option --prune permet donc de **nettoyer ce cache** pour ne garder que les machines virtuelles réellement présentes.

2. Vagrant workflow :

Workflow de Développement avec Vagrant

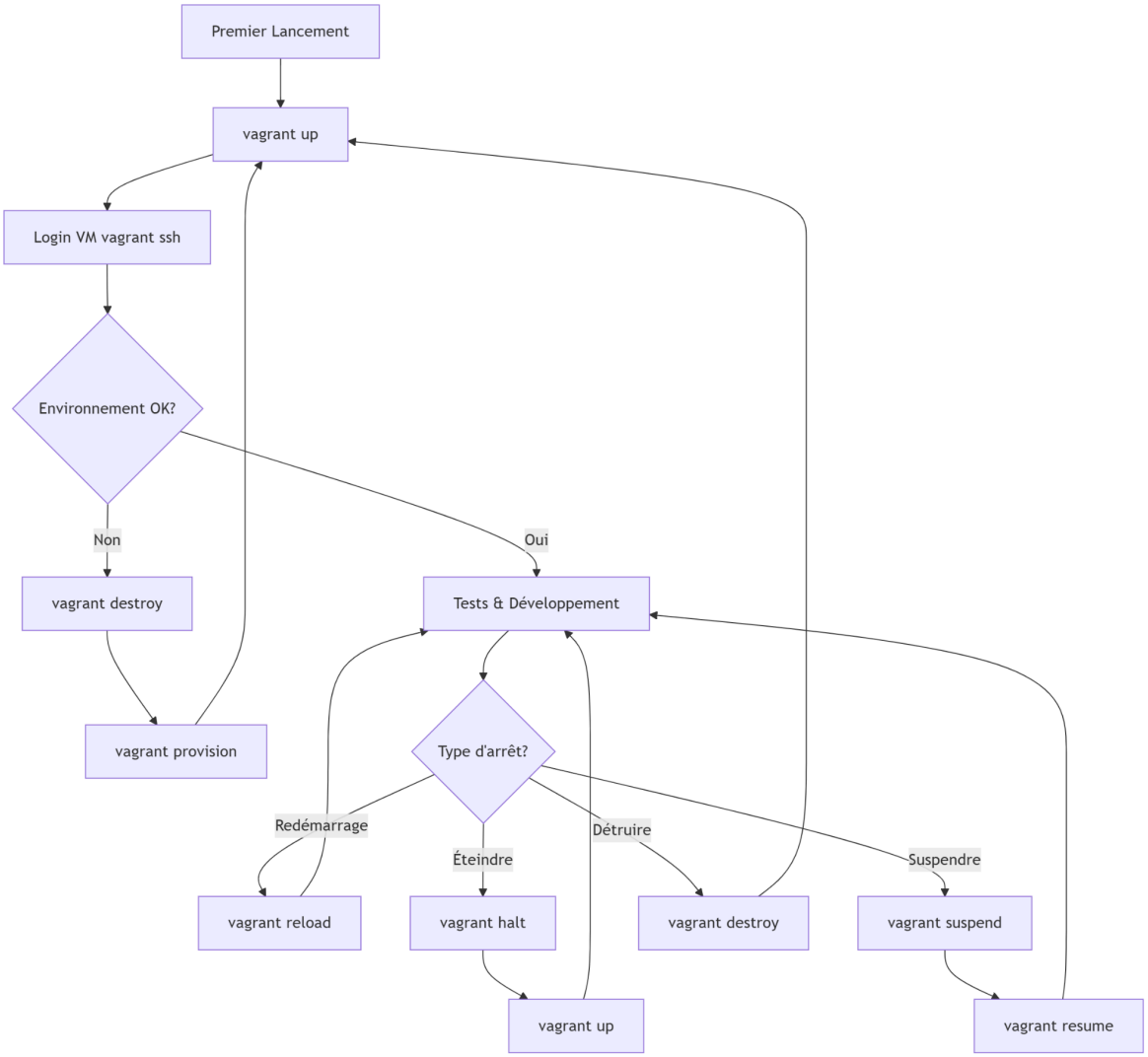
Le processus commence par le démarrage de l'environnement virtuel avec la commande `vagrant up`, qui crée et lance la machine virtuelle. Une fois la VM active, on s'y connecte pour commencer à travailler.

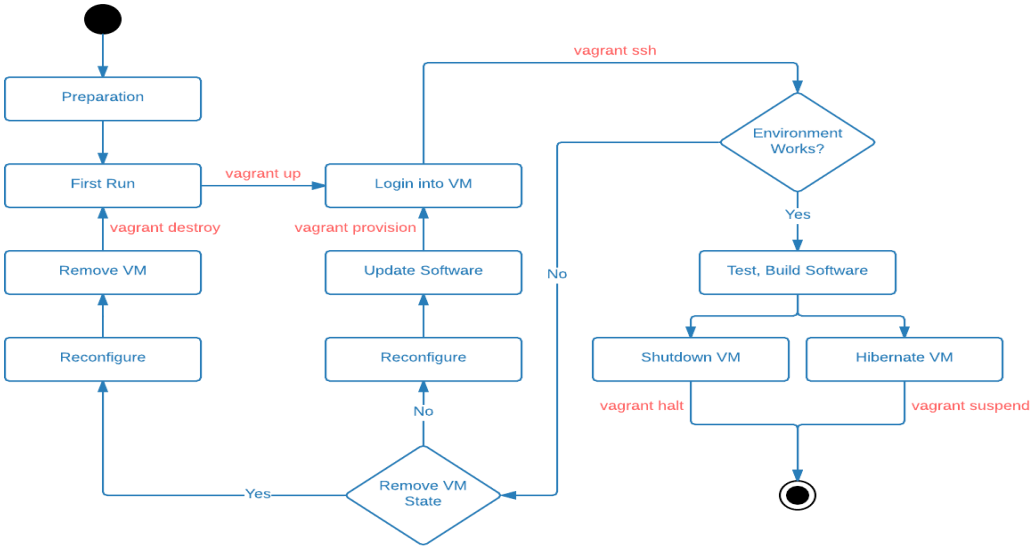
Le cœur du cycle de développement consiste à tester et construire le logiciel, puis à vérifier si l'environnement fonctionne correctement.

Si l'environnement fonctionne (YES) :
On peut soit arrêter proprement la VM avec `vagrant halt`, soit la mettre en veille avec `vagrant suspend` pour une reprise ultérieure plus rapide. À tout moment, on peut se reconnecter à la machine avec `vagrant ssh`.

Si l'environnement rencontre des problèmes (NO) :
On dispose de plusieurs niveaux de correction. Pour les problèmes mineurs, on utilise `vagrant provision` pour reconfigurer la machine sans la recréer. Si cela ne suffit pas, on passe à une reconstruction complète avec `vagrant destroy` qui supprime la VM, suivie d'une mise à jour du logiciel et d'une nouvelle configuration.

Ce workflow itératif permet de maintenir un environnement de développement stable et reproductible, avec des mécanismes de escalade pour résoudre les problèmes efficacement, tout en permettant une gestion flexible des états de la machine virtuelle selon les besoins du développement.


Premier Lancement
vagrant up
Login VM vagrant ssh
Environnement OK?
Non
Oui
vagrant destroy
Tests & Développement
vagrant provision
Type d'arrêt?
Redémarrage
Éteindre
Détruire
Suspendre
vagrant reload
vagrant halt
vagrant destroy
vagrant suspend
vagrant up
vagrant resume

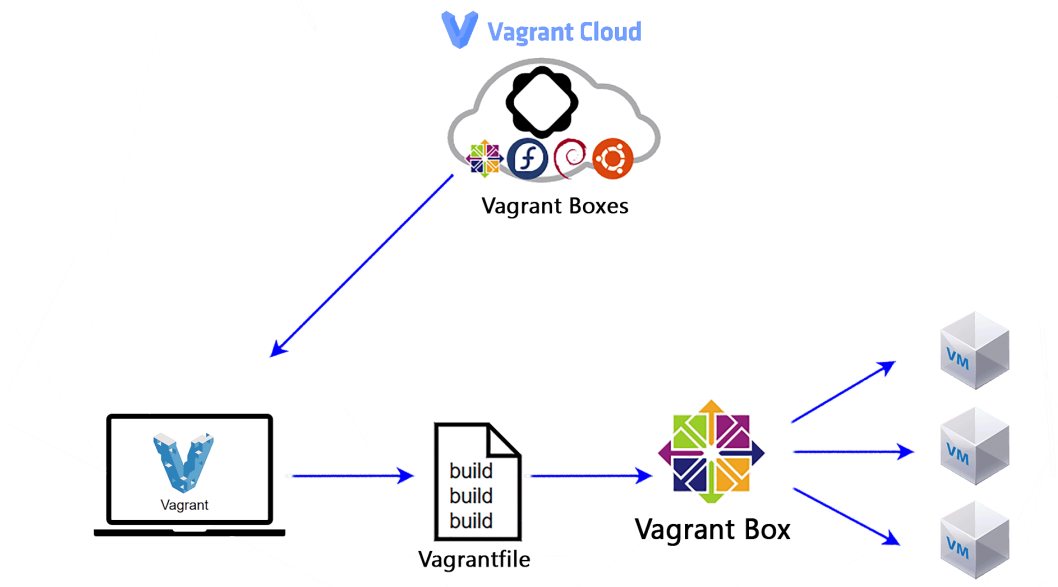


3. Vagrant Boxes :

Les Box sont des machines virtuelles préconfigurées (templates).

Cela devrait accélérer le processus de développement et la distribution de logiciels. Chaque box qui a été utilisée par l'utilisateur est stockée sur l'ordinateur et n'a donc pas besoin d'être re-téléchargée depuis Internet.

Les boîtes peuvent être ajoutées via les commandes vagrant box add [box-name] et vagrant box add [box-url]. On supprime une boîte au moyen de la commande vagrant box remove [box-name].



- Une box peut être utilisée par n'importe qui, sur n'importe quelle plateforme supportée par Vagrant, pour créer un environnement de travail identique
- La façon la plus simple d'utiliser une box est d'ajouter une box depuis le catalogue Vagrant
- Vous pouvez également ajouter et partager vos propres boxes personnalisées sur ce site web
- Les boxes Vagrant sont spécifiques à chaque fournisseur (provider) :
    - Une box pour VirtualBox est incompatible avec VMware Fusion ou tout autre fournisseur
    - Une box doit être installée pour chaque fournisseur
    - Les boxes peuvent partager le même nom tant que les fournisseurs sont différents

4. Les composants de la vagrant box :

Un fichier `.box` de Vagrant est une archive tarball (tar, tar.gz, zip) qui contient toutes les informations nécessaires pour qu'un fournisseur puisse lancer une machine Vagrant.

Il y a quatre composants différents qui constituent une box :

Artéfacts de VM (obligatoire) - Ceci est l'image de la VM et d'autres artéfacts dans le format accepté par le fournisseur pour lequel la box est destinée. Par exemple, une box ciblant le fournisseur VirtualBox pourrait avoir un fichier `.ovf` et quelques fichiers `.vmdk`.

metadata.json (obligatoire) - Contient une map avec des informations sur la box. Plus important encore, le fournisseur cible.

- Provider
- Architecture
- Autres métadonnées

info.json - Ceci est un document JSON qui peut fournir des informations supplémentaires sur la box qui s'affichent lorsqu'un utilisateur exécute `vagrant box list -i`.

- Nom et version de la box
- Description
- Fournisseur supporté (VirtualBox, VMware, etc.)
- URL de vérification

Vagrantfile - Le Vagrantfile intégré dans la box Vagrant fournira quelques valeurs par défaut pour les utilisateurs de la box.

- Paramètres réseau
- Dossiers partagés
- Configuration SSH

Ainsi, si vous extrayez une box et regardez son contenu, cela ressemblera à :

```
>>>

# contenu de la box hashicorp/bionic64

# réf : https://app.vagrantup.com/hashicorp/boxes/bionic64

$ ls hashicorp_bionic_box

Vagrantfile          metadata.json

box.ovf              ubuntu-18.04-amd64-disk001.vmdk
```

5. **Box add et ses options : *vagrant box add ADDRESS***

Cette commande ajoute une box à Vagrant en utilisant l'adresse spécifiée. L'adresse peut être l'une des trois options suivantes :

Types d'adresses supportées :

1. Nom abrégé du catalogue public d'images Vagrant
   Exemple : "hashicorp/bionic64"
2. Chemin de fichier ou URL HTTP vers une box dans un catalogue
   - Authentification basique supportée pour HTTP
   - Variables d'environnement http_proxy respectées
   - HTTPS également supporté
3. URL directe vers un fichier de box
   - Nécessite l'option --name (voir ci-dessous)
   - Les fonctionnalités de versioning et mise à jour ne fonctionneront pas

Si une erreur survient lors du téléchargement ou si celui-ci est interrompu avec Ctrl-C, Vagrant tentera de reprendre le téléchargement la prochaine fois qu'il sera demandé. Cette reprise n'est possible que pendant 24 heures après le téléchargement initial.

--box-version VALEUR

- Version spécifique de la box à ajouter
- Par défaut : dernière version disponible
- Formats acceptés :
  - Version exacte : "1.2.3"
  - Contraintes de version : ">= 1.0, < 2.0"
  - ex: vagrant init ubuntu/trusty64 --box-version 20191107.0.0

6. Vagrantfile

LA hierachie complete de config :

```
config
├── vm                              # Configuration de la machine virtuelle
│   ├── box                         # Nom de la box à utiliser (ex: "hashicorp/bionic64")
│   ├── box_url                     # URL de la box (si non publique)
│   ├── box_version                 # Version spécifique de la box
│   ├── hostname                    # Nom d'hôte de la machine
│   ├── network                     # Définition réseau
│   │   ├── forwarded_port          # Redirection de ports
│   │   │   ├── guest               # Port invité
│   │   │   ├── host                # Port hôte
│   │   │   ├── protocol            # tcp / udp
│   │   │   ├── auto_correct        # Correction auto des collisions
│   │   │   ├── id                  # Nom de la règle
│   │   │   ├── host_ip             # IP d'écoute côté hôte
│   │   │   └── guest_ip            # IP d'écoute côté invité
│   │   ├── private_network          # Réseau interne privé
│   │   │   ├── type                # "dhcp" ou "static"
```

```
│   │   │   ├── type                # "dhcp" ou "static"
│   │   │   ├── ip                  # Adresse IP statique
│   │   │   ├── netmask             # Masque réseau
│   │   │   └── auto_config         # true/false : désactiver config auto
│   │   └── public_network           # Réseau ponté (bridged)
│   │       ├── bridge              # Interface physique à utiliser
│   │       ├── ip                  # IP statique si souhaitée
│   │       ├── use_dhcp_assigned_default_route
│   │       └── auto_config
│   ├── synced_folder               # Dossiers synchronisés
│   │   ├── source                  # Chemin local
│   │   ├── destination             # Chemin dans la VM
│   │   ├── type                    # "virtualbox", "nfs", "rsync"
│   │   ├── create                  # Créer dossier s'il n'existe pas
│   │   ├── owner/group             # Droits sur la VM
│   │   └── mount_options           # Options système de montage
│   ├── provision                   # Provisionnement automatique
│   │   ├── type                    # "shell", "ansible", "puppet", etc.
│   │   ├── inline                  # Commande inline (pour shell)
│   │   ├── path                    # Script externe
│   │   ├── run                     # Quand exécuter (ex: "always")
```

```
|   |   ├── run                    # Quand executer (ex: "always")
|   |   ├── env                    # Variables d'environnement
|   |   └── args                   # Arguments passés au script
|   ├── provider                   # Configuration spécifique à un provider
|   |   ├── "virtualbox"           # Pour VirtualBox :
|   |   |   ├── name               # Nom dans VirtualBox
|   |   |   ├── gui                # true/false (GUI activée)
|   |   |   ├── memory             # RAM allouée
|   |   |   ├── cpus               # Nombre de CPU
|   |   |   ├── linked_clone       # Clone rapide
|   |   |   ├── customize          # Commandes VBoxManage
|   |   |   └── natdnshostresolver1 # DNS NAT
|   |   ├── "hyperv"
|   |   |   ├── memory, cpus, vmname, ip_address
|   |   |   └── integration_services
|   |   └── "libvirt"
|   |       ├── uri, storage_pool, memory, cpus
|   |       └── nested             # Virtualisation imbriquée
|   ├── post_up_message            # Message affiché après démarrage
|   ├── graceful_halt_timeout      # Timeout d'arrêt propre
|   ├── boot_timeout               # Timeout de boot
```

Copier le co

```
│   ├── boot_timeout              # Timeout de boot
│   ├── communicate               # Méthode de communication (ssh/winrm)
│   └── usable_port_range         # Plage de ports valides
│
├── ssh                           # Connexion SSH au guest
│   ├── host                      # IP ou nom d'hôte
│   ├── port                      # Port SSH (défaut 22)
│   ├── username                  # Nom d'utilisateur SSH
│   ├── password                  # Mot de passe SSH
│   ├── private_key_path          # Clé privée utilisée
│   ├── insert_key                # true/false : insérer clé auto
│   ├── forward_agent             # true/false : forward d'agent SSH
│   ├── forward_x11               # true/false : X11 forwarding
│   ├── keep_alive                # Paquets keep-alive
│   ├── proxy_command             # Commande proxy SSH
│   ├── compression               # Compression SSH
│   ├── verify_host_key           # :never / :always
│   └── connect_timeout           # Timeout de connexion
│
├── vagrant                       # Configuration interne de Vagrant
```

```
├── vagrant                    # Configuration interne de Vagrant
│   ├── host                   # Type d'hôte (:detect, :windows, etc.)
│   ├── plugins                # Plugins à installer
│   │   ├── version            # Version du plugin
│   │   ├── entry_point        # Fichier d'entrée Ruby
│   │   └── sources            # Sources de téléchargement
│   └── sensitive              # Variables à masquer dans logs
│
├── winrm                      # Connexion Windows (équivalent SSH)
│   ├── username
│   ├── password
│   ├── host
│   ├── port
│   ├── timeout
│   ├── transport              # http / https
│   └── ssl_peer_verification  # true/false
│
└── trigger                    # Déclencheurs d'événements
    ├── before                 # Avant une commande Vagrant
    ├── after                  # Après une commande
```

a. Pourquoi le vagrantfile ?

La **configuration d'une machine virtuelle (VM)** utilisée dans un projet est **stockée dans un fichier nommé Vagrantfile**, présent **dans chaque projet**.

- Chaque projet possède **son propre Vagrantfile**, qui définit les **propriétés** de la machine virtuelle utilisée (comme la mémoire, le réseau, les dossiers partagés, etc.).

- Ce fichier est écrit en **langage Ruby** et contient notamment le **nom de la "base box"**, c'est-à-dire l'image de base du système d'exploitation invité (guest OS) sur laquelle la VM sera construite.

- Le Vagrantfile doit être **partagé entre tous les développeurs** d'un même projet, il faut donc **l'ajouter au système de gestion de versions** (comme Git).
  Ainsi, si tous les développeurs utilisent le même Vagrantfile, **leurs environnements de développement seront identiques**, ce qui évite les différences de configuration d'un poste à l'autre.
  -

b. config:

```
Vagrant::Config.run do |config| ❶
  config.vm.box = "precise64" ❷

  config.vm.share_folder "v-root", "/vagrant", "." ❸

  config.vm.provision "shell" do |s| ❹
    s.path = "script.sh"
  end
end ❺
```

Explication ligne par ligne :

1. **Vagrant::Config.run do |config|**
   Cette ligne **démarre un bloc de configuration Vagrant**.
   Le |config| signifie que, dans ce bloc, on peut utiliser la variable config pour définir les paramètres de la machine virtuelle.
   (C'est une syntaxe Ruby standard : le bloc do ... end crée une sorte d'espace de configuration temporaire.)

2. **config.vm.box = "precise64"**
   Ici, on définit la **"base box"** de la VM, c'est-à-dire **l'image du système d'exploitation** que Vagrant utilisera.
   Dans ce cas, la box s'appelle "precise64" (Ubuntu 12.04 en version 64 bits).

3. **config.vm.share_folder "v-root", "/vagrant", "."**
   Cette ligne indique qu'un **dossier partagé** doit être créé entre la machine hôte (ton ordinateur) et la machine virtuelle.

   - "v-root" : le nom interne du dossier partagé
   - "/vagrant" : l'emplacement du dossier dans la machine virtuelle
   - "." : le dossier courant sur l'hôte (le projet local)

4. Cette instruction est une **fonction** (et non une simple affectation), qui prend ici trois **paramètres**.

5. **config.vm.provision "shell" do |s| ... end**
   Cette portion définit un **provisionnement**, c'est-à-dire un script qui sera exécuté **automatiquement à la création de la VM**.

Ici, on utilise un **provisionneur de type "shell"**, et dans le bloc suivant :

- s.path = "script.sh" indique le **chemin vers le script** à exécuter.

6. **end (la dernière ligne)**
   Elle **ferme le bloc de configuration** commencé à la première ligne.

c. config.vm:

config.vm est le **cœur de la configuration de la machine virtuelle** dans Vagrant.
C'est ici qu'on définit :

- quelle box utiliser,
- comment la machine démarre,
- quels dossiers et réseaux sont partagés,
- quels scripts s'exécutent,
- et comment Vagrant communique avec la VM.

Les paramètres contenus dans config.vm servent à **configurer la machine virtuelle** que Vagrant crée et gère.

Chaque option permet d'ajuster un aspect du comportement de la VM, de son démarrage à son réseau, en passant par la synchronisation des dossiers ou les téléchargements de box…

Ici config est vu comme un objet ayant plusieurs fonctions ou méthodes.

Les différentes fonctions inclues dans config:

1. **Gestion du système de fichiers**

- **config.vm.allow_fstab_modification** :
  Si true, Vagrant peut **modifier le fichier /etc/fstab** dans la machine virtuelle pour monter automatiquement les dossiers partagés.
  Si false, Vagrant ne touche pas à ce fichier (les dossiers ne seront alors pas montés après un redémarrage).
  Valeur par défaut : true.

- **config.vm.allow_hosts_modification** :
  Si false, Vagrant **n'écrit pas dans /etc/hosts** (le fichier de résolution d'hôtes).

Valeur par défaut : true.

- **config.vm.allowed_synced_folder_types** :
  Permet de **restreindre les types de dossiers synchronisés** autorisés (par exemple nfs, virtualbox, etc.).
  On fournit ici une **liste de plugins** acceptés.

2. Réseau et interfaces

- **config.vm.base_mac / config.vm.base_address** :
  Permettent de **définir l'adresse MAC** ou **l'adresse IP** de l'interface NAT principale de la machine.
  Ces options dépendent du **fournisseur** (VirtualBox, VMware, etc.).

3. Démarrage et arrêt

- **config.vm.boot_timeout** :
  Temps d'attente (en secondes) pendant lequel Vagrant attend que la machine démarre et devienne accessible.
  Par défaut : 300 secondes (5 minutes).

- **config.vm.graceful_halt_timeout** :
  Temps d'attente (en secondes) pour un arrêt "propre" quand on exécute vagrant halt.
  Par défaut : 60 secondes.

4. Gestion de la box

- **config.vm.box** :
  Nom de la *box* à utiliser (l'image de base du système).
  Exemples : "ubuntu/bionic64", "centos/7".

- **config.vm.box_architecture** :
  Architecture processeur de la box (amd64, arm64, etc.).
  :auto détecte automatiquement celle de l'hôte.
  Par défaut : :auto.

- **config.vm.box_check_update** :
  Si true, Vagrant **vérifie les mises à jour de la box** à chaque vagrant up.
  Par défaut : true.

- **config.vm.box_url** :
  Lien (ou liste de liens) vers l'endroit où la box peut être téléchargée.

Peut être une URL HTTP(s) ou un fichier local (file://).

- **config.vm.box_version** :
  Version spécifique de la box à utiliser (par exemple "= 1.2.0" ou ">= 1.0, < 2.0").
  Par défaut : la dernière version disponible.

5. **Options de téléchargement** (box_download_*) :

   - config.vm.box_download_checksum et config.vm.box_download_checksum_type : permettent à Vagrant de **vérifier l'intégrité** de la box téléchargée (avec SHA ou MD5).

   - config.vm.box_download_insecure : désactive la **vérification SSL** (non recommandé).

   - config.vm.box_download_client_cert, config.vm.box_download_ca_cert, etc. : définissent les certificats à utiliser pour les téléchargements sécurisés.

6. **Cloud et initialisation**

- **config.vm.cloud_init** :
  Contient les paramètres pour la configuration via **cloud-init** (outil d'initialisation des machines dans le cloud).

- **config.vm.cloud_init_first_boot_only** :
  Si true, cloud-init ne s'exécute **qu'au premier démarrage**.
  Par défaut : true.

7. **Connexion et communication**

- **config.vm.communicator** :
  Définit le **type de communication** entre Vagrant et la VM.

   - "ssh" pour Linux (par défaut)

   - "winrm" pour Windows

8. **Système invité (Guest)**

- **config.vm.guest** :
  Spécifie le **type de système d'exploitation invité** (:linux, :windows, etc.).
  Aide Vagrant à adapter ses commandes selon le système.

- **config.vm.hostname** :
  Définit le **nom d'hôte (hostname)** de la machine virtuelle.

- **config.vm.ignore_box_vagrantfile** :
  Si true, ignore le Vagrantfile inclus dans la box de base.
  Par défaut : false.

9. **Réseau et dossiers partagés**

- **config.vm.network** :
  Configure les **interfaces réseau** (NAT, privé, public, etc.).

- **config.vm.synced_folder** :
  Configure les **dossiers synchronisés** entre l'hôte et la VM.

- **config.vm.usable_port_range** :
  Définit la **plage de ports** que Vagrant peut utiliser (par défaut : 2200..2250).

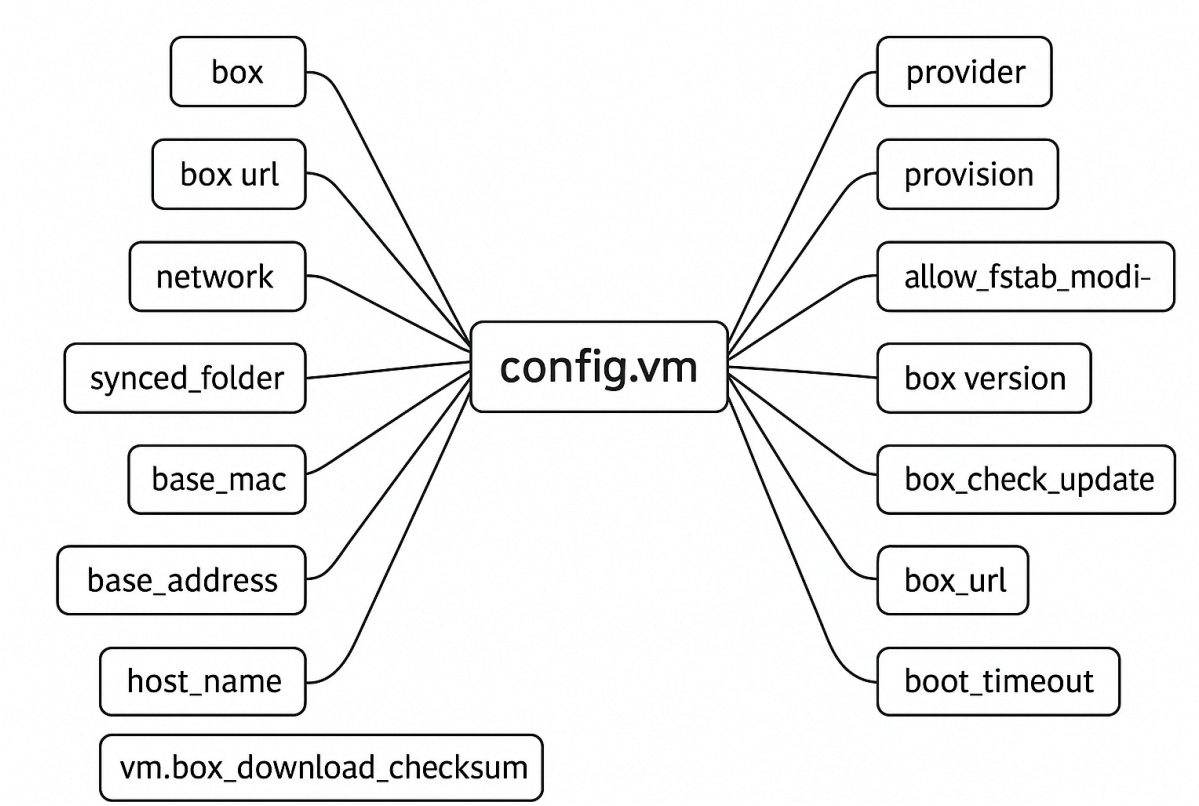
10. **Fournisseurs et provisioning**

- **config.vm.provider** :
  Permet de définir des **paramètres spécifiques à un fournisseur** (VirtualBox, VMware, etc.).

- **config.vm.provision** :
  Sert à **automatiser l'installation ou la configuration** de logiciels lors de la création de la machine (via script, Ansible, etc.).

11. **Messages et personnalisation**

- **config.vm.post_up_message** :
  Affiche un **message personnalisé** après le lancement de la machine (vagrant up), utile pour indiquer des instructions (par exemple : "Accédez à http://localhost:8080").



d. Vagrant.ssh :

Les paramètres **config.ssh** contrôlent la manière dont Vagrant se connecte à la machine virtuelle par SSH.
Même si les valeurs par défaut conviennent souvent, certaines options sont essentielles à connaître et à ajuster selon le contexte.

1. **config.ssh.username** : définit l'utilisateur utilisé pour la connexion SSH. Par défaut, c'est *vagrant*.

2. **config.ssh.port** : indique le port de connexion SSH, généralement *22*.

3. **config.ssh.host** : précise le nom d'hôte ou l'adresse IP de la machine, souvent détecté automatiquement par Vagrant.

4. **config.ssh.private_key_path** : indique le chemin vers la clé privée utilisée pour se connecter à la VM. Si vous créez vos propres boxes, vous pouvez y spécifier votre clé.

5. **config.ssh.insert_key** : lorsqu'il est activé (valeur par défaut), Vagrant remplace la clé SSH "insecure" par une clé générée automatiquement pour plus de sécurité.

6. **config.ssh.keep_alive** : permet d'envoyer des paquets "keep-alive" pour éviter que la connexion SSH ne se coupe.

7. **config.ssh.forward_agent** : autorise le transfert de votre agent SSH local, pratique pour utiliser vos clés locales à l'intérieur de la VM.

8. **config.ssh.forward_env** : permet de transférer certaines variables d'environnement de l'hôte vers la machine virtuelle.

9. **config.ssh.connect_timeout** et **config.ssh.connect_retries** : déterminent respectivement le temps d'attente maximal et le nombre de tentatives pour établir une connexion SSH.

10. **config.ssh.verify_host_key** : règle le niveau de vérification de la clé d'hôte SSH (par défaut, aucune vérification stricte).

11. **config.ssh.shell** : définit le shell utilisé pour exécuter les commandes via SSH, *bash -l* par défaut.

12. **config.ssh.sudo_command** : personnalise la commande utilisée lorsque Vagrant exécute des actions nécessitant *sudo*.

e. config.vagrant :

Les paramètres contenus dans **config.vagrant** permettent de **modifier le comportement global de Vagrant lui-même**, et non celui de la machine virtuelle (comme c'est le cas pour config.vm ou config.ssh).

- **config.vagrant.host**

Ce paramètre définit **le type de machine hôte** (le système sur lequel Vagrant est exécuté).

Par défaut, il est réglé sur :detect, ce qui signifie que **Vagrant détecte automatiquement** le système hôte (Windows, macOS, Linux, etc.).

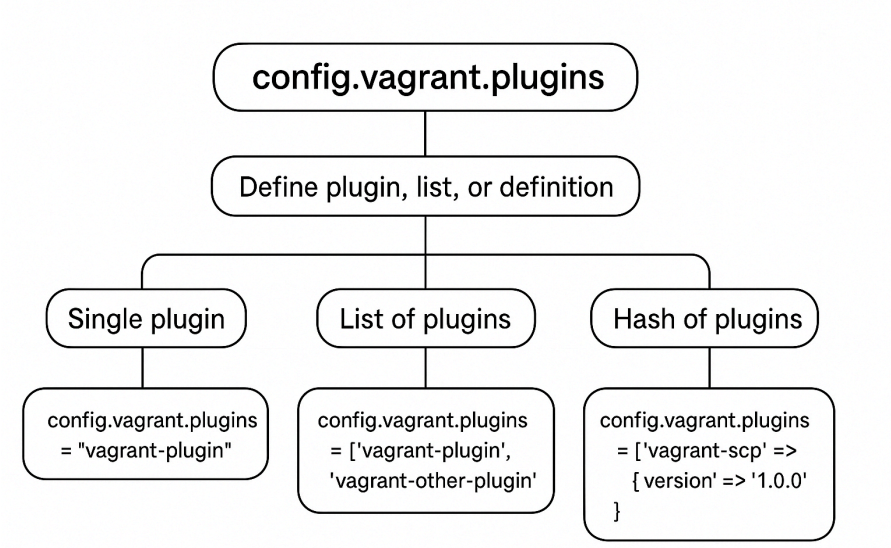
Cette information est nécessaire pour certaines opérations spécifiques, comme la configuration automatique des dossiers partagés avec NFS.

En général, vous n'avez **pas besoin de le modifier**, sauf si la détection automatique échoue.

- **config.vagrant.plugins**

Ce paramètre indique **quels plugins doivent être installés** pour le projet.

Vagrant vérifiera que ces plugins sont bien présents, et **les installera automatiquement** si ce n'est pas le cas.



- **config.vagrant.sensitive** ❗

Ce paramètre permet de **masquer certaines valeurs sensibles** dans la sortie de Vagrant (comme des mots de passe ou des tokens).

Ces valeurs seront retirées de l'affichage dans le terminal et des journaux.

Très utile pour **protéger les informations confidentielles** lors du déploiement ou du partage du projet.

f. config.winrm:

Ces paramètres **config.winrm** permettent de configurer la manière dont Vagrant accède à votre machine virtuelle Windows via WinRM. Comme pour la plupart des paramètres Vagrant, les valeurs par défaut conviennent généralement, mais vous pouvez les personnaliser selon vos besoins.

Ces paramètres ne sont utilisés que si vous avez défini votre type de communicateur sur :winrm.

g. config.winssh

Le communicateur WinSSH est conçu spécifiquement pour le portage natif d'OpenSSH sous Windows. Il ne dépend pas d'un environnement de type POSIX, ce qui élimine la nécessité d'installer des logiciels supplémentaires (comme Cygwin) pour un fonctionnement optimal.

h. Les variables dans le vagrantfile

La gestion des variables d'environnement dans Vagrant permet de personnaliser le comportement des machines virtuelles **sans modifier le Vagrantfile à chaque changement**. Cette approche offre une flexibilité accrue pour créer des configurations réutilisables et adaptables à différents environnements.

- Les variables locales :

```
# Variables de configuration principales
VM_BOX = "ubuntu/focal64"
VM_MEMORY = 2048
VM_CPUS = 2
VM_HOSTNAME = "mon-serveur-dev"
VM_IP = "192.168.56.10"
```

Et apres on peut les utiliser dans les config :

```ruby
Vagrant.configure("2") do |config|
  # Configuration de base
  config.vm.box = VM_BOX
  config.vm.hostname = VM_HOSTNAME

  # Configuration réseau
  config.vm.network "private_network", ip: VM_IP
  config.vm.network "forwarded_port", guest: 80, host: 8080, auto_correct: true

  # Configuration du provider VirtualBox
  config.vm.provider "virtualbox" do |vb|
    vb.memory = VM_MEMORY
    vb.cpus = VM_CPUS
    vb.name = VM_HOSTNAME
    vb.gui = false
  end
```

- Les variables d'environnement :

  Style d’architecture de fichier

```
text


mon-projet/
├── Vagrantfile
├── .env                    # Variables d'environnement (optionnel)
├── .env.example           # Modèle des variables requises
├── scripts/
│   └── provision.sh
└── .gitignore
```

Implémentation pratique:

```ruby
# Au début du Vagrantfile
begin
  require 'dotenv'
  Dotenv.load
rescue LoadError
  puts "dotenv non installé, continuation sans variables .env"
end

Vagrant.configure("2") do |config|
  # Variables avec valeurs par défaut
  box_name = ENV.fetch('VAGRANT_BOX', 'ubuntu/focal64')
  memory_size = ENV.fetch('VAGRANT_MEMORY', 1024).to_i
  cpu_count = ENV.fetch('VAGRANT_CPUS', 1).to_i

  config.vm.box = box_name
  config.vm.hostname = ENV.fetch('VAGRANT_HOSTNAME', 'vm-dev')
```

Meilleures pratiques et conseils de performance

- Validez les variables requises à l'exécution et définissez des valeurs par défaut : utilisez-les `ENV.fetch('VAR','default')`pour éviter les valeurs nulles.
- Mettez en cache localement les images de boîtes fréquemment utilisées pour accélérer le chargement `vagrant up`.
- Regroupez les variables d'environnement associées dans un `.env`fichier et ajoutez-le au répertoire de configuration `.gitignore`pour éviter les fuites de mémoire.

Considérations de sécurité

- Limitez les permissions d'accès aux fichiers `.env` `chmod 600 .env`.
- Évitez de stocker les informations confidentielles en clair ; utilisez Vault ou le chiffrement des identifiants de Vagrant.
- Exécutez les commandes Vagrant avec les privilèges minimaux pour éviter toute modification indésirable de l'hôte.

i. vagrant validate

Cette commande vous dira si le fichier Vagrantfile qui se trouve dans le dossier avec est valide ou pas. Elle soulignera les erreurs au cas contraire.

7. Les réseaux dans vagrant

a. Le port forwarding :
Les ports redirigés dans **Vagrant** permettent de **rediriger un port du système hôte vers un port de la machine virtuelle (guest)**, en utilisant le protocole **TCP** ou **UDP**.

```
Vagrant.configure("2") do |config|
  config.vm.network "forwarded_port", guest: 80, host: 8080
end
```

This will allow accessing port 80 on the guest via port 8080 on the host.

Par défaut, la redirection de port s'applique à toutes les interfaces réseau de l'hôte. Pour **restreindre l'accès**, vous pouvez préciser host_ip ou guest_ip.

Voici les principales options configurables pour forwarded_port :

- **guest (int)** : port sur la machine virtuelle (ex. 80).
- **host (int)** : port sur la machine hôte (ex. 8080).
- **auto_correct (booléen)** : ajuste automatiquement le port hôte s'il est déjà utilisé (défaut : *false*).
- **guest_ip (string)** : adresse IP du guest sur laquelle lier le port (par défaut : toutes les interfaces).
- **host_ip (string)** : adresse IP de l'hôte à laquelle le port est lié (par défaut : toutes les interfaces).
- **protocol (string)** : protocole utilisé, "tcp" (par défaut) ou "udp".
- **id (string)** : identifiant du mappage (utile pour le voir dans VirtualBox, ex. "tcp80").

Gestion des collisions de ports

Lorsque plusieurs machines Vagrant utilisent le même port hôte (ex. 8080), une **collision** peut se produire.
Vagrant détecte automatiquement ces conflits et peut les **corriger** si vous l'y autorisez :

```
Vagrant.configure("2") do |config|
  config.vm.network "forwarded_port", guest: 80, host: 8080,
    auto_correct: true
end
```

The final `:auto_correct` parameter set to true tells Vagrant to auto correct any collisions. During a `vagrant up` or `vagrant reload`, Vagrant will output information about any collisions detections and auto corrections made, so you can take notice and act accordingly.

b. Les réseaux privés :

Un **réseau privé** dans **Vagrant** permet d'attribuer une **adresse IP locale** à votre machine virtuelle, **non accessible depuis Internet**.

Cela permet à votre VM d'être joignable uniquement depuis votre machine hôte ou d'autres VM sur le même réseau privé.

Via DHCP :

```
Vagrant.configure("2") do |config|
  config.vm.network "private_network", type: "dhcp"
end
```

This will automatically assign an IP address from the reserved address space. The IP address can be determined by using `vagrant ssh` to SSH into the machine and using the appropriate command line tool to find the IP, such as `ifconfig` or `ip addr show`.

Via ip static :

```
Vagrant.configure("2") do |config|
  config.vm.network "private_network", ip: "192.168.50.4"
end
```

It is up to the users to make sure that the static IP does not collide with any other machines on the same network.

c. Les réseaux publics :

Un **réseau public** dans Vagrant permet à une machine virtuelle d'être **accessible depuis le réseau local** (et parfois au-delà).

Contrairement aux **réseaux privés**, qui isolent la machine, le réseau public **la connecte directement au réseau de l'hôte**, comme si c'était un autre ordinateur du même LAN.

Les **Vagrant boxes** sont **insecure par défaut** :

- mot de passe par défaut public (vagrant / vagrant),
- clé SSH connue,
- accès root facile.

Cela signifie que **toute personne sur le même réseau** peut potentiellement accéder à votre VM.
Avant d'activer un réseau public, **vérifiez les paramètres de sécurité** de votre box et ne l'utilisez **jamais sur un réseau non fiable**.

Dans certains cas, il est nécessaire de conserver la route par défaut attribuée par DHCP. Il est alors possible de spécifier l'`use_dhcp_assigned_default_route` option correspondante. Par exemple :

```
Vagrant.configure("2") do |config|
  config.vm.network "public_network",
    use_dhcp_assigned_default_route: true
end
```

Choix de l'interface réseau (bridge)

Quand plusieurs interfaces réseau existent sur la machine hôte (Wi-Fi, Ethernet, etc.), Vagrant vous demandera sur laquelle **"faire le pont" (bridge)**.

Vous pouvez définir une interface par défaut :

```
config.vm.network "public_network", bridge: "en1: Wi-Fi (AirPort)"
```

La chaîne de caractères identifiant l'interface souhaitée doit correspondre exactement au nom d'une interface disponible. Si elle est introuvable, Vagrant vous proposera de choisir parmi une liste d'interfaces réseau disponibles.

Chez certains fournisseurs, il est possible de spécifier une liste d'adaptateurs à utiliser :

```
config.vm.network "public_network", bridge: [
  "en1: Wi-Fi (AirPort)",
  "en6: Broadcom NetXtreme Gigabit Ethernet Controller",
]
```

NB: Il est possible de configurer manuellement l'interface réseau de la vm dans le cas d'un OS linux par exemple et pour cela il suffit de désactiver la configuration automatique :

```
Vagrant.configure("2") do |config|
  config.vm.network "public_network", auto_config: false
end
```

Le provisionneur de shell peut ensuite être utilisé pour configurer l'adresse IP de l'interface :

```
Vagrant.configure("2") do |config|
  config.vm.network "public_network", auto_config: false

  # manual ip
  config.vm.provision "shell",
    run: "always",
    inline: "ifconfig eth1 192.168.0.17 netmask 255.255.255.0 up"

  # manual ipv6
  config.vm.provision "shell",
    run: "always",
    inline: "ifconfig eth1 inet6 add fc00::17/7"
end
```

8. Multi-machine

Vagrant permet de définir et gérer plusieurs machines virtuelles au sein d'un même Vagrantfile, créant ainsi un environnement multi-machines. Cette fonctionnalité est particulièrement utile pour :

- Reproduire des architectures de production (serveur web + base de données séparés)
- Modéliser des systèmes distribués et leurs interactions
- Tester des interfaces (API, services)
- Simuler des scénarios de crise (panne réseau, latences, etc.)

Contrairement aux approches traditionnelles qui regroupent tous les services sur une seule machine, Vagrant offre un modèle plus fidèle à la réalité.

a. config.vm.define

   Plusieurs machines virtuelles sont définies dans le même fichier Vagrantfile de projet grâce à la méthode config.vm.define. Cette directive de configuration est un peu

particulière, car elle crée une configuration Vagrant à l'intérieur d'une autre. L'exemple suivant illustre ce principe :

```
Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline: "echo Hello"

  config.vm.define "web" do |web|
    web.vm.box = "apache"
  end

  config.vm.define "db" do |db|
    db.vm.box = "mysql"
  end
end
```

- config.vm.define "web" crée une **machine nommée “web”**.
- Le bloc do |web| ... end contient les paramètres spécifiques à cette machine.
- Tout ce qui est défini dans config (à l’extérieur des blocs) s’applique **à toutes les machines**.

En Ruby, c’est une question de **portée (scope)** : la configuration générale (config) est héritée, puis la configuration spécifique (web ou db) vient la compléter.

```
Vagrant.configure("2") do |config|
  config.vm.provision :shell, inline: "echo A"

  config.vm.define :testing do |test|
    test.vm.provision :shell, inline: "echo B"
  end

  config.vm.provision :shell, inline: "echo C"
end
```

Lorsqu'on utilise ces étendues, l'ordre d'exécution des éléments tels que les provisionneurs devient important. Vagrant impose un ordre d'exécution de

l'extérieur vers l'intérieur, conformément à l'ordre indiqué dans le fichier Vagrantfile. Par exemple, avec le fichier Vagrantfile ci-dessous :

Dans ce cas, les modules de provisionnement afficheront d'abord « A », puis « C », puis « B ». Notez que « B » est affiché en dernier. Cela s'explique par le fait que l'ordre de traitement est de l'extérieur vers l'intérieur, conformément à l'ordre du fichier.

b. contrôler plusieurs machines

Plusieurs configurations

```
Vagrant.configure("2") do |configuration1|
 # configuration de la machine 1
end

Vagrant.configure("2") do |configuration2|
 # configuration de la machine 2
end

Vagrant.configure("2") do |configuration3|
 # configuration de la machine 3
end
```

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

  config.vm.define "machine1" do |configuration1|
    # configuration de la machine 1
  end

  config.vm.define "machine2" do |configuration2|
    # configuration de la machine 2
  end
end
```

Cette approche plus bas definit une liste de machines avec leurs configurations. Dans la configuration de Vagrant, nous utilisons une boucle pour traverser chaque element de la liste et utiliser chaque configuration pour lancer une machine.

```ruby
machines = [
  { :hostname => "VM1", :ip => "10.0.0.10" },
  { :hostname => "VM2", :ip => "10.0.0.11" },
  { :hostname => "VM3", :ip => "10.0.0.12" }
]

# Variables locales
BOX = "ubuntu/xenial64"
BOX_VERSION = "20211001.0.5"
CPU = 1
RAM = 1024

Vagrant.configure("2") do |config|
  machines.each do |machine|
    config.vm.define machine[:hostname] do |node|
      node.vm.box = BOX
      node.vm.box_version = BOX_VERSION
      node.vm.hostname = machine[:hostname]
      node.vm.network "private_network", ip: machine[:ip]

      node.vm.provider "virtualbox" do |vb|
        vb.gui = false
        vb.memory = RAM
        vb.cpus = CPU
      end

      node.hostsupdater.aliases = [machine[:hostname]]
    end
  end
end
```

```
RAM = 1024
CPU = 1

Vagrant.configure("2") do |config|
    # Configure load balancer machine
     config.vm.define "lb" do |lb|
         lb.vm.box = "ubuntu/xenial64"
         lb.vm.network "private_network", ip: "10.0.0.10"
     end
     config.vm.provider "virtualbox" do |v|
       v.memory = RAM
       v.cpus = CPU
   end
     # Configure first web machine
     config.vm.define "web1" do |web1|
         web1.vm.box = "ubuntu/xenial64"
         web1.vm.network "private_network", ip: "10.0.0.11"
     end
     config.vm.provider "virtualbox" do |v|
       v.memory = RAM
       v.cpus = CPU
     end
```

Au moment où plus d'une machine est définie dans un **Vagrantfile**, l'utilisation des différentes commandes **Vagrant** change légèrement. Ce changement reste cependant assez intuitif.

Les commandes qui n'ont de sens que lorsqu'elles ciblent une seule machine, comme vagrant ssh, nécessitent désormais le nom de la machine à contrôler. En reprenant l'exemple ci-dessus, vous diriez donc vagrant ssh web ou vagrant ssh db.

D'autres commandes, comme vagrant up, s'appliquent par défaut à **toutes les machines**. Ainsi, si vous exécutez vagrant up, Vagrant lancera à la fois la machine **web** et la machine **db**. Vous pouvez également être plus précis et exécuter vagrant up web ou vagrant up db.

De plus, il est possible de spécifier une **expression régulière** pour ne faire correspondre que certaines machines. Cela peut être utile dans certains cas où vous définissez plusieurs machines similaires — par exemple, si vous testez un service distribué avec une machine principale (**leader**) et plusieurs machines secondaires (**follower0**, **follower1**, **follower2**, etc.).
Si vous souhaitez démarrer uniquement les machines « follower » sans lancer la machine « leader », vous pouvez simplement exécuter :
vagrant up /follower[0-9]/.
Lorsque Vagrant voit un nom de machine encadré par des barres obliques (/), il suppose que vous utilisez une **expression régulière**.

c. Communication entre machines

Pour faciliter la communication entre les machines dans une configuration multi-machines, il convient d'utiliser les différentes options de réseau . En particulier, un réseau privé permet de créer un réseau privé entre plusieurs machines et l'hôte.

Exemple :

```ruby
config.vm.define "web" do |web|
  web.vm.network "private_network", ip: "192.168.50.10"
end

config.vm.define "db" do |db|
  db.vm.network "private_network", ip: "192.168.50.11"
end
```

> ✅ Ainsi, le serveur web peut contacter la base de données via `192.168.50.11`.

d. spécification d'une machine leader

```ruby
config.vm.define "web", primary: true do |web|
  # ...
end
```

> Par exemple, `vagrant ssh` sans argument se connectera automatiquement à la machine principale.

e. Machine à démarrage

Par défaut, dans un environnement multi-machines, vagrant upVagrant démarrera toutes les machines définies. Ce autostartparamètre permet d'indiquer à Vagrant de *ne pas* démarrer certaines machines. Exemple :

```
config.vm.define "web"
config.vm.define "db"
config.vm.define "db_follower", autostart: false
```

Avec vagrant uples paramètres ci-dessus, Vagrant démarrera automatiquement les machines « web » et « db », mais pas la machine « db_follower ». Vous pouvez forcer manuellement le démarrage de la machine « db_follower » en exécutant la commande suivante : `vagrant start` vagrant up db_follower.

. Hierachie type de config :

config

├── vm

│ ├── box

│ ├── hostname

│ ├── network

│ ├── synced_folder

│ ├── provision

│ └── provider

├── ssh

├── vagrant

├── winrm

└── trigger

9. Plugins

Vagrant intègre de nombreuses fonctionnalités prêtes à l'emploi pour faciliter la mise en place de vos environnements. Cependant, il arrive que l'on souhaite modifier le fonctionnement de Vagrant ou ajouter des fonctionnalités. C'est possible grâce aux *plugins* Vagrant .

Les plugins sont des éléments puissants et essentiels qui étendent Vagrant grâce à une API stable et bien documentée, capable de supporter les mises à jour majeures.

En réalité, la majeure partie du noyau de Vagrant est implémentée à l'aide de plugins .

a. Le plugin vagrant-hostsupdater :

Le plugin **vagrant-hostsupdater** sert à **modifier automatiquement le fichier /etc/hosts de ta machine hôte** (ton ordinateur, pas la VM)

afin que tu puisses accéder à tes machines virtuelles **via un nom de domaine lisible**, au lieu d'une adresse IP.

1. Pendant vagrant up

Quand tu démarres tes VM avec :

```
vagrant up
```

le plugin :

- lit les alias définis dans ton Vagrantfile (node.hostsupdater.aliases),
- récupère les IP privées des machines (node.vm.network "private_network", ip: ...),
- puis **ajoute automatiquement des lignes** dans ton fichier /etc/hosts.

Exemple :

```
10.0.0.10  VM1
10.0.0.11  VM2
10.0.0.12  VM3
```

Ainsi, tu peux faire :

```
ping VM1
```

ou ouvrir dans ton navigateur :

http://VM2/

➡️ Sans devoir te souvenir de 10.0.0.11.

🔹 2. Pendant vagrant halt, destroy ou suspend

Le plugin **nettoie** le fichier /etc/hosts :

- il supprime les entrées qu'il avait ajoutées,
- il ne touche pas aux autres lignes de ton fichier.

Cela évite que ton /etc/hosts se remplisse de vieilles IP inutiles.

10. Les fichiers sous vagrant (intéressants)

Les dossiers synchronisés permettent à Vagrant de synchroniser un dossier de la machine hôte avec la machine invitée, ce qui vous permet de continuer à travailler sur les fichiers de votre projet sur votre machine hôte, tout en utilisant les ressources de la machine invitée pour compiler ou exécuter votre projet: **on peut éditer du code avec notre IDE sur notre PC, et l'exécuter dans l'environnement isolé de la VM**.

Par défaut, Vagrant partagera votre répertoire de projet (le répertoire contenant le Vagrantfile) avec `/vagrant`.

a. utilisation basique :

Quand tu fais vagrant up, Vagrant :

1. crée /srv/website dans la VM (s'il n'existe pas),
2. y **monte** ton dossier src/ local.

```ruby
Vagrant.configure("2") do |config|
  # other config here

  config.vm.synced_folder "src/", "/srv/website"
end
```

Un exemple complet :

🧱 Exemple plus complet

```ruby
config.vm.synced_folder "src/", "/srv/website",
  owner: "vagrant",
  group: "www-data",
  create: true,
  mount_options: ["dmode=775", "fmode=664"]
```

- mount_options sont des options de montages dans l'exemple plus, les mount_options définissent les droits unix sur les dossiers (dmode) et sur les fichiers (fmode), un exemple plus bas avec des options pour restreindre l'utilisation des fichiers partagé qu'à certaines personnes :

👥 Droits utilisateurs : attention aux priorités

Si tu mets à la fois :

ruby Copier le code

```ruby
owner: "vagrant",
group: "vagrant",
mount_options: ["uid=1234", "gid=1234"]
```

➡️ Les options `uid/gid` **dans mount_options** prennent le dessus sur `owner` et `group`.

11. Les provisionner

Les *provisioners* (outils de provisionnement) dans **Vagrant** permettent d'automatiser l'installation de logiciels, la modification de configurations et bien plus encore, au cours du processus d'exécution de la commande vagrant up.

Cette fonctionnalité est particulièrement utile, car les *boxes* (images de base) ne sont généralement pas parfaitement adaptées à votre cas d'usage. Bien sûr, vous pouvez toujours utiliser vagrant ssh pour installer manuellement les logiciels nécessaires, mais l'utilisation du système de provisionnement intégré à Vagrant rend le processus **automatique, reproductible et sans intervention humaine**. Ainsi, vous pouvez exécuter un simple vagrant destroy suivi d'un vagrant up pour obtenir un environnement de travail entièrement opérationnel en une seule commande. Puissant, n'est-ce pas ?

Vagrant propose plusieurs méthodes de provisionnement, allant de **simples scripts shell** à des outils de **gestion de configuration complexes et standardisés** utilisés dans l'industrie (ansible, jenkins…)

Si vous débutez avec les systèmes de gestion de configuration, il est recommandé de commencer par des scripts shell simples afin de vous familiariser avec le concept de provisionnement.